© 2015-2020 by Erlang Solutions Limited, all rights reserved

All material in these pages, including text, layout, presentation, logos, icons, photos, and all other artwork is the Intellectual Property of Erlang Solutions Limited, unless otherwise stated, and subject to Erlang Solutions Limited copyright.

No commercial use of any material is authorised without the express permission of Erlang Solutions Limited. Information contained in, or derived from these pages must not be used for development, production, marketing or any other act, which infringes copyright.

This document is for informational purposes only. Erlang Solutions Limited makes no warranties, express or implied, in this document.
# Table Of Contents

Table Of Contents .......................................................................................... 1

What is WombatOAM? ....................................................................................... 7

Getting started with WombatOAM ................................................................. 10
  Prerequisites .......................................................................................................... 10
  Installing WombatOAM .......................................................................................... 10
  Upgrading WombatOAM ......................................................................................... 10
  The WombatOAM web dashboard ......................................................................... 11
  Running WombatOAM ............................................................................................. 11
  Using WombatOAM .................................................................................................. 11
  Installing WombatOAM in multi-server environment ........................................... 11
  Configuring WombatOAM for servers and containers .......................................... 12
  Mounting volumes in containers .......................................................................... 12
  Using Wombat in Amazon AWS ......................................................................... 12
  Configuring networking in containerised environments ....................................... 12
  Installing WombatOAM on Windows .................................................................... 13
  Troubleshooting ..................................................................................................... 13

Exploring and understanding WombatOAM ................................................. 14
  Basics ...................................................................................................................... 14
  Further exploration ............................................................................................ 22
  More stories from the front line ........................................................................... 40
  Conclusion ............................................................................................................ 42

Authentication and Authorization in WombatOAM ...................................... 44
  Default user .......................................................................................................... 44
  Users, roles and rights .......................................................................................... 44
  Changing a user's role and rights ......................................................................... 45

WombatOAM Dashboard .................................................................................. 46
  Topology ................................................................................................................. 46
  Metrics .................................................................................................................... 46
  Notifications ......................................................................................................... 46
  Alarms ..................................................................................................................... 46
  Front page .............................................................................................................. 47
  Tools ....................................................................................................................... 47

Alarms .................................................................................................................. 48
  Managing alarm messages .................................................................................... 48
  Built-in alarms ....................................................................................................... 48
  Threshold based alarms ....................................................................................... 57
  Notification based alarms .................................................................................... 59
  Email configuration ............................................................................................... 60
  Email alarm notifications ..................................................................................... 60

Notifications ...................................................................................................... 61
  Viewing notifications ............................................................................................ 62
  Mailing notifications ............................................................................................ 62
  Built-in notifications ............................................................................................ 62
  Configuring logging ............................................................................................... 64

Metrics .................................................................................................................. 64
  Viewing metrics ..................................................................................................... 65
  Built-in metrics ..................................................................................................... 65

Services ................................................................................................................ 78
  Configurator .......................................................................................................... 78
  Explorer .................................................................................................................. 80
  Executor .................................................................................................................. 82

Role based information retrieval ....................................................................... 83
  Built-in tags ............................................................................................................ 84
  Custom tags .......................................................................................................... 85

Configuration ....................................................................................................... 86
  wombat.config and sys.config .......................................................................... 87
  Changing WombatOAM's node name and IP address ......................................... 87
  Local ports used by WombatOAM ....................................................................... 88

---

rabbittmq queues plugin ................................................................. 191
rabbittmq queue messages plugin .................................................. 193
RabbitMQ Recovery from netsplit: .................................................... 195
riak_core plugin .............................................................................. 196
riak_kv plugin ............................................................................... 198
riak_repl plugin .............................................................................. 198
sasl plugin ...................................................................................... 200
worker_pool plugin ......................................................................... 200
yokozuna plugin .............................................................................. 200
MongooseIM plugins ......................................................................... 202
Poolboy plugin ............................................................................... 204
Ecto plugin ...................................................................................... 205
telemetry plugin .............................................................................. 205
SumoDB plugin ................................................................................. 206
Yaws Plugin ...................................................................................... 207

WombatOAM Orchestration .............................................................. 209
Orchestration v1 and v2 ................................................................. 209
Configuring WombatOAM Orchestration ......................................... 209
Using WombatOAM Orchestration ................................................... 210

RESTful API .................................................................................... 211
Authentication and user management REST interface ...................... 212
Get the authentication status .......................................................... 212
Get username of the current user ..................................................... 212
Get WebSockets pass of the current user ......................................... 212
Get whether current user is admin .................................................. 213
Get role of the current user ............................................................ 213
Get rights of the current user ......................................................... 214
Set new password for the current user ............................................ 214
Set role of the current user ............................................................ 214
Set rights of the current user ......................................................... 216
Get the list of registered users ....................................................... 217
Add new user .................................................................................. 217
Delete user ...................................................................................... 218

Tags REST interface ........................................................................ 220
Get all available tags ...................................................................... 220
Get the active tags set for a user ..................................................... 220
Change the tags set for a user ........................................................ 221

Topology REST interface ............................................................... 222
Get metadata about nodes ............................................................. 222
Get metadata about node families .................................................... 222
Get more information about a node .................................................. 223
Get more information about a family .............................................. 224
Add new node ............................................................................... 224
Add new node and discover connected nodes .................................. 225
Add new family ............................................................................... 226
Remove node ................................................................................. 227
Remove family ............................................................................... 228
Move a node to another family ...................................................... 228
Analyse reachability of a node ....................................................... 229
Get node statistics ......................................................................... 230

Node information REST interface .................................................. 232
Get system information about a node ............................................. 232
Get application information about a node ...................................... 232
Get module information about a node ............................................ 234
Get the metric limits for a node ...................................................... 235

Alarms REST interface .................................................................... 236
Get all alarms ............................................................................... 237
Get all alarms from a specified ID .................................................. 238
Get alarms for a node .................................................................... 238
Get alarms for a node from a specified ID ....................................... 239
Get alarms for a node family .......................................................... 240
Get alarms for a node family from a specified ID ............................ 240
Get all alarm sources ..................................................................... 241
Get alarm sources for a node ............................................................... 241
Get alarm sources for a node family .................................................. 242
Send acknowledge for an alarm ........................................................ 242
Send unacknowledge for an alarm ...................................................... 243
Send clear for an alarm ................................................................... 244
Send comment for an alarm .............................................................. 245

Logs/notifications REST interface ..................................................... 245
Get all logs ....................................................................................... 246
Get 20 logs from a cursor position .................................................... 247
Get logs for a node ............................................................................ 248
Get 20 logs from a cursor position for a node ................................. 249
Get logs for a node family ................................................................. 249
Get 20 logs from a cursor position for a node family ....................... 250
Search in logs .................................................................................. 251
Get 20 logs from a cursor position for the search query ................... 252
Get all log originators .................................................................... 253
Get log originators for a node ........................................................... 254
Get log originators for a node family ................................................. 254
Get log levels for a node ................................................................. 255
Set log level for current node ........................................................... 255
Get part of a log message ................................................................. 256

Metrics REST interface ................................................................ 257
Get metrics metadata for nodes ....................................................... 258
Get metric metadata for node families .......................................... 260
Get metric historical data for a node .............................................. 260
Get metric historical data for a node family .................................. 262
Get metric histogram for a node .................................................. 263
Get metric duration for a node ..................................................... 264
Get metric meter for a node .......................................................... 266
Get spiral metric for a node .......................................................... 267
Get gauge info for a node ............................................................... 268

Services REST interface .............................................................. 269
Get all configurator services available for a node ...................... 270
Get all configurator services available for a node family .......... 272
Get all explorer services available for a node .............................. 273
Get all explorer services available for a node family ................. 276
Get all executor services available for a node ......................... 277
Get all executor services available for a node family ............... 278
Initiate a new service request ....................................................... 279
Initiate a feature request ............................................................... 279
Stop a running request ................................................................. 281
Initiate new process monitors ....................................................... 281
Get information about a request .................................................. 282
Get all running request per node ................................................. 286
Get all running request per node family .................................. 286
Get all finished requests per node ............................................... 287
Get all finished requests per node family .................................. 287

Orchestration REST interface ....................................................... 288
Get supported providers ................................................................. 289
Get available providers ................................................................. 290
Get information about a provider ................................................. 291
Get available releases ................................................................. 291
Get information about a release .................................................. 292
Get available node families (Orchestration v1) ......................... 293
Get available node families (Orchestration v2) ......................... 295
Get information about a node family (Orchestration v2) ....... 296
Get available domains of a node family ................................... 297
Get information about a domain ............................................... 298
Get available nodes (Orchestration v1) ...................................... 298
Get available nodes (Orchestration v2) ...................................... 299
Get information about a node (Orchestration v2) ................. 300
Register provider ................................................................. 301
Upload release ..................................................................... 302
Create node family (Orchestration v1) ...................................... 303
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create node family (Orchestration v2)</td>
<td>304</td>
</tr>
<tr>
<td>Attach release to a node family</td>
<td>305</td>
</tr>
<tr>
<td>Set firewall configuration for a node family</td>
<td>306</td>
</tr>
<tr>
<td>Set node hooks for a node family</td>
<td>307</td>
</tr>
<tr>
<td>Add deployment domains to a node family</td>
<td>307</td>
</tr>
<tr>
<td>Deploy a node</td>
<td>310</td>
</tr>
<tr>
<td>Upload file to a node (deprecated - only Orchestration v1)</td>
<td>317</td>
</tr>
<tr>
<td>Unregister provider</td>
<td>317</td>
</tr>
<tr>
<td>Delete release</td>
<td>318</td>
</tr>
<tr>
<td>Delete family (Orchestration v1)</td>
<td>318</td>
</tr>
<tr>
<td>Delete family (Orchestration v2)</td>
<td>319</td>
</tr>
<tr>
<td>Terminate family (Orchestration v2)</td>
<td>319</td>
</tr>
<tr>
<td>Terminate a node</td>
<td>320</td>
</tr>
<tr>
<td>Plugins REST interface</td>
<td>320</td>
</tr>
<tr>
<td>Get node plugins state</td>
<td>321</td>
</tr>
<tr>
<td>Set node all plugins state</td>
<td>321</td>
</tr>
<tr>
<td>Set node plugin state</td>
<td>322</td>
</tr>
<tr>
<td>Bookmark REST interface</td>
<td>324</td>
</tr>
<tr>
<td>Create bookmarks</td>
<td>324</td>
</tr>
<tr>
<td>Read all bookmarks</td>
<td>324</td>
</tr>
<tr>
<td>Update bookmarks</td>
<td>325</td>
</tr>
<tr>
<td>Delete bookmarks</td>
<td>326</td>
</tr>
<tr>
<td>Front page REST interface</td>
<td>327</td>
</tr>
<tr>
<td>Get metadata about front pages</td>
<td>328</td>
</tr>
<tr>
<td>Get metadata about front pages owned by a user</td>
<td>328</td>
</tr>
<tr>
<td>Get front page by name</td>
<td>329</td>
</tr>
<tr>
<td>Create front page</td>
<td>329</td>
</tr>
<tr>
<td>Update front page</td>
<td>330</td>
</tr>
<tr>
<td>Delete front page</td>
<td>331</td>
</tr>
<tr>
<td>Front page JSON</td>
<td>331</td>
</tr>
<tr>
<td>System configuration REST interface</td>
<td>332</td>
</tr>
<tr>
<td>Get default values from sys.config</td>
<td>333</td>
</tr>
<tr>
<td>Help/about REST interface</td>
<td>333</td>
</tr>
<tr>
<td>Get license text</td>
<td>334</td>
</tr>
<tr>
<td>WombatOAM debug script</td>
<td>335</td>
</tr>
<tr>
<td>WombatOAM reference</td>
<td>335</td>
</tr>
<tr>
<td>Logs</td>
<td>336</td>
</tr>
<tr>
<td>WombatOAM data</td>
<td>336</td>
</tr>
<tr>
<td>Managing older Erlang nodes</td>
<td>336</td>
</tr>
<tr>
<td>Multiple WombatOAM instances</td>
<td>336</td>
</tr>
<tr>
<td>Gauges</td>
<td>336</td>
</tr>
</tbody>
</table>
What is WombatOAM?

WombatOAM is an operations and maintenance tool for proprietary and open source Erlang and Elixir systems. All of the generic functionality to monitor and manage scalable, highly available systems, often only in part reimplemented from project to project, is now available in a battle-proven standalone solution. Using WombatOAM allows you to focus on your business logic, while reusing thousands of lines of code and tapping into decades of operational experience of Erlang-based systems.

WombatOAM's functionality is divided into two categories:

- **WombatOAM Monitoring**: Monitoring nodes, including collecting and displaying metrics, alarms and logs from the managed nodes. WombatOAM also contains tools that allow you to inspect and troubleshoot your system, as well as APIs that you can use to hook into your existing OAM and SAAS tools.

- **WombatOAM Orchestration**: Deploying nodes in the cloud or on specific proprietary clusters.

Orchestration is currently in Beta. We are looking for users interested in working with us while we bring it to R1.

The WombatOAM server consists of one or more standalone nodes that connect to a system running on the Erlang VM using Distributed Erlang. The system consists of one or more Erlang VMs, possibly running different releases. The nodes in the system WombatOAM is connected to are called the managed nodes. They could be a proprietary application, or standalone open source applications such as CouchDB, Riak, RabbitMQ or Phoenix. Depending on the OTP applications that are running on the managed nodes, WombatOAM starts non-intrusive agents that enable it to monitor alarms and notifications, and collect information such as metrics and logs. This agent code is loaded on the fly, without having to include any proprietary WombatOAM applications in your release and without the nodes having to be restarted. WombatOAM will connect to them seamlessly – even if they have been running for years, or are running older versions of the Erlang VM.

The plugins used by WombatOAM have been optimized to reduce overheads caused by the agents monitoring the managed nodes. Overheads vary between 0.5%–1.5% of the total CPU cycles used by the Erlang VM, depending on how many applications you are monitoring and the underlying hardware, operating system, and possible virtualization layers. The performance of the business logic running in the managed nodes will barely be affected by the minimal overhead.

The WombatOAM web dashboard presents information to the operator in a manageable way. The dashboard is intended for use by DevOps teams when troubleshooting Erlang systems and for companies who have not integrated their Erlang systems to their existing OAM infrastructure. WombatOAM can also share this information with other third-party OAM tools, acting as a hub towards Nagios, Cacti, Graphite, Grafana or SAAS providers such as Splunk or PagerDuty (to mention just a few). As a result, it provides a single point of integration with the wider OAM infrastructure and SAAS providers without the need to upgrade and manipulate production code running in the managed nodes.
• Autodiscovery of your node topology. Provide a node and its cookie, and WombatOAM will discover all existing nodes in your cluster, group them into releases (called node families), and start monitoring them. WombatOAM supports both long and short names, and can currently monitor the R14 Erlang runtime alongside later versions. Versions older than R14 can also be supported on demand.

• Anomaly detection and early warnings, presented in the form of alarms and notifications. This allows DevOps teams to address and resolve problems before they escalate and cause service disruption. Integration has been made with the SASL alarm handler and the Elarm application, forwarding alarms specific to your system. SASL and Lager logs, including crash, warning and error reports, are also forwarded, giving you access to data specific to all your clusters in one place.

• Automatic collection of more than a hundred built-in metrics from the Erlang runtime system, including different memory types, system limits, socket, port and process-specific metrics. Additional metrics retrieved from plugin modules of other supported OTP applications are also uploaded. Metrics from Folsom and Exometer are collected seamlessly if the managed nodes are using them to generate business metrics.

• Application-specific plugins that run on the managed node and send metrics, notifications and alarms to WombatOAM. A set of built-in plugins covering many standard OTP applications and popular open source ones are shipped with WombatOAM. They can be turned on and off for each individual managed node. You can also implement proprietary plugins following a simple API.

• Plugins have been implemented to monitor SASL, Lager, Folsom, Exometer, Elarm, Mnesia, OSmon and Cowboy OTP applications. For Riak, WombatOAM monitors Riak Core, key value (Bitcask and Active Anti-Entropy), replication services and Yokozuna. Many more plugins are being implemented, and you can write your own following a simple API.

• Seamless integration with your OAM infrastructure by having existing integration plugins with Graphite, Grafana, Cacti, Graylog, Splunk, Zabbix, Datadog, Nagios, Logstash, AppDynamics and PagerDuty. If you are using proprietary OAM tools or SAAS providers currently not supported, the WombatOAM architecture provides flexibility to add integration points without the need to upgrade or restart your managed nodes. Integration happens in the WombatOAM node, reducing overhead in the managed nodes while reducing the risk of OAM-related issues escalating in the nodes managing the business logic.

• A Web Dashboard and a REST interface, the former for interactive use if you need all of the information in one place, the latter to support automation and integration with your existing tool chain and scripts.
- Visualization of the node topology and the ability to inspect specific node information, which facilitates troubleshooting. WombatOAM also acts as a point of integration of tools that allow closer inspection of what is happening in the managed nodes and facilitate troubleshooting, without the need to access the Erlang nodes and shell.

- The dashboard can plot both historic and live metrics, showing you memory usage in real time, or helping you detect spikes no one would have noticed otherwise. Multiple metrics can be shown on the same graph for comparison purposes. Besides numeric metrics such as counters and gauges, WombatOAM also supports meters, spirals and histograms.

- Deployment of Erlang nodes in heterogeneous clouds or on specified machines that scale to tens of thousands of nodes with no single point of failure. WombatOAM orchestration and monitoring has been tested on a cluster of 10,000 Erlang VMs, but is linearly scalable beyond that.
Getting started with WombatOAM

Prerequisites

Before installing WombatOAM, ensure that the machine meets the prerequisites:

- A UNIX operating system (WombatOAM has been tested on Linux, OS X)
- The Erlang run-time system, ERTS (R15B03 or later)
- OpenSSL and libcrypto (already present on Mac OS X)

WombatOAM needs a license key to work. The license key can be changed any time by replacing the license key file and restarting WombatOAM. The license file contains how many nodes can be managed by WombatOAM and it includes an expiry date. Once WombatOAM is installed, you can check the details of the license by clicking on "Help" on the WombatOAM Dashboard.

Installing WombatOAM

The fastest way to install WombatOAM on your computer is from a package. Execute the following commands:

1. Extract the WombatOAM package:
   
   ```
   tar xzf wombat-[VERSION].tar.gz
   cd wombat-[VERSION]
   ```

2. Copy your license key into the WombatOAM root directory:
   
   ```
   cp .../my.lic_key .
   ```

3. Install WombatOAM:
   
   ```
   ./wombat install
   ```

Upgrading WombatOAM

To upgrade an existing installation of a previous version of WombatOAM:

1. Extract the WombatOAM package:
   
   ```
   tar xzf wombat-[VERSION].tar.gz
   cd wombat-[VERSION]
   ```

2. If you would like to use a new license key, copy it to the new WombatOAM installation's root directory. (If the version of the old WombatOAM installation is 2.0.0-rc1 or earlier, this is necessary.) If you would like to use the same license key as in the old installation, then skip this step.
   
   ```
   cp .../my.lic_key .
   ```

3. Execute the following command:
   
   ```
   ./wombat upgrade
   ```

   You will have to input the previous Wombat installation directory.

4. The script will perform the upgrade and after running it you can start WombatOAM.
The command `./wombat upgrade` will preserve your collected data and custom configuration (in your old 
_build/default/rel/wombat/files/wombat.config file). It will copy the old license from the old installation. Backups are created to the backups directory from the current installation.

The WombatOAM web dashboard

To access the web dashboard, go to http://localhost:8080 in your browser. Log in with the following details:

- **Name**: admin
- **Password**: admin

See the Configuration page for further information about configuring the web interface (for example to enable SSL).

Running WombatOAM

You can start and stop WombatOAM with the following commands:

```
./wombat start  # start WombatOAM
./wombat stop   # stop WombatOAM
```

You can also use the script `bin/wombat`, located in the _build/default/rel/wombat/bin/wombat folder. This script provides additional options for interacting with WombatOAM but you will have to configure Wombat through environment variables. Please refer to the `./wombat` script or use it through `./wombat start_script`. You can find more information in the Configuring WombatOAM for servers and containers chapter.

To check whether WombatOAM is already running, navigate your browser to the WombatOAM web dashboard on http://localhost:8080.

Using WombatOAM

To see how WombatOAM handles nodes, you can install the WombatOAM node itself into WombatOAM:

1. Select the **Topology** tab and click **Add Node**.
2. Enter the following:
   - **Node name**: wombat@127.0.0.1
   - **Cookie**: wombat
3. Click the **Add node** button. The WombatOAM node should come up in a few seconds. If it doesn't, check the following log file for details: _build/default/rel/wombat/log/wombat.log

Click the **Metrics** tab, select the node, and then select a metric. For example, click **Memory → Total memory**. You should see the measurements made on that node. For more information about the different metrics, see the section "Metrics".

Installing WombatOAM in multi-server environment

After you tested WombatOAM locally, you want to install it in a multi-server environment (monitoring test or production servers). The steps are very similar to what is described in *Installing WombatOAM* above but before starting Wombat make sure to set the WombatOAM node name to match the server's host name or IP address. This will switch WombatOAM to run in Distributed security mode (as opposed to Local-
only security mode) which requires some further configuration changes to the default config. See the sections on Node name and Security modes in the Configuration chapter for details.

The default naming and structure for nodes and families can also be customized. For further information see the section Custom topology.

**Configuring WombatOAM for servers and containers**

It is possible to configure WombatOAM through environment variables. The available options are the following:

- **WOMBAT_NODENAME**, the erlang node name of the node, by default it is `wombat@127.0.0.1`. Inside containers it is recommended to configure this to a static value because mnesia relies on the node's name being always the same.
- **WOMBAT_COOKIE**, the cookie of the node, by default it is `wombat`.
- **HEART**, enable automatic restart of Wombat in case of failure using Heart, by default it is `false`. It is not useful to use this option inside containers because when Wombat stops in case of error the container will stop as well.

It is recommended to use a separate (virtual) network for Wombat and its managed nodes.

**Mounting volumes in containers**

It is necessary to provide persistent storage for WombatOAM if you want to keep historical data. The following directories can be mounted in the provided docker image:

- `/wombat/data` -- The data directory of Wombat
- `/wombat/log` -- The logs generated by Wombat
- `/wombat/wombat.lic_key` -- Wombat will look for the license in its root directory, it is advised to mount it if you are using containers.
- `_build/default/rel/wombat/files/wombat.config` -- Wombat config file. Place your changes to the default config into this file.

**Using Wombat in Amazon AWS**

We build AMIs from Wombat. Please get in touch at wombat@erlang-solutions.com if you have no access to them.

For configuring the AMI please check the 'Using the official AMI containing WombatOAM' section.

**Configuring networking in containerised environments**

Wombat will only be able to connect to containers where it can resolve the hostname to an IP and that IP is accessible. It is necessary to place every monitored container into a network which is available for Wombat.

If you are using docker-compose then all containers in the same file will be in the same network by default, this is not true for containers started by the `docker run` command. If you started the containers with `docker run` then it's necessary to create a network and attach the containers using `docker network`. If you are using `docker stack` then an overlay network is needed to be configured.

In case you are using Amazon ECS it is possible to start the containers in the same network by configuring `awsvpc`.
Installing WombatOAM on Windows

Currently on Windows only running it inside Docker is supported. Please get in touch with us if you need Windows support.

Troubleshooting

Cannot connect to WombatOAM remotely

WombatOAM is configured in Local-only mode

Please configure WombatOAM to run in 'Distributed' security mode. Follow the instructions in WombatOAM Manual (chapter Configuration, section Security modes) to switch to Distributed security mode.

WombatOAM is configured in Distributed mode

Please make sure that if there's firewall configured on the host where WombatOAM is running, the HTTP and HTTPS ports (by default TCP 8080 and TCP 8443, configurable by the wo_rest, http_port and wo_rest, https_port settings) are not blocked. In particular, the default CentOS 7 installation contains a firewall configuration that blocks incoming connection to these ports. Please check the documentation of your operating system on how to enable access to these ports.

WombatOAM is not started after running the start script

It is possible that WombatOAM does not start if the path to the application or the erlang system library contains spaces or other special characters. A solution to this issue is to remove the spaces or special characters from the paths or use the provided Docker container to run Wombat.
Exploring and understanding WombatOAM

"I have installed WombatOAM, now what?"

This chapter gives you a walkthrough of WombatOAM, demonstrating some of the issues you might encounter in your Erlang and Elixir based systems. It highlights how WombatOAM provides full visibility of what is happening, helping you with pre-emptive support in order to avoid outages, and also with post-mortem debugging to quickly and efficiently find and fix problems, ensuring they do not happen again. Besides, WombatOAM is handy for troubleshooting, online diagnosing and resolving problems and incidents on the fly. This walkthrough should take you about 30 minutes, and is recommended for anyone evaluating or trying to get the most out of WombatOAM.

Basics

To get started, go to the WombatOAM dashboard and click **Topology**. Click **Add node**, and enter a node name and cookie.

If you are using distributed Erlang in your cluster, select **Discover connected nodes** to automatically add nodes meshed with the one you are adding, to save you the effort of doing it later.

When you have entered the node name and cookie, click the **Add node** button. This will connect WombatOAM to your cluster, and allow it to start monitoring the nodes.
If you are using multiple cookies, add a node from each cluster using the specific cookie. Note that WombatOAM allows you to connect to nodes with long and short names.

During this walkthrough, you will experiment and simulate a few system issues. If you are connected to a production system and are worried about the impact of this, you can also start and connect to a standalone node. To do this, enter the following command:

```
erl -sname test -setcookie abc123
```

Once you have started the node, add it to WombatOAM (the node name is "test" and the cookie is "abc123"). Ensure that you remain connected to the node in the shell. If you need to stop and restart the node later, you can use the same command provided above.

Depending on what applications you are running, WombatOAM will start agents that collect metrics, alarms, notifications, other important information about your nodes and also provide tools. Tools are not metrics, notifications or alarms, but aid devops with their daily tasks. Have a look at what these agents do, as you can easily write your own plugins introducing new agents.

Select the node to see information about the system, modules, applications and the agents that are running.
The agents that come with WombatOAM do the work of collecting and generating metrics, notifications and alarms. Also, they enable retrieving live information about running systems (such as the state of a gen_server), recover from outages for instance by forcing the GC to free up memory, or fix misconfiguration issues by changing configuration parameters. A basic installation (excluding the application-specific agents listed below) will include:

- More than 100 metrics and the automatic collection of Exometer and Folsom metrics
- Automatic collection of Lager and SASL logs
- Over 25 built-in alarms and automatic collection of elarm and SASL alarms
- Configuration management
- Etop-like process listing
- ETS table viewer
- Various process inspectors
- Executor for user commands and other actions such as initiating garbage collection

We currently have agents supporting the following applications:

- Kernel
- OS_Mon
- SASL
- Lager
- Exometer
- Folsom
- Cowboy (versions 0.x, 1.x)
- Phoenix
- Ecto
- Poolboy
- MongooseIM
- RabbitMQ
- Mnesia
- Riak Core
- Riak KV
- Riak Multi-Datacenter Replication
- Yokozuna
- SumoDB
- Yaws

Explore some more by clicking the Topology tab. Here you can view comprehensive system, application and module information.
This information can be useful for troubleshooting your system. For example:

- Module information: one of your nodes may be using an older version of a module
- Application information: an application running on one of your nodes may be using a wrong value (such as an old IP address)
- System information: check which release is currently used by the node

To see how WombatOAM monitors your systems, let’s use the shell to simulate problems that might occur. If you have a node in your system that you can allow to crash, connect to it. Otherwise, use a standalone test node.

If you have started a separate node, add it to WombatOAM. Ensure that you are connected to the node in the shell, then enter the following command:

```
1 2  put(sample_key, value), [ self() ! {count, Int} || Int <- lists:seq(1,100000) ].
```

This will associate value with sample_key in the shell process's local store called the "Process Dictionary" and then send 100,000 messages to the shell process, simulating producers sending requests at a faster rate than the consumers can handle. Long message queues are often a symptom of a bottleneck and result in the degradation of throughput, with the system eventually running out of memory.

First, retrieve live information about your running processes. In WombatOAM, go to **Tools**. Select your test node and then open the **Process Manager** panel.

Notice that the shell process has the largest message queue.
By clicking on the Pid of the shell process, various information, such as process info, process messages, process dictionary, process state, process stack trace) can be retrieved about this process. These come in handy when your system doesn't behave as expected and you want to find the root cause.

Next, go to **Metrics**, select the node to which you issued the command, select **Memory**, and then select **Total memory** to view that metric as a graph. (Note that metrics are collected once a minute, so you may have to wait briefly for the graph to update.)

Notice how sending the messages affected the total memory increase.

Select **Process memory** used to see how process memory affected the total memory.
Select the Alarms tab. Note the new **process_message_queue_major** alarm with severity **major** that will appear.

Go back to Metrics and clear all the metrics (click on the “trash” button at the upper right of the graph). Select the test node and click **Processes → Sum message queue length** to see all the messages you sent to the shell.
Next, take it a step forward, and click **Live Metrics**. Again select the **Sum message queue length** metric. While you keep monitoring the message queue length in real time, enter the following in the shell:

```shell
1
flush().
```

Notice how the graph drops as the queue is being cleared.

When the message queue is cleared, click on **Alarms**. You should see that the alarm has now been cleared. (You may need to give it some time to clear, as checks are made on average once a minute.)
Go to the **Notifications** tab, and you should see notifications from the code tracer and the alarm handler.

All alarms raised and cleared are logged under Notifications, along with all shell interactions, modules loaded and purged, and so forth. You might not be sitting in front of your monitor and miss an alarm being raised and cleared - maybe missing that a message queue grew to 100,000 before being handled! You should check alarm logs regularly, and address them if necessary. Next time, it might happen that the queue length goes far beyond 100,000, causing the node to slow down and eventually crash.

Following these steps should show you how using WombatOAM might let you uncover problems that would otherwise have remained unnoticed. The following story from a WombatOAM customer attests to the importance of this kind of pre-emptive support:

"In one WombatOAM installation, all seemed to run as normal until we looked at the alarm logs. We noticed that twice in three days, three processes had reached a message queue length of 100,000, which got cleared without causing any issues. We narrowed down
the cause to the process acting as a cache when the client connection was dropped for a longer time period. This became only visible in live systems, as client connectivity in the test plant worked well. Changes were immediately needed, as loss of network connectivity or a firewall misconfiguration between the perimeter network and that machine would have caused the connection to all clients to be dropped, resulting in the node running out of memory."
WombatOAM will show that the node is down, and also show an alarm (look out for the popup notification) after a brief moment. After this, restart the node. Again, WombatOAM will automatically notice (within 30 seconds) that it is up again, and the alarm will be cleared.

**Hitting system limits**

**Filling the atom table**

Converting values into atoms without any control can fill the atom table. To try this out, enter the following command in your Erlang shell:

```
[ list_to_atom("atom_" ++ integer_to_list(I)) || I <- lists:seq(1, 964689) ].
```

You should get the following alarm: **atom_limit_major**

This alarm may be an indication of unsatisfactory code that dynamically turns strings into atoms. Since the atom table is not garbage-collected, the only way to recover from this situation is to restart the node.

Converting user inputs into atoms can also easily hit the system limit, since the atom table can only grow. Usually the table grows slowly, so the node doesn’t crash immediately, but only after a long period of time (for instance, after two months).

You should restart the node after trying this out.

**Creating too many ETS tables**

Enter the following command in your Erlang shell:

```
[ ets:new(list_to_atom("_ets_table_" ++ integer_to_list(I))) || I <- lists:seq(1, 1750) ], ets:insert('_ets_table_1', {v1, v2, v3}).
```

You should get the following alarm: **ets_limit_major**
If you don't get the alarm, it may be because your system limits differ from the defaults. Try this alternative instead, as it first determines the system limits:

```erlang
ETSLimit = try erlang:system_info(ets_limit) of
           Limit -> round(Limit * 0.92)
       catch
           error:badarg -> 1900
       end.

ts:insert('_ets_table_1', {v1, v2, v3}).
```

Hitting the limit of the number of ETS tables can happen quite easily, as there is no automatic garbage collection for tables. Even if there are no references to a table from any process, it will not automatically be destroyed unless the owner process terminates. Imagine a permanent server process creating new ETS tables for new sessions. If there is a spike in requests, you might hit the limit. This alarm is also seen when executing many simultaneous mnesia transactions, as each transaction creates a dynamic ETS table.

Now, it is a good time to explore another service provided by WombatOAM. Go to **Tools** and select the node to which you issued the command. Open the **Table Visualizer** panel, change the value of the **Order by column** input field to be **name** and submit the request.

A listings will appear showing the ETS tables with their properties.
Click the `ets_table_1` link in the first row and view its content by clicking View content in the local menu that appeared. Notice that the displayed content is the same what we inserted into this table.

Go back to the Table Visualizer tab, and click the pid of the owner process which created these ETS tables. A local menu with the same content we saw at Etop has appeared. Click Terminate process that will kill your shell process and therefore all ETS tables created by this process will be deleted. Within a minute, the corresponding alarm will be cleared automatically.

**Process limit**

The maximum number of simultaneously alive Erlang processes is by default 32,768 (this limit can be configured at startup). To trigger an alarm where the number of processes approaches this limit, enter the following commands into your shell:

```
1    Shell = self().
2    [ spawn(fun()-> receive after 120000 -> Shell ! I end end) || I <- lists:seq(1, 30147) ].
```
If you don't get the alarm, it may be because your system limits differ from the defaults. Try the following alternative, which determines the system limits first:

```erlang
ProcessLimit = try erlang:system_info(process_limit) of
  Limit -> round(Limit * 0.92)
catch
  error:badarg -> round(32768 * 0.92)
end.
Shell = self().
[ spawn(fun()-> receive after 12000 -> Shell ! I end end) || I <- lists:seq(1, ProcessLimit) ].
```

Within about a minute, the following alarm should be raised:

```
process_limit_major
```

Spawning a process is very easy in Erlang, and serving each request by a new process increases the system's throughput. However, the system limit will be reached if the processes can't terminate as they are waiting for a resource, or the number of arriving requests is much larger than expected.

Had this alarm been monitored, we would probably have gotten an early warning that the following node's process limit had to be reconfigured:

```
"We were supporting a really old installation of Erlang where the processes limit was set to the default value of a few hundred thousand. A firewall misconfiguration caused all of the client connections to fail. The connectivity towards the node was down for five minutes, resulting in more and more users hitting the retry button. When the firewall configuration was fixed, one of the front-end nodes managing the connectivity was hit with so many requests, reaching the process limit and terminating the node. This caused all the clients to reattempt connecting to the remaining nodes, resulting in a cascading failure where all the nodes went down one by one."
```

---

**Long message queue**
After triggering the previous alarm (process_limit), wait 2–3 minutes. Another alarm should be raised: **process_message_queue_major**

This alarm means that an upper boundary of the message queue length of a single Erlang process has been reached. Large message queues are early warn signs. Generally speaking, a large message queue can indicate that a) your system is not totally well-balanced; b) new tasks arrive to a process in bursts; or c) a process is being stalled because it is waiting for something. If you see no reason for large message queues, or the alarm isn't cleared automatically, you should investigate the issue further to avoid possible outages.

After this, you should clear the message queue alarm. Enter the following in the shell:

```
1 flush().
```

**Learning more**

To find out more about memory and system limits, see the following official Erlang documentation:

http://www.erlang.org/doc/efficiency_guide/advanced.html

**Notifications**

**Log entries**

WombatOAM can aggregate errors and warnings coming from logging applications such as SASL or Lager. You can trigger SASL log entries to see how WombatOAM responds and shows you such events. In the node you are running, enter the following:

```
1 error_logger:error_msg("My error").
```

You will see the **error_logger** notification in the list of notifications for your node:
If Lager is also running on your node, you will see the **lager** notification, too.

Shell commands

As you try out the alarms and notifications suggested in this walkthrough, notice the commands executed in the shell are logged under Notifications. For example, the previous command (for the `error_logger` entry) resulted in the following:

This can be useful simply as a history of the commands you entered; it can also show you what other people may have done when you are
Enabling System Monitor notifications

Go to Live Metrics, select Process notifications, and then select the Busy port, Large heap and Long schedule notifications. (You can select the others as well, if you want to.)

In the shell, type the following, and execute it 5 times:

```
```

This sends 200 large strings to the shell process (and the shell will show an impressive block of random text characters). You should see the large heap and long schedule notification counters increase – these are triggered as a result of the process blocking the scheduler, meaning it has been running uninterrupted for a longer time than expected. This command will also increase the heap size of the shell.

System notifications are important to monitor, as they affect the soft real-time properties of your system. If a process spends too long holding a resource, be it the port or a scheduler, the counters are incremented. Counters exist for:

- **Large heap**: Triggered because a process is consuming a large amount of memory. A process on its own will probably not affect the system, but many of them running concurrently might.
- **Long GC**: Triggered because a process spent an unusually long time garbage collecting.
- **Long schedule**: Triggered because a process was not pre-empted, probably because it was running a BIF or NIF, giving it more CPU
time than its peers. This disrupts the soft real-time properties of the system.

- **Busy dist port**: Triggered when the distributed Erlang port is kept busy by a process sending large volumes of data.
- **Busy port**: Triggered when the port is kept busy by a process sending large volumes of data.

Keep an eye on the above metrics. They will tell you if there is an issue with parts of your system, more specifically how you handle your data and memory. The metrics will not tell you where the issue is, just that there might be one. If you suspect that is the case, turn on and receive the system monitor notifications.

You do so by setting some configuration flags. Find the `wombat.config` file in the directory `rel/wombat/files`, and add the following line:

```erlang
{set, wo_plugins, plugins, code_tracer, report_system_monitor_notifs, true}.
```

When you have done this, you need to stop and restart WombatOAM. On the command line, go to the WombatOAM root directory (you don't need to start a shell) and enter the following:

```bash
./stop.sh
```

Wait for "ok", and then enter the following:

```bash
./start.sh
```

(You'll be able to do this from the dashboard in an upcoming version of WombatOAM.)

**Note:** After completing your tests, you may want to turn these notifications off again. We have worked with systems that, under heavy load, generated millions of log entries per day! To turn off the notifications, remove the line that you added earlier from the `wombat.config` file, and then stop and restart WombatOAM again.

With the System Monitor notifications enabled, re-enter the following command in your shell:

```erlang
```

After your command has run for a while, you should see the following `code_tracer` notification: **System monitor: Large heap**
Also, two new alarms have appeared, namely, `system_memory_high_watermark` and `process_memory_high_watermark`, telling you that your system uses too much memory.

A possible solution for such a problem is to force garbage collection on the node. To do so, go to **Topology** → select the node with high memory usage → **Home**. Customise the garbage collection in **Garbage Collect Processes** panel and submit the request.
The garbage collection will be carried out on all processes of the node freeing up memory.

Now as the node is using less memory both alarms should have been cleared.

After completing this test, enter flush() to clear the messages sent to the shell.

“We once spent three months soak testing a system which, contractually, had to run for 24 hours handling 15,000 operations per second sustained. Each operation consisted of an average of 4 http requests, 7 ETS reads, 3 ETS writes and about 10 log entries to file, alongside all of the business logic. The system was running at 40% CPU with plenty of memory left over on each node. After a random number of hours, nodes would crash without any warning, having run out of memory. None of the memory graphs we had showed any leaks, and even if we were polling at 10-second intervals, about 400 MB of memory was still available in the last poll right before the crash. We suspected memory leaks in the VM, runaway non-tail recursive functions, reviewed all the code and ended up wasting a month before discovering system monitors (which at the time were hidden at the bottom of a module description in the documentation). Eventually we turned on the system monitor and noted that a few seconds right before the crash, an unusually high number of long garbage collection and large heap trace events were generated. These were connected to the creation of a session, where an XML file sent back to us with session data caused a huge memory spike when parsed. We were seeing memory spikes when plotting our graphs, but did not think much about them because they were contained. What happened in the run-up to every crash was a surge in session initialization requests, causing these spikes to pool together and create a spike in memory usage which caused the VM to run out. We eventually discovered that the more cores we used increased the probability of this monster wave happening. In less than half a second, this memory surge used up all available memory and caused the node to crash. We estimate this particular issue in the 3 months of soak testing (as each crash took up to 20 hours to reproduce) and kept two people busy for a whole month trying to figure out what happened. Had the system monitor been there from day one, we would have saved three months of trouble shooting and optimizing, and solved the problem in no time.”

Loading a new module

Loading a module will produce a notification. You can try this out by creating a simple module (that doesn't do anything!) and loading it using the shell. In the directory where your node is running, create a text file named `my_mod.erl` with the following content:

```erlang
-module(my_mod).
-export([f/0]).

f() ->
  ok.
```

In the shell, enter the following, which both compiles and loads the module:

```
c(my_mod).
```

This will trigger the following code_tracer notification: **Module checker:**

**Module loaded: my_mod**

---

Bonus: old_code alarm

Following the preceding steps you took to compile and load a module, you can trigger an old_code alarm. Note that the code_tracer configuration should still be enabled for this.

Stop your test node and start it again. Enter the following to load the module once more:

```
l(my_mod).
```

Then enter the following to compile and load it again:

```
c(my_mod).
```

This should produce an old_code alarm, which indicates that a process or a user (via a shell) loaded a new version of a module on the node.
This is quite normal during a hot code upgrade or on a development node that is configured to recompile modules when the source code is changed, but in a production environment it is likely to indicate a problem either in configuration or in deployment management.

While it useful to apply patches manually to keep control of the process, automating system deployments by using scripts is less prone to errors. Nonetheless, scripts can also fail, so if you suspect that some nodes are using different versions of the same module, check the MD5 hashes or the used compilation options hold for the module loaded on those nodes. You can use WombatOAM to easily retrieve this information: Under **Topology**, select the node or node family, click **Modules**. WombatOAM shows detailed information about all the modules loaded on your system. You can find a specific module by using the search box at the top.

When the old version of the module is no longer in use, there is no need to keep it loaded in. To get rid of such old versions, go to **Topology** and select the node having old code → **Home**. In the **Soft purge modules** panel submit the request. Executing this request is safe, as only the old version of those modules are purged that aren't referenced by any process.

As the result, the old version of my_mod was successfully purged, which
cleared the old_code alarm automatically.

different_application_versions alarm

It isn't only the presence of different module versions that can lead to partial service degradations. If different application versions are running on some nodes of a cluster, users are also likely to experience problems.

"I was visiting a customer trialling WombatOAM to show how they can totally exploit what WombatOAM provides. First, I noticed that they have an active alarm. Going into the Alarms page, I saw the different_application_versions alarm relevant for their production Riak cluster had been raised. Having described the alarm, which is raised if some nodes belonging to the same cluster use different version of a certain application, they immediately questioned the devops. The devops were saying for sure that it couldn't happen to their production cluster, but they left us alone to double check. While the devops were away we were exploring the Notification's features. Browsing the collected log entries from the Riak cluster we noticed that crash reports about calling an undefined function were repetitively generated on some Riak nodes. These nodes were exactly the ones that were running the old version according to the alarm! At that time the devops returned and admitted that they had forgotten to upgrade those Riak nodes and just completed the upgrade process. In evidence, the active alarm got automatically cleared by WombatOAM confirming that the anomaly disappeared."

Crash reports

WombatOAM shows crash reports under Notifications. If you have a few hundred nodes in production, you can conveniently aggregate crashes in one place, or write handlers to forward them to a third-party tool, email them, or automatically raise tickets.

Below are two examples, encountered during heavy soak testing. The first one was collected from SASL:
The following crash report was collected from the lager handler:

```
17:00:23.494 [error] Error in process <0.14437.142> on node 'wo_soak_test_8@10.100.0.132' with exit value: {badarith,{{wo_soak_test_crazy,start_crashing_proc,1},[[file,"src/wo_soak_test_crazy.erl"],[line,52]]}}
```

Consider the following story from a WombatOAM customer:

"We had a client with 200 nodes in production. The only way for them to detect that a process had crashed was to log on to a machine, into the node shell and filter in the SASL report browser. WombatOAM retrieves all of the SASL and Lager reports for you in one place, so you can notice crashes and address them."

**Tools**

Functionalities under Tools are several standalone features, which are not metrics, notifications or alarms, but aid devops with their daily tasks. It is handy for troubleshooting, online diagnosing and resolving problems and incidents on the fly.

You have seen how you can exploit the Etop-like process listing, the ETS table viewer, the various process inspectors.

To experiment with these features, first start two Erlang nodes and add them to WombatOAM.

```
erl -name n1@127.0.0.1 -setcookie abc123
erl -name n2@127.0.0.1 -setcookie abc123
```

**User command executor**

This service allows users to execute arbitrary Erlang expressions on the nodes and the result of the execution is shown to the users.

Go to **Topology** → select the n1@127.0.0.1 node **Home**, and in the **Evaluate Erlang/Elixir expressions** panel. Copy the following Erlang expression into the input field and start the request.

```
application:set_env(kernel, test_config, test_value),
application:get_env(kernel, test_config).
```
This will associate test_value with the kernel application's test_config parameter, and then will return the current value of the same parameter, which will be the result of the request.

Use this service to retrieve information specific to your business logic or adjust your system to the current requirements. However, if there are commands you often use, you may consider adding them as new services. Implementing your own service is easy and will speed up your maintenance processes.

**Configuration management**

Tailoring an application to fit a certain system is achieved by configuring the application. An application can be configured using its *configuration parameters*, which are often referred to as environmental variables or simply envs.

There are two kinds of configuration parameters, namely, a configuration parameter can be local or global. *Local configuration parameters* are specific to Erlang nodes, their values can vary among the nodes. The data's root directory of an Erlang node, the HTTP port on which the Erlang node listens are good examples for local configuration parameters. On the other hand, *global configuration parameters* must have the same value set on all the nodes belonging to the same node family as they describe properties of the overall system. As examples, consider the IP address of the load balancer or the lifetime of sessions.

WombatOAM allows you to manage your configs both on node and on
cluster level. For instance, what we did in the previous example can be done using the node level services.

Also, WombatOAM periodically scans for anomalies among the nodes that should share the values of global configs. It will raise an alarm whenever different values are set to any global configuration or whenever any global configuration is missing from any of the nodes.

To explore this feature, go to **Topology** and select the *node family* to which both test nodes belong → **Configure**. Then, open the **Set application parameter as global** panel, choose *kernel* as the *Application* and choose *test_config* as the *Key* and then submit the request.

This will mark the *test_config* parameter as global on both test nodes. As previously we changed the *test_config* parameter only on one node, a new alarm will be raised because this config is missing from the other node.

To resolve this issue, go to **Topology** → and select the *node family* again → **Configure**. Then, open the **Change global config’s value** panel, choose *kernel* as the *Application*, *test_config* as the parameter name, *new_test_value* into the input field and then click **Add** button. Finally, submit the request.
This will change the config's value on both nodes.

As we managed to set the same value for the config on all the nodes, the values of the config are consistent with each other, thus the alarm got cleared by WombatOAM. Also, notice that all the changes performed by WombatOAM are always recorded as notifications, providing a history available for all team members.

### Overview page

WombatOAM's Overview page gives a brief summary about the current state of your system. By default it shows the most helpful metrics, the active alarms and general statistics. It is easily customisable, e.g. graphs with any groups of metrics can be added to the page.

Click the WombatOAM logo in the top left corner to explore the Overview page.
More stories from the front line

In the following stories, we'll see how WombatOAM addresses real-life problems that customers have encountered.

**From a major mobile network operator**

"I had a node crashing and restarting over a 3 month period at a customer site. Some refactoring meant they were not handling the EXIT signal from the ports we were using to parse the XML. Yaws recycled processes, so every process ended up having a few thousand EXIT messages from previous requests which had to be traversed before the new request could be handled. About once a week, the node ran out of memory and was restarted by heart. The mailboxes were cleared. Our customers complained that at times, the system was slow. We blamed it on them using Windows NT. We saw the availability drop from 100% to 99.999% (The external probes running their request right during the crash or when the node was restarting) about every 4 weeks. We rarely caught this issue as external probes sent a request a minute, took half a second to process, whilst the node took 3 seconds to restart. So we blamed it on operations messing with firewall configurations. With triple redundancy, it was only when operations happened to notice that one of the machines running at 100% CPU that we got called in. Many requests going through the system, I thought, but it was only 10 requests per second. Had we monitored the message queues, we would have picked this up immediately. Had we had notifications on nodes crashing, we would have picked up after the event."

WombatOAM would have raised three alarms as a result of the above issue: an alarm when the message queue of a process exceeded certain thresholds, another when it reached a high memory utilisation, as well as one when the node was down. Even if alarms were cleared, following up on them would have lead to the memory metrics, showing the increase in memory usage; more specifically, in the process memory usage. This would have led us to look at the other process related metrics, spotting that the message queue was growing out of hand.
From an enterprise Riak user

"A customer having a Riak cluster with 5 nodes reported that one Riak node crashed without any early warning signs. The customer reported that no maintenance activities had been performed on that node before the crash occurred. After long hours spent on investigating the issue, ESL pointed out that the root cause was a recent configuration change that increased the number of concurrent hand-offs 50 times larger than its default on the bad node. This change allowed the node to accept so many transactions that made the node totally overloaded. After reverting this change the cluster started functioning properly again."

Here, WombatOAM would have logged a notification and raised an alarm: a notification about the configuration change, and an alarm about the inconsistency of the global config's values. Using the configuration management service, the misconfiguration would have been fixed quickly.

Runaway modules

"Someone tried to patch a module, loaded it, saw that it did not fix the bug and deleted the beam file, not knowing he had to purge the module as well. An engineer wasted a whole week figuring that one. That is why we have an Alarm if nodes of the same type run different modules."

If more than one node of the same type existed, WombatOAM would have raised an alarm that the two nodes were running different versions of a module with the same name. To find out how that came about, you would have looked at all of the code related notifications on that node, and finally, narrowed it down to the shell command.

File sanity check

"There was a software upgrade in real-time which required upgrades of the Erlang Environment variables. Operations upgraded the sys.config file, copying and pasting from a word document, invisible control characters included. Months later, a power outage caused the nodes to be rebooted. But because of the corrupt sys.config file, they would not start. The error messages in this case were so cryptic and the control characters in the sys.config file not visible, it took us a few hours to find out what the issue was and restore the service."

WombatOAM regularly checks the sanity of all of the files needed at startup. This includes the boot, app, config and others. If any of them are corrupt and will prevent your system from restarting correctly, WombatOAM will raise an alarm.
Conclusion

This walkthrough shows how WombatOAM collects many metrics, notifications and alarms from managed Erlang nodes. The dashboard helps to visualise this information, with graphs that can be placed on top of each other, and notifications and alarms that can be filtered. When the granularity of the collected information is not enough, you can zoom in and use Live Metrics to see the value of metrics at each second, or the explorer services to online debug your system.

WombatOAM monitors the managed nodes for signs of possible future problems – processes having long message queues, having too many ETS tables, reaching the process limit, filling the atom table, the Erlang node being down, and so forth – and generates alarms from these. It also generates notifications for events that happen within a node, such as commands typed in the Erlang Shell, modules that were loaded, and processes that crashed, thereby giving a history of events on the node. Alarms and error log entries raised by the application will also appear as alarms and error notifications in WombatOAM. All are useful to improve your system by pointing out the weakest part, and then Services can help you to further narrow the case. Also, when there is an on-going outage Services will help you to recover from it immediately.

While Services allow you to inspect and change your system, metrics, alarms and notifications are useful both for detecting early signs of errors that might cause future outages, and investigating past incidents so that you can make sure they don't happen again.

What’s next

WombatOAM has a plugin system that lets you write your own plugins, which can collect metrics, notifications and alarms specific to your business logic, and provide new services best fitting to your daily routine. For example, if an application has its own API for serving metrics, a plugin can use this API and provide the metric values to WombatOAM. Or if the user wants to generate a notification each time a certain function is called, a plugin can subscribe to calls of this function and generate the appropriate notifications.

WombatOAM has integration capabilities with other OAM systems, meaning that the information collected by WombatOAM can be pushed into these tools. Supported OAM systems include Graphite, Grafana, Cacti, Graylog, Splunk, Zabbix, Datadog, Logstash, Nagios, AppDynamics and PagerDuty, and because WombatOAM exposes the collected information via its REST API, integration with other OAM systems is also possible. If you already use OAM systems, then hooking WombatOAM into your existing infrastructure is a logical next step, since it would allow you to view the new information via the usual channels.

After a few days of having a system monitored by WombatOAM, it is worth analysing the information collected:

- Look at the active alarms and try to uncover their reasons. Do the same with notifications and cleared alarms.
- Look at the error and warnings notifications, including crash logs.
- Go through the metrics to see if there is anything suspicious, such as strange spikes in memory metrics and the message queue length metric. Going through the metrics will also help to establish a baseline level for them, so that anomalies can be spotted in the future.
- Check the Process notification metrics, such as Busy port and Busy dist port. Numbers significantly higher than 0 indicate possible performance problems.

You may be surprised at what you discover.
Authentication and Authorization in WombatOAM

WombatOAM ships with built-in user management functionality. This can be disabled in the configuration file.

WombatOAM Authentication restricts the usage of the WombatOAM Dashboard, the WombatOAM REST interface (including web socket communications) to authenticated users only.

WombatOAM Authorization offers the admins to assign roles and rights to users, who will access different functionality based on their role and rights.

Default user

By default, there is a single admin user, with the user name `admin` and the default password `admin`. (It is strongly advised to change this default password at first login.)

Users, roles and rights

Each user has one of the following roles.

- **Admin role**: Only the admin users can manage users, add and remove nodes, and deploy nodes with Orchestration.

- **User role**: All new users are granted the privileges of the User role, together with read and write rights. Each user has a subset of the available rights (read, write and execute). It is possible for a user to have all rights, but not possible to have no right at all. The users can perform different actions depending on which rights they have:
  - The read right allows a user to perform read operations that communicate with the plugins on the managed node.
  - The write right allows the user to modify the data inside WombatOAM or affect how WombatOAM collects data from the managed node.
  - The execute right allows the user to run actions that change the manage node's state.

See more details in the table below.

- **Guest role**: Guest users can examine the data inside WombatOAM, but they cannot perform any action that has any effect on a managed node. E.g. they cannot view live metrics, because when a live metric is enabled, the appropriate plugin running on the managed node will start sending the metric data.

All users can access the WombatOAM dashboard and WombatOAM's REST interface. All users can change their passwords via the Dashboard. Usernames and passwords must contain only alphanumeric characters. Passwords are stored encrypted.

The following table describes the permissions needed for different actions:

<table>
<thead>
<tr>
<th>Action</th>
<th>Necessary permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create bookmarks</td>
<td>Available for all users</td>
</tr>
<tr>
<td>Create front pages</td>
<td>Available for all users</td>
</tr>
<tr>
<td>Change own password</td>
<td>Available for all users</td>
</tr>
<tr>
<td>Action</td>
<td>Permission</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Change own tags</td>
<td>Available for all users</td>
</tr>
<tr>
<td>View collected metrics</td>
<td>Available for all users</td>
</tr>
<tr>
<td>View notifications</td>
<td>Available for all users</td>
</tr>
<tr>
<td>View alarms</td>
<td>Available for all users</td>
</tr>
<tr>
<td>View service requests</td>
<td>Available for all users</td>
</tr>
<tr>
<td>View live metrics</td>
<td>Read</td>
</tr>
<tr>
<td>Start and stop explorer services</td>
<td>Read</td>
</tr>
<tr>
<td>Acknowledge, clear, command alarms</td>
<td>Write</td>
</tr>
<tr>
<td>Set log level</td>
<td>Write</td>
</tr>
<tr>
<td>Start/stop plugins</td>
<td>Write</td>
</tr>
<tr>
<td>Run custom commands on the node</td>
<td>Execute</td>
</tr>
<tr>
<td>Start and stop executor and configurator services</td>
<td>Execute</td>
</tr>
<tr>
<td>Add and remove nodes and node families</td>
<td>Admin only</td>
</tr>
<tr>
<td>Deploy nodes via WombatOAM Orchestration (beta)</td>
<td>Admin only</td>
</tr>
<tr>
<td>Adding and deleting users</td>
<td>Admin only</td>
</tr>
</tbody>
</table>

### Changing a user's role and rights

Users can be created and their role and rights altered on the WombatOAM Dashboard by clicking on the username and selecting Administration.

The roles and rights can also be changed by modifying the `wombat.config` file and then restarting WombatOAM. Each configuration entry will update the user's permission in the user database when WombatOAM starts.

The following is an example configuration snippet that assigns permissions to several users:

```
{set, wo_rest, users_permission, <<"guest1">>, guest}.  
{set, wo_rest, users_permission, <<"guest2">>, guest}.  
{set, wo_rest, users_permission, <<"dev_team">>, [read]}.  
{set, wo_rest, users_permission, <<"ops_team">>, [read, write]}.  
{set, wo_rest, users_permission, <<"A_team">>, [read, write, execute]}.  
{set, wo_rest, users_permission, <<"superuser">>, admin}.  
```
WombatOAM Dashboard

WombatOAM's monitoring features are accessible from the web dashboard. You can use the dashboard to manage nodes, and view metrics, notifications and alarms. The following is an overview of WombatOAM's monitoring features, with details about how to access these features, where relevant.

Topology

Topology is an entry point for all tools that allow you to inspect and monitor a node as a whole.

- Add nodes: Topology → Add Node. If you select the "Discover connected nodes" checkbox, all nodes that are connected to the given node (including hidden nodes) that have the same cookie will be installed into WombatOAM.
- View node information: Topology → select a node → System.
- View family information: Topology → select a node family → Information.
- Access family configuration: Topology → select a node family → Configure.
- Observe the status of the nodes (UP/DOWN) in the node tree panel. WombatOAM automatically recognizes if a node goes down or if it comes up again.
- View Agents information, agent state, and turn agents on or off.
- View module information.
- View node applications and change the value of configuration parameters.
- Remove nodes or node families: Topology → Node Info → select a node or node family → Remove node or Remove node family.
- View a visual representation of the system: Topology → Node Graph.

Metrics

Metrics are useful for recognizing trends and troubleshooting systems if failure occurs. Nodes are grouped by families, so you can view metrics for a family or for individual nodes. Different metrics can be combined on the same graph.

- View numeric metrics: Metrics → Numeric Metrics → select a node → select a metric. A new data point in the total memory graph arrives every 60 seconds (configurable). If you select a node family, the metrics of the nodes are stacked onto each other.
- WombatOAM has 90 built-in metrics; it also collects folsom and exometer metrics created on the node, as well as metrics from custom plugins, allowing you to combine business and system metrics.
- WombatOAM can push the metrics to other OAM tools. This means that you can integrate WombatOAM metrics with other metrics that aren't Erlang-related. See the "Integration with third-party tools" section.

Notifications

With Notifications, you can view important crash and error reports from all nodes in one place. When a crash occurs, this should be the first port of call. All alarms raised and cleared are also logged under Notifications, along with all shell interactions, modules loaded and purged, and so forth.
• View notifications: **Notifications** → **Manage Notifications** → select a node. WombatOAM displays logs created with `error_logger` or `lager` from the managed nodes.

• Search notifications: **Notifications** → **Manage Notifications** → enter a value in the search box.

• Popup notifications are shown for events that are at "error" level or higher.

• The level of log entries to be collected is configurable.

**Alarms**

You should check alarm logs regularly, and address them if necessary. All alarms raised and cleared are also logged under Notifications, in case you miss an alarm being raised and cleared.

• View alarms: **Alarms** → **Manage Alarms**. This shows the alarms raised by WombatOAM (for example, if a node went down). Alarms can also be cleared by WombatOAM (for example, when a node comes up again).

• Users can acknowledge, unacknowledge, clear, and comment on alarms.

• The web dashboard displays popup notifications for new alarms.

• WombatOAM generates notifications from alarm events (displayed in the **Notifications** tab).

**Front page**

If you click on the WombatOAM icon or WombatOAM text, you will see the front page with selected information (metrics, alarms, node statistics). Users can create their own pages with the following constraints:

• Every page has a name which is unique.

• Users can create, modify or delete pages.

• It is not allowed to modify or delete a page that is not owned by the user, except if it is not owned by anyone.

• The admin user can create, modify or delete any page, but even the admin cannot delete the default page.

• When a user is deleted, the admin can decide whether to delete the user's pages too. If they are not deleted, they will not be owned by anyone.

• If authentication is switched off, users have the same rights as admins when authentication is on.

**Tools**

This page contains the Process Manager and the Table Visualizer tools:

• View the running processes: **Tools** → select a node → **Process Manager**.

• View the ETS tables: **Tools** → select a node → **Table Visualizer**.

• View more information about a process: **Tools** → select a node → **Process Manager** → click on the process pid.

• View the ETS table contents: **Tools** → select a node → **Table Visualizer** → click on the id of the table.
Alarms

WombatOAM shows alarms that occur on managed nodes on the web dashboard's Alarms tab. A message appears for each alarm, while up to three messages at a time also appear as popup windows, showing the latest alarms. All messages increment the counter badge on the Alarms tab.

In addition to the built-in alarms (described below), WombatOAM also collects alarms entries from the alarm_handler module and the elarm application, if they are present on the managed node.

Tags are assigned to the built-in alarms enabling role based information retrieval. Other kind of alarms, such as SASL and elarm alarms, are tagged with the dev and op tags, which can be overridden by adding the following line to wombat.config.

\[
\{\text{set, wo_alarms, default_tags, ["Tag1"", "Tag2"]}\}.
\]

Managing alarm messages

You can acknowledge, unacknowledge, manually clear, and comment on alarms. Most alarms will clear themselves, but can also manually clear those that do not.

The controls in the "Manage listed alarms" box lets you select multiple alarm messages according to their state, severity, or the node on which they occurred, and perform the available actions.

To view and respond to an individual message, click the arrow button on its right. This will show details of the alarm. For new messages, you can use the buttons under the details to acknowledge or clear the message, or to post a comment.

Built-in alarms

Node down

- Alarm id: node_down.
- Severity: major.
- Tags: dev, op

Occurs when a node is not reachable by WombatOAM. This can happen if the node is not running (for example, if it crashed or was shut down), but also in case of network partitions. In the latter case, you may still be able to access the node from other nodes, but not via WombatOAM. You can define the minimum time that needs to elapse between the node becoming unreachable and the alarm being raised. This may be done to avoid raising alarms with minor, intermittent errors. During this time period WombatOAM will keep trying to reconnect to the node. If the reconnection succeeds within this period, no alarm will be raised. This period is 30 seconds by default. You can redefine it in milliseconds by adding the following line to wombat.config.

\[
\{\text{set, wo_core, node_monitor_max_passive_time, 30000}\}.
\]

ETS table count

- Alarm id: ets_limit_minor, ets_limit_major.
- Severity: major/minor.
- Tags: op
An upper boundary (70% & 85%) of the emulator’s limit has been reached.

This alarm is most likely to be raised when the applications running on the managed node create ETS tables dynamically. For example, Mnesia creates an ETS table for each transaction; if there are many transactions running at the same time, it is possible to reach the ETS limit.

The ETS limit can be configured by setting the environment variable ERL_MAX_ETS_TABLES before starting the Erlang runtime system.

**Port count**
- Alarm ids: port_limit_minor, port_limit_major.
- Severity: major/minor.
- Tags: op

An upper boundary (70% & 85%) of the emulator’s limit has been reached.

The port limit can be configured at startup by using the following command-line flag: +Q

**Process count**
- Alarm ids: process_limit_minor, process_limit_major.
- Severity: major/minor.
- Tags: op

An upper boundary (70% & 85%) of the emulator’s limit has been reached.

The process limit can be configured at startup by using the following command-line flag: +P

A bunch of process related metrics are available under the metric category named Processes by default. For instance, the Processes metric shows the number of processes currently existing at the selected node whilst the Orphan processes metric shows the number of only those processes that do not belong to any group leader.

**Atom count**
- Alarm id: atom_limit_minor, atom_limit_major.
- Severity: major/minor.
- Tags: dev, op

An upper boundary (70% & 85%) of the emulator’s limit has been reached.

This alarm is most likely an indication of unsatisfactory code dynamically turning strings into atoms. Since the atom table is not garbage-collected, the only way to recover from this situation is to restart the node. That can be prevented by preferring the `list_to_existing_atom/1` to the `list_to_atom/1` function.

The atom limit can be configured at startup by using the following command-line flag: +t

**Module count**
- Alarm ids: module_limit_minor, module_limit_major.
- Severity: major/minor.
- Tags: dev, op

An upper boundary (70% & 85%) of the emulator’s limit has been reached.
This alarm is most likely to be raised when new modules that are dynamically created by an on-the-fly code generator are not deleted later. To recover from this situation, identify the modules that are no longer needed, and use `code:soft_purge/1` to erase them from the system. Pro tip: use WombatOAM's Services to get rid of all unnecessary modules.

**Exported function count**
- Alarm ids: `export_limit_minor, export_limit_major`.
- Severity: major/minor.
- Tags: dev, op

An upper boundary (70% & 85%) of the emulator's limit has been reached.

See the description for the very similar module count alarm for information on how to recover from this situation.

**Memory used by beam**
- Alarm ids: `memory_limit_minor, memory_limit_major`.
- Severity: major/minor.
- Tags: dev, op

An upper boundary (70% & 85%) of the system's total memory has been used by the Erlang node.

This alarm is only raised when `os_mon` provides total system memory statistics. To recover, identify the processes that consume the most memory. Perhaps, by reviewing their implementations, the memory consumption can be improved. Pro tip: Use WombatOAM's ETOP like process listing to identify the processes that consume the most memory.

**Open file count**
- Alarm ids: `open_file_limit_minor, open_file_limit_major`.
- Severity: major/minor.
- Tags: op

An upper boundary (70% & 85%) of the OS's limit has been reached.

This alarm does not necessarily mean the Erlang node is misbehaving; any other OS process on the host may also be responsible.

There is a built-in metric called `open_x_ports` that shows the number of open ports belonging to specific types. By default, only ports with type TCP/UDP/SCTP are displayed, nonetheless, the metric can be configured to provide a much more detailed view. See the "builtin_metrics plugin" for more information.

This alarm is only raised on Linux and Solaris.

**Open socket count**
- Alarm ids: `open_socket_limit_minor, open_socket_limit_major`.
- Severity: major/minor.
- Tags: op

An upper boundary (70% & 85%) of the OS limit has been reached by all the processes on the current host.

There is a built-in metric category called `Ports and sockets` aiding the investigation of the problem in more details. It includes metrics such as `The Open TCP sockets, Open UDP sockets, and Open SCTP sockets`. 

This alarm is only raised on Linux, Solaris and OS X.

On Linux, the plugin attempts to call the `ss` utility (`ss -s`). If it fails, it will attempt to read `/proc/net/sockstat6` directly. If that fails too, a call to `netstat -an` will be used. A value from `sysctl -n fs.file-max` is used as system limit. If `netstat` fails, 0 will be returned.

On OS X, a call to `sysctl -n` will be used, summing up values for `net.inet.tcp.cubic_sockets`, `net.inet.tcp.background_sockets` and `net.inet.tcp.newreno_sockets`. A value from `sysctl -n kern.maxfiles` is used as system limit. If `sysctl` call fails, attempt to call `netstat -an` will be performed, else 0 will be returned.

On Solaris, a call to `netstat -an` is used. A value from `sysctl -n fs.file-max` is used as system limit. If `netstat` fails, 0 will be returned.

**Open file descriptors limits**

- **Alarm id:** `open_fd_limit_user_minor`, `open_fd_limit_user_major` and `open_fd_limit_node_minor`, `open_fd_limit_node_major`.
- **Severity:** major/minor.
- **Tags:** op

An upper boundary (70% & 85%) of user/node open file descriptors limit has been reached by the Erlang process.

These alarms are raised on Linux, OS X and Solaris.

File descriptor alarms include both sockets and files, as they are sharing same (file descriptor or Erlang ports) resource.

User alarm counts fd's opened by current user against ulimit -n. Node alarm counts ports open in current Erlang node against node limits. Ports are used for both files and sockets in Erlang.

**Disk capacity**

- **Alarm ids:** `disk_capacity_minor`, `disk_capacity_major`.
- **Severity:** major/minor.
- **Tags:** op

An upper boundary (70% & 85%) of the disk capacity has been reached on any of the disks on the host.

The disk capacity on x is a built-in metric, which may be useful to manage the issue. It shows the percentage of disk space occupied on the mounted partition named x.

This alarm does not necessarily mean the Erlang node is misbehaving, any other OS process on the host may also be responsible.

This alarm is only raised when `os_mon` provides disk usage statistics.

**Max message queue length**

- **Alarm ids:** `process_message_queue_minor`, `process_message_queue_major`.
- **Severity:** major/minor.
- **Tags:** dev, op

An upper boundary (5,000 & 20,000 messages) of the message queue length of a single Erlang process has been reached. Long message queues are often a symptom of a bottleneck and result in the degradation of throughput, with the system eventually running out of memory.

If it occurs frequently, you should improve your code to remove the bottleneck. Pro tip: Use WombatOAM's ETOP like process listing to
identify the processes with the largest message queue.

**CPU load**
- Alarm ids: os_cpu_load_minor, os_cpu_load_major.
- Severity: major/minor.
- Tags: op

An upper boundary (70% & 85%) of the CPU utilisation has been reached.

This alarm does not necessarily mean the Erlang node is misbehaving, any other OS process on the host may also be responsible.

This alarm is only raised when os_mon provides CPU usage statistics.

**Shell history size**
- Alarm ids: shell_history_size_minor, shell_history_size_major.
- Severity: major/minor.
- Tags: op

An upper boundary (20 MB & 100 MB) of the cumulative shell history size of all shell processes has been reached. Large shell sizes have been known to cause the node to run out of memory and crash.

To recover from this situation, use one of the following options:
- Terminate unnecessary shell processes. To do this, press Ctrl+G and then use the k command.
- Reduce the shell history size to the N most recent commands. To do this, execute history(N) in the shells you want to keep (for example, the {shell,start,[init]} job).
- Remove unnecessary variable bindings. Use the f(X) and f() shell commands.

**Invalid application version**
- Alarm id: invalid_application_version.
- Severity: major.
- Tags: dev, op

The indicated applications have an older application version than required by the OTP release.

This alarm is only likely to be raised on a node running a custom release composed of OTP applications from different OTP releases. You can find more information under Topology.

**Invalid app file**
- Alarm id: invalid_app_file.
- Additional info: Name of the applications with missing application files.
- Severity: major.
- Tags: dev, op

The indicated applications have an invalid .app file.

While this alarm may be raised frequently, it is sometimes safe to ignore. A typical problem is when an application's version number does not follow the common dot-separated integer format (for example, 0.5.3), but contains a commit ID or similar piece of information (such as 0.5.3-163-gfcbad3). Such an application can be perfectly loaded and used, but will fail to pass this sanity check and accordingly raise an alarm.
Missing runtime dependencies
- Alarm id: missing_runtime_dependencies.
- Additional info: Names of the missing dependencies.
- Severity: major.
- Tags: op

The indicated applications require runtime dependencies (applications that are loaded but not necessarily started) that are not available.

This alarm should be raised on systems that are running an incorrectly built OTP release. However, declaring run time dependencies was introduced in OTP 17 and the documentation states that these specifications are not yet entirely correct, so false positives reported by this sanity check are possible.

Old code
- Alarm id: old_code.
- Severity: minor.
- Tags: op

The indicated modules has old code loaded on the node.

This alarm indicates that a process or a user (via a shell) loaded a new version of a module on the node. This is normal during a hot code upgrade or on a development node that is configured to recompile modules when sources are changed, but in a production environment it is likely to indicate a problem in configuration management and change control.

To recover from this condition, erase the old version of the module's code by calling the following: `code:purge/1`

Module clash
- Alarm id: module_clash.
- Severity: major.
- Tags: dev

The indicated modules have multiple versions in separate directories. This means a name clash in module names.

Properly built OTP releases shouldn't have code clashes, because the release is checked for such errors at build time. This alarm is likely to be raised on systems where the code path is manipulated at runtime.

Different module versions
- Alarm id: different_module_versions.
- Additional info: Info about the name of the module and the module versions used by the listed nodes belonging to the same node family.
- Severity: warning.
- Tags: op

Different nodes in the same node family have different versions of the indicated module loaded.

This alarm indicates a configuration management issue in the cluster. To recover, ensure all nodes have the same module version in a node family.

Missing modules
- Alarm id: missing_modules.
  - Additional info: List the names of the modules which are not loaded into some nodes belonging to the same node family.
  - For configuration possibilities refer to Configuration section.
  - Severity: warning.
  - Tags: op

The modules listed in additional info are not loaded in all nodes belonging to the node family.

This alarm indicates a configuration management issue in the cluster. To recover, ensure the modules are loaded in all nodes in a node family.

**Different application versions**
- Alarm id: different_application_versions.
  - Additional info: Info representing the name of the application and the application versions running on the listed nodes of that node family.
  - For configuration possibilities refer to Configuration section.
  - Severity: warning.
  - Tags: op

Different versions of the indicated application are currently running on nodes in the same node family.

This alarm indicates a configuration management issue in the cluster. To recover, ensure all nodes have the same application version in a node family.

**Missing application**
- Alarm id: missing_application.
  - Additional info: Info about the name of the application which is currently not running on some nodes belonging to the same node family.
  - For configuration possibilities refer to Configuration section.
  - Severity: warning.
  - Tags: op

The indicated application is currently not running on all nodes in the node family.

This alarm indicates a configuration management issue in the cluster. To recover, ensure the same applications are running on all nodes in a node family.

**Different timezones**
- Alarm id: different_timezones.
  - Additional info: Info about the name of the node family, and the timezones with the node names.
  - For configuration possibilities refer to Configuration section.
  - Severity: warning.
  - Tags: op

Nodes in the same node family have different timezone settings.

To recover from this situation, synchronize the timezone settings in the OS of the hosts in the cluster.

**Missing global config**
- Alarm id: global_config_missing.
  - Additional info: Info about the missing global config.
The indicated global config is currently not set on all nodes in the node family. Perhaps, the process of changing global configurations was aborted or a node with old configuration has joined the cluster.

This alarm indicates a configuration management issue in the cluster. To recover, ensure that all the global configs share the same values on all nodes in a node family. Pro tip: use WombatOAM's Configuration management feature.

**Different global configs**
- Alarm id: inconsistent_global_config.
- Additional info: Info about the inconsistent global config, the name of the node family, and the different values with the node names.
- Severity: major.
- Tags: op

The indicated global config is inconsistent. It has different values set on nodes in the node family. Perhaps, the process of changing global configurations didn't succeed on all nodes, or the value was changed locally on some nodes.

This alarm indicates a configuration management issue in the cluster. To recover, ensure that all the global configs share the same values on all nodes in a node family. Pro tip: use WombatOAM's Configuration management feature.

**Disk almost full**
- Alarm id: {disk almost full, Mount}.
- Severity: minor.
- Tags: op

A lower boundary of free space on the indicated disk has been reached.

This alarm is only raised when os_mon provides disk usage statistics. The threshold also depends on the os_mon settings.

**System memory high watermark**
- Alarm id: system_memory_high_watermark.
- Severity: major.
- Tags: op

An upper boundary of total memory usage of all OS processes has been reached.

This alarm is only raised when os_mon provides memory usage statistics. The threshold also depends on the os_mon settings.

**Process memory high watermark**
- Alarm id: process_memory_high_watermark.
- Additional info: Pid of the process.
- Severity: major.
- Tags: dev, op

An upper boundary of memory usage of the indicated Erlang process has been reached.

This alarm is only raised when os_mon provides memory usage statistics. The threshold also depends on the os_mon settings.
**Riak Node down**
- Alarm id: \{riak_core_nodedown, Node\}.
- Severity: major.
- Tags: op

Occurs when a Riak node is not reachable by WombatOAM. This can happen if the node is not running (for example, if it crashed or was shut down), but also in case of network partitions. In the latter case, you may still be able to access the node from other nodes, but not via WombatOAM.

It is recommended to take a look at Riak and other log files, find and fix the cause.

**Mnesia inconsistent**
- Alarm id: mnesia_inconsistent.
- Severity: major.
- Tags: dev, op

This alarm means the database across the cluster is inconsistent. Perhaps, there was a power failure, a temporary network issue or a table was force loaded to some nodes.

To recover, it needs to be resolved manually. The most basic solution is to restore the database from a backup.

**Mnesia overload**
- Alarm id: mnesia_overload.
- Severity: major.
- Tags: dev, op

Mnesia is currently overloaded, this can be caused by frequent or voluminous updates. This is usually a transient condition and the alarm is cleared automatically by WombatOAM.

If it occurs frequently, you should review the design of your code to decrease the use (notably write operations) of Mnesia.

**Poolboy**
- Alarm Id: \{poolboy, POOL\}
- Severity: major.
- Tags: dev, op

An alarm is raised when a pool becomes full: all workers are used and no more workers can be launched. This can happen when the system is under heavy load that is much larger than expected or the workers of the pool are handled incorrectly. Pool workers may need more time to process or too few workers launched.

If the alarm is often raised you may refine the pools' configurations, for instance, consider to increase the pool sizes; or balance the load to other nodes if system resources are scarce.

**SASL alarms**
- Alarm id: As specified when raising an alarm using \texttt{alarm\_handler}.
- Severity: indeterminate.
- Tags: dev, op

Any alarm reported to the SASL \texttt{alarm\_handler} will be reported by WombatOAM.
- Alarm id: As specified when calling elarm:raise.
- Severity: indeterminate.
- Tags: dev, op

Any alarm reported to the Elarm Alarm Manager will be reported by WombatOAM.

**Threshold based alarms**

In the majority of cases good system behaviour can be described with good metric values within a range. WombatOAM can raise alarms based on a given metric crossing a given threshold indicating the need for human intervention.

Notes:

- All threshold based alarms from a node are cleared when the node goes down for some reason.
- Counters and gauges are supported (spirals, meters and histograms are not).

**Configuring threshold based alarms**

Threshold alarms can be configured directly from the GUI.

Select the **Metrics** tab then **Metric Thresholds** tab.

Thresholds can be configured in wombat.config as well with the **threshold_sets** and **threshold_templates** entries within the wo_metrics application. A Threshold Set specifies a list of rules (basically each rule is a threshold for a metric) applicable for a list of nodes and node families. They can have the following fields:

- **nodes** (default []): list of node display names as strings as seen on the Dashboard for which the given rules apply.
- **node_families** (default []): list of node family display names as strings as seen on the Dashboard for which the given rules apply.
- **rules** (mandatory): list of Threshold Rules, each of which can have the following fields:
  - **name**: alarm id to be raised.
  - **metric** ("Category", "Metric"): the metric category and name as seen on the Dashboard on the left side bar. Example categories: "Memory", "Exometer metrics". (Note that metrics belonging to Exometer and Folsom categories have an extra "exometer" or "folsom" prefix in their names when published on the REST API and in other OAM tools such as Graphite but here the shorter Dashboard format must be
used.)

- **direction** (warn above or warn below): should the alarm be raised above or below raise level.
- **raise_level** (number): crossing the raise level in the given direction will trigger the alarm.
- **cease_level** (number): crossing the cease level in the opposite direction will clear the alarm. Setting a different value than raise level will prevent triggering a lot of alarms when the metric value is oscillating around raise level. Cease level must not be higher than raise level if the direction is warn above or lower if the direction is warn below.
- **unit** (numeric or percentage): specifies if the levels are absolute values in the unit of the metric or a percentage of the value of percentage base.
- **percentage_base** (number): must only be given if unit is percentage. In that case levels mean a percentage of this absolute value which should be given in the unit of the metric.
- **template** (optional): name of the Threshold Template to take default values from.

A Threshold Rule can reference a Threshold Template which can have the same fields (but all are optional). The fields from the two are merged with the fields in the Threshold Rule overwriting the same field value from the Threshold Template. A Threshold Template cannot reference another Threshold Template. The `threshold_templates` entry should be a list of two-tuples where the first element is the name of the template, the second is a key-value list.

Example:

The following example configuration raises an alarm with `test_processmem_alarm` id when the "Memory/Process memory" metric goes above 400 (which is 80% of the 500 percentage base value) on nodes `test1` and `test2`. When the metric goes below 300 (which is 60% of the base value), then the alarm is cleared. `test_totalmem_alarm` and `prod_processmem_alarm` are similar alarms so we used a template for the general settings of an above-80% alarm. As the available memory is different in the test and production environment we set different, appropriate values as percentage base for the two groups of nodes. A `msg_queue_alarm` is also produced if the sum of the length of the message queues on any node goes above 300.

```erlang
{set, wo_metrics, threshold_templates,
  [[warn_above_80_percent,
    {raise_level, 80},
    {cease_level, 60},
    {unit, percentage},
    {direction, warn_above}]]}.

{set, wo_metrics, threshold_sets,
  [{nodes, ["test1", "test2"]},
   rules,
   [[{name, "test_processmem_alarm"},
     {metric, ["Memory", "Process memory"]},
     {template, warn_above_80_percent},
     {percentage_base, 500}],[
     {name, "test_totalmem_alarm"},
     {metric, ["Memory", "Total memory"]},
     {template, warn_above_80_percent},
     {percentage_base, 700}]]},

{%node_families, ["prod_location1", "prod_location2"]},
  rules,
  [[{name, "prod_processmem_alarm"},
    {metric, ["Memory", "Process memory"]},
    {template, warn_above_80_percent},
```
As another example, let's configure an Exometer metric. Let's assume that the Exometer metric is displayed as 
[<<"localhost">>, session_count] on the Dashboard, and that you can query by calling 
exometer:get_value([<<"localhost">>, session_count]) on the node. The following rule will raise an alarm if the value of the metric is more than 1000:

```erlang
{set, wo_metrics, threshold_sets, 
 [ 
   [{nodes, all}, 
    {rules, 
     [[{name, "too_many_sessions_alarm"}, 
       {metric, {"Exometer metrics", "[<<"localhostafd631d06d902d04ac4d3713e739adcd8c51960quot;>>,session_count]"}}, 
        {raise_level, 1000}, 
        {cease_level, 500}, 
        {unit, numeric}, 
        {direction, warn_above}]]}] 
]
```

Notification based alarms

WombatOAM can be configured to monitor notifications. If a new notification matches the user defined criteria then WombatOAM will raise an alarm for the node that the log entry has been collected from.

Configuring notification based alarms

Criteria for an alarm can be defined as a rule. The list of rules can be configured in wombat.config with the log_alarms entry within the wo_alarms application. Each rule have the following fields:

- **message_pattern**: Each notification's message will match against this regular expression. Note that a dot in the pattern matches all characters, including those indicating newline.
- **match_tags**: You can further filter notifications by tags. It can be the all atom, or a list of tags in binary format. You can read more about it in the Tags section.
- **alarm_properties**: List of properties that the alarm will have.
  - **alarm_id**: Alarm id to be raised in atom format.
  - **severity**: Severity as an atom.
  - **alarm_tags**: List of binary tags.
  - **probable_cause**: Describes the alarm's probable cause in binary format.
  - **proposed_repair_action**: Proposed repair action in binary format.

Example:

The following example configuration contains one rule to raise an alarm with custom_app_alarm id when a notification arrives that is tagged with <<"dev">> and the message contains the custom_app_alarm text:
{set, wo_alarms, log_alarms,
  [{alarm_properties,
    [{alarm_id, custom_app_alarm},
      {severity, error},
      {alarm_tags, ["<<"dev">>"]},
      {probable_cause, ""},
      {probable_repair_action, ""}],
    {match_tags, ["<<"dev">>"]},
    {message_pattern, "custom_app_alarm"}],
    [{alarm_properties,
      [{alarm_id, new_app_alarm},
        {severity, warning},
        {alarm_tags, ["<<"op">>"]}],
      {match_tags, ["<<"dev">>, "<<"op">>"]},
      {message_pattern, "unwanted result"}]
  ]}.

Email configuration

Configuring the email and SMTP settings can be done through the GUI.

Select the Admin menu then Configuration tab. Under Email and SMTP, you can configure the default Email.

![Email configuration GUI](image)

Email alarm notifications

WombatOAM can send notifications of alarms that are raised on the monitored nodes via email. You can be notified of all alarms, or specify the alarms for which you want to receive notifications.

Configuring email alarm notifications

Alarm notification emails are configured in the elarm and elarm_mailer entries in the sys.config file. Overriding these in the wombat.config file is recommended. Edit the fields to match your system, and restart WombatOAM to load the new configuration. The following example configuration uses Gmail as a relay for outbound emails.

```erl
%% Elarm mailer config
{set, elarm_mailer, sender, "wombat@gmail.com"}.
{set, elarm_mailer, recipients, ["alarms@example.com"]}.
{set, elarm_mailer, gen_smtp_options,
  [{relay, "smtp.gmail.com"},
   {port, 465},
   {ssl, true},
   {username, "wombat"},
   {password, "password"}]}.
```

The following fields need to be edited:

- **elarm_mailer**
  - **sender**: The email address from which alarm notifications will be sent.
  - **recipients**: A list of the email addresses to which the alarm notifications should be sent.

- **gen_smtp_options**
  - **relay**: The server via which emails should be sent. For example, "smtp.gmail.com", or "localhost" if you have a local SMTP server.
  - **username**: The username of the account used for sending emails.
  - **password**: The password of the account used for sending emails.
  - **port**: The SMTP port on the machine from which emails will be sent.

- **subscribed alarms**: The alarms IDs of the alarms for which emails should be sent. For example: `system_memory_high_watermark, disk_capacity_minor, process_message_queue_major, node_down`. WombatOAM will *not* send email notifications for alarms that are not in this list.
Notifications

WombatOAM shows notifications that occur on managed nodes on the web dashboard's Notifications tab. A message appears for each notification, and all messages increment the counter badge on the Notifications tab.

In addition to the built-in notifications (listed below), WombatOAM also collects log entries from the `error_logger` module and the `lager` application if they are present on the managed node.

Tags are assigned to notifications enabling role based information retrieval. All notifications are tagged with the `dev` and `op` tags by default, which can be overridden by adding the following line to `wombat.config`.

```erlang
{set, wo_logs, default_tags, [<<"Tag1">>, <<"Tag2">>>].
```

Viewing notifications

By default, only logs at warning level and above are collected. You can change this on the web dashboard on a per-node basis. On the "Notifications" tab, select the node, and then use the options under "Change log level on selected node".

To view details for an entry, click the arrow button on the right. To find specific log entries, you can use the "Application filter" box and "Search in logs" box above the list of notifications.

If new logs arrive after you have opened the page, WombatOAM will show a message above the list of notifications, with a link to show the new entries. To view entries older than the those that are visible, use the buttons below the list.

Mailing notifications

WombatOAM can be configured to email some or all collected notifications to any number of recipients. At the moment it is only possible to filter notifications by specifying the minimum severity they should have to be emailed.

There are plans to also include other filters such as the actual list of severities, a regular expression the notification message must contain and the possibility to send abridged emails that contain the last `n` notifications.

The details on how to configure this mailer feature can be found in the "Configuration" section.

Built-in notifications

Busy port

Triggered by a system monitor on busy ports. A process in the system was suspended because it was sending to a busy port. This notification is disabled by default, see the documentation of the `code_tracer` plugin on how to enable it.

Busy dist port

Triggered by a system monitor on busy dist ports. A process in the system was suspended because it was sending to a process on a remote node whose inter-node communication was handled by a busy port. This notification is disabled by default, see the documentation of the `code_tracer` plugin on how to enable it.
Long GC
Triggered by a system monitor by an unusually long garbage collection. A garbage collection in the system took longer than expected. This notification is disabled by default, see the documentation of the code_tracer plugin on how to enable it.

Long schedule
Triggered by a system monitor as a result of a process hogging the scheduler. A process or port in the system has been running uninterrupted for a longer time than expected. This notification is disabled by default, see the documentation of the code_tracer plugin on how to enable it.

Large heap
Triggered by a system monitor due to an unusually large heap. A garbage collection in the system resulted in the size of a heap being unusually large. This notification is disabled by default, see the documentation of the code_tracer plugin on how to enable it.

Crash dump saved
A crash dump was saved for inspection. Erlang produces a crash dump file if the runtime terminates abnormally. This can occur because of depletion of memory, or if a supervisor cannot recover a given supervision tree.

Code loaded
A module was loaded in the system.

Code purged
Modules were purged from the system.

Shell command
A shell is evaluating an expression.

SASL error report
An error report was logged.

SASL warning
A warning was logged.

SASL crash report
A crash report was logged.

SASL info report
An info report was logged.

Scheduler sleeps
Indicates a change in the number of online schedulers.

Scheduler wakes up
Indicates a change in the number of online schedulers.
Configuring logging

In the configuration files WombatOAM users can set the behaviour of the following two types of log files where collected logs from the managed nodes will be stored.

- **wombat log files**: With default options all logs excluding the evaluated shell commands will be stored in these files. In the config the size of one file, and the size of all log files can be set with the options shown below. (In the example, we set the size of one log file to 500 Kilobytes, and the size of all log files to 10 Gigabytes.)

```erlang
% size of one log file in bytes
{set, wo_logs, wo_logs_disklog, log_file_size, 512000}.

% size of all logs in bytes
{set, wo_logs, wo_logs_disklog, log_max_size, 10737418240}.
```

- **backup log files**: Using filters we can define what type of log items will be written to the backup log files. These files will have the same size as the log files (log_file_size option will be used), but the size of all backup logs is not limited by the log_max_size config option. However, with the current implementation the number of all backup logs are limited to 50,000.

By default all log entries about the evaluated shell commands will be stored in the backup logs, and everything else in the logs.

The log file size and the size of all log files are defined in the config files. The default options are:

- Log file size: 20 MB
- Size of all logs: 2 GB

Example configuration:

```erlang
% size of one log file in bytes
{set, wo_logs, wo_logs_disklog, log_file_size, 500000}.

% size of all logs in bytes
{set, wo_logs, wo_logs_disklog, log_max_size, 10000000000}.

% A list of filter expressions used to configure log entries which will never be deleted
% Use one of the following expressions for filtering:
% {originator, Originator :: binary()}
% {node_id, NodeId :: binary()}
% {node_family_id, NodeFamilyId :: binary()}
% {message, Message :: binary()}
{set, wo_logs, wo_logs_disklog, backup_filters, []}.
```
Metrics

WombatOAM has more than 100 built-in metrics, organised into metric categories on the dashboard (such categories are Memory, Runtime, I/O, etc.) See the description of all built-in metrics below.

WombatOAM also collects metrics from the folsom and exometer applications if they are running on the managed node. These are shown under the "folsom" and "exometer" metric categories on the dashboard.

Viewing metrics

On the Metrics tab, select a node or node family. This reveals the available metrics, grouped by type and category. Select a type (counter, gauge, meter, spiral, histogram or duration - these are shown as tabs) and a category to reveal individual metrics, and then select a metric to display it.

For individual nodes, each of the numeric metrics are displayed as line charts. If you select a node family, the metrics are displayed as stacked graphs showing all the nodes in the family. For node families only one metric can be selected for stacked display and only counters and gagues are supported (as they only have single datapoints).

You can superimpose metrics to view several metrics on the same graph. Each metric you select is added to the graph. To remove a metric, click it again to clear the checkbox next to it. To clear all metrics from the graph, click the "trash" icon on the right above the graph. You can also remove individual metrics on the "Configure metrics" window.

Other viewing options:

- Refresh interval: To change the frequency at which the graph refreshes, click the "settings" icon on the right above the graph to open the "Configure metrics" window, and then select an option in the "Refresh interval" list.
- Markers: Different metrics are distinguished on the graph by different colours and markers. To hide or show these markers, click the "settings" icon, and select or clear the "Enable markers" checkbox.
- Delta: To view deltas instead of actual values, select the "Delta" (Δ) icon on the right above the graph.

By default, each metric is polled once a minute, i.e. the metric graphs will show a new data point once a minute. If you would like to change this setting, please refer to the Configuration reference in the "Configuration" section.

Built-in metrics

I/O

Input I/O bytes

The total number of bytes received through ports. * Tags: dev, op

Output I/O bytes

The total number of bytes output to ports. * Tags: dev, op

TCP: Total bytes received

The total number of bytes that have been received by TCP. * Tags: dev, op
**TCP: Packets received**
The number of TCP packets that have been received. * Tags: dev

**TCP: Average received packet size**
The average size of TCP packets that have been received. * Tags: dev

**TCP: Maximum received packet size**
The maximum size of TCP packets that have been received. * Tags: dev

**TCP: Total bytes sent**
The total number of bytes that have been sent by TCP. * Tags: dev, op

**TCP: Packets sent**
The number of TCP packets that have been sent. * Tags: dev, op

**TCP: Average sent packet size**
The average size of TCP packets that have been sent. * Tags: dev

**TCP: Maximum sent packet size**
The maximum size of TCP packets that have been sent. * Tags: dev

**UDP: Total bytes received**
The total number of bytes that have been received by UDP. * Tags: dev, op

**UDP: Packets received**
The number of UDP packets that have been received. * Tags: dev

**UDP: Average received packet size**
The average size of UDP packets that have been received. * Tags: dev

**UDP: Maximum received packet size**
The maximum size of UDP packets that have been received. * Tags: dev

**UDP: Total bytes sent**
The total number of bytes that have been sent by UDP. * Tags: dev, op

**UDP: Packets sent**
The number of UDP packets that have been sent. * Tags: dev, op

**UDP: Average sent packet size**
The average size of UDP packets that have been sent. * Tags: dev

**UDP: Maximum sent packet size**
The maximum size of UDP packets that have been sent. * Tags: dev

**Disk usage on x**
The result of the latest disk check for each partition. Reports the disk usage (e.g. the percentage of disk space occupied) on a mounted partition. * Tags: op

**Inode usage on x**
The result of the latest disk check for each local mount point. Reports the inode usage (e.g. the percentage of inodes used) on a mounted partition. * Tags: op

**Memory**

**Total memory**

The total amount of memory currently allocated, which is the same as the sum of memory size for processes and system. * Tags: dev, op

**Process memory**

The total amount of memory currently allocated by the Erlang processes. * Tags: dev, op

**Process memory used**

The total amount of memory currently used by the Erlang processes. This memory is part of the memory presented as process memory. * Tags: dev

**System memory**

The total amount of memory currently allocated by the emulator that is not directly related to any Erlang process. Memory presented as processes is not included in this memory. * Tags: dev, op

**Atom memory**

The total amount of memory currently allocated for atoms. This memory is part of the memory presented as system memory. * Tags: dev

**Atom memory used**

The total amount of memory currently used for atoms. This memory is part of the memory presented as atom memory. * Tags: dev

**Binary memory**

The total amount of memory currently allocated for binaries. This memory is part of the memory presented as system memory. * Tags: dev

**Code memory**

The total amount of memory currently allocated for Erlang code. This memory is part of the memory presented as system memory. * Tags: dev

**ETS memory**

The total amount of memory currently allocated for ETS tables. This memory is part of the memory presented as system memory. * Tags: dev

**System total memory**

The amount of memory available to the whole operating system. * Tags: dev, op

**Total memory available**

The total amount of memory available to the Erlang emulator, allocated and free. May or may not be equal to the amount of memory configured in the system. * Tags: dev, op
**Buffered memory**
The amount of memory the system uses for temporary storing raw disk blocks. * Tags: dev

**Cached memory**
The amount of memory the system uses for cached files read from disk. * Tags: dev

**Free memory**
The amount of free memory available to the Erlang emulator for allocation. * Tags: dev, op

**Free swap memory**
The amount of memory the system has available for disk swap. * Tags: dev

**Swap memory used**
The amount of memory the system is using for disk swap. * Tags: dev, op

**Atoms**
The total number of atoms in the system. * Tags: dev

**DETS tables**
The number of open DETS tables on the selected node. * Tags: dev, op

**ETS tables**
The number of ETS tables at the selected node. * Tags: dev, op

**Low memory**
The total amount of memory allocated in low memory areas that are restricted to less than 4GB even though the system may have more physical memory. The metric is available only on 64-bit halfword emulator. * Tags: dev

**Maximum memory**
The maximum total amount of memory allocated since the emulator was started. The metric is available only when the emulator is run with instrumentation. * Tags: dev

**Allocated atom_table area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Allocated bif_timer area**
Memory allocated for timers in bytes. * Tags: dev

**Allocated bits_bufs_size area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Allocated dist_table area**
Memory allocated for the distribution table in bytes. * Tags: dev

**Allocated ets_misc area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Allocated export_list area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Allocated export_table area**
Memory allocated for the export table in bytes. * Tags: dev

**Allocated fun_table area**
Memory allocated for the function table in bytes. * Tags: dev

**Allocated link_lh area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Allocated loaded_code area**
Memory allocated for all the loaded code in bytes. * Tags: dev

**Allocated module.refs area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Allocated module_table area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Allocated node_table area**
Memory allocated for the table of nodes in bytes. * Tags: dev

**Allocated process_table area**
Memory allocated for the process table in bytes. * Tags: dev

**Allocated register_table area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Allocated static area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Allocated sys.misc area**
The amount of allocated memory for this area in bytes. * Tags: dev

**Mnesia System metrics**
These metrics are collected by the mnesia plugin. For more information, see the "mnesia plugin" section.

**checkpoints**
The checkpoints currently active on the node.

**db_nodes**
The nodes which make up the persistent database.

**dc_dump_limit**
Controls how often disc_copies tables are dumped from memory. Lower values reduce CPU overhead but increases disk space and startup times.
**dump_log_time_threshold**
The time threshold for transaction log dumps in milliseconds.

**dump_log_write_threshold**
The write threshold for transaction log dumps as the number of writes to the transaction log.

**extra_db_nodes**
Extra db_nodes to be contacted at start-up.

**held_locks**
Locks held by the local Mnesia lock manager.

**local_tables**
Tables which are configured to reside locally.

**lock_queue**
Transactions that are queued for execution by the local lock manager.

**master_node_tables**
Tables with at least one master node

**no_table_loaders**
The number of parallel table loaders during start. More loaders can be good if the network latency is high or if many tables contains few records.

**running_db_nodes**
Nodes where Mnesia currently is running. For more information, see mnesia:system_info/1

**subscribers**
Local processes currently subscribing to system events.

**tables**
Locally known tables.

**transaction_commits**
A number that indicates how many transactions have terminated successfully since Mnesia was started.

**transaction_failures**
A number that indicates how many transactions have failed since Mnesia was started.

**transaction_log_writes**
A number that indicates the number of write operations that have been performed to the transaction log since start-up.

**transaction_restarts**
A number which indicates how many transactions have been restarted since Mnesia was started.
transactions
All currently active local transactions.

**Nodes, modules and applications**

**Known nodes**
The number of nodes that are known to the selected node; this includes not only visible nodes, but also hidden nodes and previously known nodes, etc. * Tags: dev, op

**Connected nodes**
The number of nodes that are connected to the selected node. * Tags: dev, op

**Visible nodes**
The number of nodes that are connected to the selected node through normal connections. * Tags: dev

**Hidden nodes**
The number of nodes that are connected to the selected node through hidden connections. * Tags: dev

**Traced nodes**
The number of nodes traced from the current node by the Erlang dbg facility. * Tags: dev

**Loaded modules**
The number of loaded Erlang modules (current and/or old code), including preloaded modules. * Tags: dev

**Old modules**
The number of modules that have old code. For more details, see "Current and Old Code" on the following page: www.erlang.org/doc/man/code.html * Tags: dev

**Module name clashes**
The number of module name clashes. The function searches the entire code space for module names with identical names. * Tags: dev, op

**Loaded applications**
The number of applications that have been loaded into the application controller. This includes any included applications. * Tags: dev

**Started applications**
The number of processes started by the application controller process, which starts all other applications. * Tags: dev

**Running applications**
The number of applications that are currently running. * Tags: dev

**Ports and sockets**

**Open ports**
The number of ports currently existing on the local node. * Tags: dev, op

Ports with driver level locking

Number of ports with driver level locking. Driver level locking implies that all instances (ports) of the same port driver will use a global lock and only one emulator thread will execute code in the driver at a time. (As opposed to port level locking where each instance of the same port driver will use a per-instance lock and multiple emulator threads may execute code in the driver at the same time.) It might indicate a bottleneck if such a driver has many instances. (See erlang:port_info/2 and the ERL_DRV_FLAG_USE_PORT_LOCKING driver flag.) This metric is always zero on VMs with SMP support disabled. * Tags: dev

Alive ports total input in bytes

The total amount of data, in bytes, queued by all ports using the ERTS driver queue implementation. * Tags: dev

Alive ports total output in bytes

The total number of bytes written to by all ports from Erlang processes using either port_command/2, port_command/3, or Port ! {Owner, {command, Data}}. * Tags: dev

Open TCP sockets

The number of TCP sockets that are connected. * Tags: dev, op

Open UDP sockets

The number of UDP sockets that are connected. * Tags: dev, op

Open SCTP sockets

The number of SCTP sockets that are connected. * Tags: dev, op

Open x ports

The number of open ports belonging to a specific type. This type is obtained from erlang:port_info/1 using the name key of the proplist. * Tags: dev, op

By default, only ports with type TCP/UDP/SCTP are displayed. The port_type_counters_mode option can be used to configure WombatOAM to show all ports, including ports opened by a running application. See the "builtin_metrics plugin" for more information.

Process notifications

Long GC

The number of "Long GC" triggers from the system monitor in the last period (minute or second). A "Long GC" trigger means that a garbage collection in the system took longer than expected. * Tags: dev

Long schedule

The number of "Long schedule" triggers from the system monitor in the last period (minute or second). A "Long schedule" trigger means that a process or port in the system has been running uninterrupted for a longer time than expected. * Tags: dev

Large heap

The number of "Large heap" triggers from the system monitor in the last period (minute or second). A "Large heap" trigger means that a garbage collection in the system resulted in the size of a heap being unusually
**Busy port**

The number of "Busy port" triggers from the system monitor in the last period (minute or second). A "Busy port" trigger means that a process in the system was suspended because it was sending to a busy port. * Tags: dev, op

**Busy dist port**

The number of "Busy dist port" triggers from the system monitor in the last period (minute or second). A "Busy dist port" trigger means that a process in the system was suspended because it was sending to a process on a remote node whose inter-node communication was handled by a busy port. * Tags: dev, op

**Processes**

**Processes**

The number of processes currently existing at the selected node. * Tags: dev

**Process limit**

The maximum number of processes that can existing simultaneously on the selected node. * Tags: dev, op

**Registered processes**

The number of process names that have been registered. * Tags: dev

**OS threads**

Returns the result of calling the function `cpu_sup:nprocs/0`. Note that this function returns the number of LWP's (aka threads) that are alive in the system. That is something similar to what you get when you run `ps -eLF`. It is a rudimentary way of measuring the system load that may be of interest in some cases. * Tags: dev, op

**Processes traced**

The total number of processes traced by the Erlang's tracing mechanism. * Tags: dev

**Sum process dictionary size**

The total size of the process dictionaries of all processes. * Tags: dev

**Sum message queue length**

The total number of messages currently in the message queue of all processes. * Tags: dev, op

**Error logger message queue length**

The number of messages currently in the message queue of the Erlang error logger. * Tags: dev, op

**Memory size of all processes**

The total size of all processes. This includes call stack, heap and internal structures. * Tags: dev

**Number of process groups**
The total number of known process groups. * Tags: dev

**Shell history length**
Sum of history length (the number of commands evaluated by a shell process) of all shell processes. * Tags: dev, op

**Shell process size**
Sum of the size of all shell processes in bytes. This includes call stack, heap, and internal structures. * Tags: dev

**Processes in running state**
The number of processes where the status of the process is running. * Tags: dev

**Processes in runnable state**
The number of processes where the status of the process is runnable (ready to run, but another process is running). * Tags: dev

**Processes in exiting state**
The number of processes where the status of the process is exiting. * Tags: dev

**Processes in GC state**
The number of processes where the status of the process is garbage_collecting. * Tags: dev

**Processes in waiting state**
The number of processes where the status of the process is waiting (for a message). * Tags: dev

**Processes in suspended state**
The number of processes where the status of the process is suspended (suspended on a "busy" port or by the erlang:suspend_process/[1,2] built-in function). * Tags: dev

**Processes with max priority**
The number of processes where the current priority level for the process is max. * Tags: dev

**Processes with high priority**
The number of processes where the current priority level for the process is high. * Tags: dev

**Processes with normal priority**
The number of processes where the current priority level for the process is normal. * Tags: dev

**Processes with low priority**
The number of processes where the current priority level for the process is low. * Tags: dev

**Processes in app x**
Every process is a member of some process group and all groups have a group leader process. Every application has a group leader. For each
application, this metric shows the number of processes associated to
the application group leader. * Tags: dev

**Orphan processes**

The number of processes that do not belong to any group leader. * Tags: dev

**Unknown processes**

The number of processes that belong to a non-application group leader process. * Tags: dev

**Runtime**

**Context switch count**

The total number of context switches since the system started. * Tags: dev

**Scheduler run queue size**

The total length of the run queues, that is, the number of processes that are ready to run on all available run queues. * Tags: dev, op

**Reductions total**

The total number of reductions performed by processes. This is an approximate measure of how much CPU time they have used. * Tags: dev

**Reduction count for alive processes**

The total number of reductions executed by all processes that are still running. * Tags: dev

**Garbage collections**

The total number of garbage collections since the system started. * Tags: dev

**Minor garbage collections**

The total of minor garbage collections that have happened so far for every process in the system. * Tags: dev

**Bytes reclaimed by GC**

The total number of bytes reclaimed through garbage collection. * Tags: dev

**Average fullsweep after**

The average value of the fullsweep_after parameter for all processes. Relates to garbage collection. * Tags: dev

**Average min binary vheap size**

The average of minimum binary virtual heap sizes (in words) for all processes. * Tags: dev

**Average min heap size**

The average of minimum heap sizes (in words) for all processes. * Tags: dev

**CPU utilization total**
The sum of CPU utilization in percentages on all the cores. In case of having 4 CPUs the maximum is 400. * Tags: dev, op

**CPU load for 1 avg**

The average system load in the last minute, as described at cpu_sup. 0 represents no load, 256 represents the load reported as 1.00 by rup. * Tags: dev, op

**CPU load for 5 avg**

The average system load in the last five minutes, as described at cpu_sup. 0 represents no load, 256 represents the load reported as 1.00 by rup. * Tags: dev, op

**CPU load for 15 avg**

The average system load in the last 15 minutes, as described at cpu_sup. 0 represents no load, 256 represents the load reported as 1.00 by rup. * Tags: dev, op

**CPU utilization - kernel on core x**

CPU utilization (the percentage share of the CPU cycles spent in this processor state) for executing code in kernel mode. Each CPU is specified separately, if this information can be retrieved from the operating system. Available for Solaris and Linux only. * Tags: dev, op

**CPU utilization - user on core x**

CPU utilization (the percentage share of the CPU cycles spent in this processor state) for executing code in user mode. Each CPU is specified separately, if this information can be retrieved from the operating system. Available for Solaris and Linux only. * Tags: dev, op

**CPU utilization - nice user on core x**

CPU utilization (the percentage share of the CPU cycles spent in this processor state) for executing code in low priority (nice) user mode. Each CPU is specified separately, if this information can be retrieved from the operating system. Available for Linux only. * Tags: dev, op

**CPU utilization - idle on core x**

The percentage share of the CPU cycles spent in the idle state. Each CPU is specified separately, if this information can be retrieved from the operating system. Available for Solaris and Linux only. * Tags: dev, op

**Time**

**Active timers**

The number of all timers (one-shot and interval timers) in the table holding timing requests and timer objects; this is an ETS table maintained by Erlang's timer server. * Tags: dev

**Interval timers**

The number of internal timers in the timer interval table, an ETS table maintained by Erlang's timer server. * Tags: dev

**CPU utilization total**

The sum of the runtime for all threads in the Erlang run-time system. This may be greater than the wallclock time. The original source of the calculated value is the times kernel call. * Tags: dev
**Wallclock total**

The time that has elapsed since the program started, measured in real time (as if checking with a clock on the wall). *Tags: dev*

**Scheduler x active wall time**

Active time of scheduler x in terms of wall-clock time. *Tags: dev*

**Scheduler x total wall time**

Time elapsed in terms of wall-clock time since activation wall-clock in scheduler x. *Tags: dev*

**Scheduler x utilization**

The average utilization (i.e., the ratio of the elapsed active wall time and total wall time) of the scheduler for the last one minute (one second in case of live metrics) interval. *Tags: dev*
Services

Functionalities under services are a bunch of standalone features, which are not alarms, metrics and notifications, but aiding devops daily tasks. These features turn WombatOAM to be an active agent. That means users can retrieve live information about running systems (such as the state of a gen_server), recover from outages for instance by forcing GC to save more memory, or fix misconfiguration issues by changing configuration parameters.

The services provided fall into 3 categories:

- **configurator**: dealing with configuration management,
- **explorer**: inspecting the subject on a read-only way,
- **executor**: executed for their side-effect.

The scope of each service is either one node or one node family. Each service has zero or more arguments. The started instance of a service is called a request.

The provided services include

- Configuration management.
- Etop like process listing.
- Inspecting processes.
- Evaluating Erlang/Elixir expressions.
- Soft purge modules.
- Garbage Collect Processes.

There are some boolean properties of services worth mentioning:

- **Exclusiveness**: if a service is exclusive only one request of the same service can be run on the same node or node family at the same time. This is because the request either uses some global resource or to prevent multiple instances using too much resources that would have impact on the managed node.
- **Periodicity**: most services are executed only once with their result shown as output or executed for their side-effects. Periodic services, however, keep running until they are stopped regularly updating their output data (for example Processes list). A Running request can be stopped explicitly by the user or it also terminates if it is left unattended (i.e. no browser is subscribed to display its output). This inactivity timeout is one minute.

Configurator

Tailoring an application to fit a certain system is achieved by configuring the application. An application can be configured using its configuration parameters, which are often referred to as environmental variables or simply envs. A configuration parameter for an arbitrary application is a pair consisting of an identifier and a value. As applications can always retrieve the current value of a configuration parameter during runtime, users can change the system's behaviour immediately - without the need of stopping or rebuilding the system.

There are two kinds of configuration parameters, namely, a configuration parameter can be local or global. Local configuration parameters are specific to Erlang nodes, their values can vary among the nodes. The data's root directory of an Erlang node, the HTTP port on which the Erlang node listens are good examples for local configuration parameters. In contrary, global configuration parameters must have the same value set on all the nodes belonging to the same node family as they describe the property of the overall system. As examples, consider the IP address of the load balancer or the lifetime of sessions.
Services provided on node level

Services provided on node level are special WombatOAM plugins running on the nodes. They are super simple. Both their authority and their knowledge about its managed node are limited and restricted. They encapsulate the domain-specific knowledge they have and provide simple, well-defined services.

The Application tab in the Topology page provides the user with information about the loaded application, which includes the application's configuration parameters, the user can select any application and with the help of the provided service, to set the value of the parameter.

- **Set an application's configuration parameters.** It changes the values of the given configs at once, Application should be a loaded Erlang application, all Keys should be atoms and all Values syntactically valid Erlang terms. Users may also want to store the new parameter permanently in the configuration file that the Erlang VM has been booted with, a dropdown menu allows the user to choose whether to persist the parameter or not.

By the above service, users can inspect and manage the current configuration of any application loaded to their managed nodes.

The plugin also detects some possible misconfigurations and raises an alarm for them:

- A configuration file can't be read/parsed or contains multiple configurations for a single application (corrupt_config_file alarm).
- A configuration file has changed - maybe the application should reload it (config_file_changed alarm).
- A configuration parameter only exists in the configuration file, not in the application environment - maybe the configuration file should be reloaded {{config_only_in_file, <app-name>, <param-name>}} alarm)
- A configuration parameter has different values in the configuration file than in the application environment - maybe the configuration file should be reloaded {{config_difference, <app-name>, <param-name>}} alarm, the alarm info contains the difference).

Services provided on node family level

Until that point configuration management was examined from the node's perspective. However, commands including configuration changes make sense on node family level. Performing a command on a node family usually means doing the same thing on all the nodes. Services are related to the global configurations of the nodes belonging to a certain node family. Facts:

- Configuration parameters which are global configurations shall share the same value on all nodes.
- If a certain global config has different values among the nodes belonging to the same node family then something went wrong, it is an anomaly.
- If a certain global config is not set on some nodes of the node family then something went wrong, it is an anomaly.
- Generally, the values of a global configuration should only be changed when the change affects all the nodes to keep the consistency of the system. However, node families having some nodes being down at the moment do not satisfy the previous condition. Despite of that, users may still want to change the global configuration related to such node families. Thus, the service is available with a clear notice on its current scope.
Global configurations are the subset of the local configurations. That means changing a global configuration only on a certain node (i.e. locally) is still allowed. However, if it happens, it is still an anomaly. It is sometimes useful to try the new value only on one node before changing the value globally.

WombatOAM maintains the set of the global configurations’ keys and their values. This set is empty by default. A configuration becomes a global configuration if and only if the user requests it.

WombatOAM periodically scans for anomalies. An alarm will be raised whenever different values are set to any global configuration or whenever any global configuration is missing from any node of the family. These alarms will be cleared when the values of all the global configurations are the same and also set on all the nodes.

WombatOAM coordinates changing the values of global configurations on all the nodes. Requested changes will be carried out only on those nodes that have been listed in the description of the requested service.

Changing the value of a global configuration is a process that is carried out on the nodes sequentially. The process is aborted by any unexpected failure. While it is in progress there is an anomaly. Some of the nodes have already been updated and are using the new value, while the others are still using the old value. The anomaly can be tolerated, but only temporarily, until the end of the process. If the process managed to change the value on all the nodes, the anomaly will disappear, otherwise the anomaly remains.

WombatOAM provides the following services:

- **Set application parameter as global.** Set the given configuration parameter, which is identified by Key and Application, as global configuration. The consistency of the config's values will be checked periodically among the nodes of the node family.

- **Set application parameter as local.** Revert the given configuration parameter, which is identified by Key and Application, to be local again and therefore to be excluded from further inconsistency checks and global configs’ changes.

- **List globals.** List global configs of Application together with their values. The values of the global configs can be read from cache or can be retrieved from nodes.

- **Change global configs’ values.** Change the values of the given global configs on the nodes sequentially. The first unexpected failure aborts the whole process. The new value can be persisted in the configuration file of each node belonging to the family.

*Explorer*

Retrieve either static or periodically updated information about a certain process or about the VM. Has none or only a very small side-effect on the managed node.

The following services are available:

- **Process Manager.** Presents information about Erlang processes similar to the information presented by "top" in UNIX. See details in the Process Manager section.

- **Process info.** Shows the process information of the process given by its pid or registere name.

- **Process messages.** Shows the message queue of the process
given by its pid or registere name.

- **Process stack trace.** Shows the stack trace of the process given by its pid or registere name.

- **Process dictionary.** Shows the dictionary of the process given by its pid or registere name.

- **Process state.** Shows the state (if available) of the process given by its pid or registere name.

- **Table Visualizer.** Shows information about the ETS tables and their contents. See details in the Table Visualizer section.

**Process Manager**

For each process the following information is displayed:

- **Pid.** The process identifier.

- **Reds.** The number of reductions executed by the process.

- **Memory.** The size of the process, in bytes, obtained by a call to `process_info(Pid, memory)`.

- **MsgQ.** The length of the message queue of the process.

Configuration possibilities are:

- **Refresh interval in seconds.** Time interval (in seconds) between each update of the output.

- **Number of listed processes.** Number of rows (processes) to display.

- **Order by column.** Identifies what information to sort by.

- **Order direction.** Identifies which direction to sort by (Ascending/Descending).

Various information can be retrieved by clicking on any **Pid**, such as:

- **Process info.**

- **Process messages.**

- **Process stack trace.**

- **Process dictionary.**

- **Process state.**

**Table Visualizer**

For each ETS table the following information is displayed:

- **Id.** The identifier of the table.

- **Name.** The name of the table.

- **Type.** The table type (set, ordered_set, bag or duplicate_bag).

- **Objects.** The number of objects inserted in the table.

- **Size (words).** The number of words allocated to the table.

- **Owner pid.** The pid of the owner of the table.

- **Owner name.** The registered name of the owner process of the table.

Configuration possibilities are:
- **Refresh interval in seconds.** Time interval (in seconds) between each update of the output.
- **Order by column.** Identifies what information to sort by.
- **Order direction.** Identifies which direction to sort by (Ascending/Descending).

By clicking on the id of the ETS table, the contents of the table can be retrieved.

This service limits the size of the data that it displays. By default:

- Only the first 200 ETS tables are shown.
- Only the first 100 rows are shown in an ETS table.
- Only the first 10 columns are shown in an ETS table.

These values can be modified in `wombat.config`:

```erlang
{set, wo_plugins, plugins, observer_ets, table_count_limit, 200}.
{set, wo_plugins, plugins, observer_ets, row_limit, 100}.
{set, wo_plugins, plugins, observer_ets, col_limit, 10}.
```

### Executor

Real commands (e.g. Garbage Collect Processes, Evaluate Erlang/Elixir expressions). Executed for their side-effects. May not have any meaningful result as output. As these services give much freedom to the WombatOAM users to modify the monitored nodes for safety reasons it might be useful to disable all executors by adding the following line to the `wombat.config`:

```erlang
{set, wo_services, enable_executor, false}.
```

The following services are provided:

- **Evaluate Erlang expression.** It evaluates the user given Erlang expression on the managed node and returns the result.
- **Soft purge modules.** Soft purge all modules from the managed node (see `code:soft_purge/1`).
- **Garbage Collect Processes.** Force garbage collection on all processes in a controlled way. It asks the VM to perform garbage collection on `GcGroupSize` processes sequentially. Then, it sleeps for `GcSleepTime` milliseconds to weaken the negative effect of GC on the managed node.
- **Terminate process.** Sends an exit signal with the given exit Reason to a process. The process can be given by PID or registered name.
- **Terminate shell processes.** Sends an exit signal with the kill reason to all processes currently executing a function from the shell module. This kills all shells and the processes linked to them.
- **Recover from partitioned mnesia.** If mnesia running on the current managed node is partitioned, this service restarts mnesia on all other nodes running this distributed mnesia and configures those nodes to load the table data from the current managed node to achieve consistent data in the tables. **Warning:** if there's different data in the tables in the other nodes (i.e. data written into the tables after mnesia was partitioned), that data is lost. Select carefully the node where you execute this service as the data of that node will be replicated.
The following additional service is provided on Elixir nodes:

- **Evaluate Elixir expression.** It evaluates the user given Elixir expression on the managed node and returns the result.
Role based information retrieval

Role based information retrieval helps users filter information to focus only on data related to their role.

Developers are usually interested in fine-grained information that gives insight into the Erlang VM, reveals performance bottlenecks and a potential overuse of resources. Operators maintaining production Erlang systems tend to use host and node level information. Operation system level metrics and alarms are usually enough to uncover bad trends. For instance, having a process with very large message queue is a useful information for both roles. However, operation should not be bothered with the amount of memory allocated by the active timers, but be warned when an Erlang node is running out of file descriptors.

The filtering is based on tags that classify metrics, notifications and alarms. There are two roles used as built-in tags: dev and op. All metrics, notifications and alarms are tagged with any subset of the two roles (i.e. dev, op, dev and op) to split information between developers and operators. The documentation of metrics, notifications and alarms include the default tags. Additionally to the two roles, object attributes can be used as tags.

You may want to overwrite the default assignments or introduce your own tags. For both reasons, you should use custom tags.

The tags assigned to users select those metrics, notifications and alarms that are matching any of the given tags, hence the tags hide irrelevant data from users. This filtering mechanism works on the REST layer, thus it can be used to customise all REST based integrations even if the authentication is switched off. By defining a set of tags, you can control the non-REST based integrations' behaviour (e.g. narrowing down the set of metrics pushed to Datadog).

Note that the tags assigned to users are used only if authentication is switched on. When authentication is disabled, the tags defined by the default_user_tags configuration parameter are used by the REST layer.

Built-in tags

Roles

The following roles can be used as tags to filter metrics, notifications and alarms: "dev" and "op".

Object attributes

The following object attributes, which should be given as binary strings, can be used as tags.

Defined for metrics, notifications and alarms:

- "metric", "log" and "alarm" respectively.
- NodeId The UUID of the managed node that the object corresponds to (e.g. "7e7e1188-ff5d-45ed-8103-1d32542efa25").
- NodeName The node name of the managed node that the object corresponds to (e.g. "wombat@127.0.0.1").
- DisplayName The display name of the managed node that the object corresponds to (e.g. "my-wombat-node").
- NodeFamilyId The id of the node family including the managed node that the object corresponds to (e.g. "056e1058-f646-494a-8fe5-2cc05d7bd1e").
- NodeFamilyName The name of the node family including the managed node that the object corresponds to (e.g. "wombat-family").
• **Originator** The short name of the plugin responsible for providing the data (e.g. `"poolboy"` stands for `wombat_plugin_poolboy`).

Object attributes that can only be used to filter alarms are

- **Severity** The severity of the alarm (e.g. `"minor"`).
- **AlarmKey** The built-in alarms are either parametric alarms or global alarms. Parametric alarms (e.g. `{missing_application, App}`) are identified as tuples. In this case the AlarmKey is the first element of the tuple (e.g. `"missing_application"`). The identifiers of global alarms are atoms (e.g. `ets_limit`). In this case, the AlarmKey is the binary string representation of the atom (e.g. `"ets_limit"`).

Object attributes that can only be used to filter metrics are

- **MetricType** The type of the metric (e.g. `"counter"`).
- **MetricGroup** The group of the metric (e.g. `"I/O"`).
- **MetricName** The name of the metric (e.g. `"Output I/O bytes"`).

Object attributes that can only be used to filter notifications are

- **Severity** The severity of the log entry (e.g. `"info"`).
- **PropertyValues** Values stored in the log object's property list.

**Custom tags**

To introduce a custom tag, define the name of the tag and a logical condition that must be satisfied by any alarm or metric to be tagged with your custom tag.

The logical condition should be provided using any valid combinations of filter expressions. A filter expression can be constructed using the `all` and `any` operators (logical conjunction, disjunction and negation operators, respectively) combined with the object attributes or roles.

Before going into details, let's see two examples.

Assuming we only want to create new PagerDuty incidents for the major alarms that are related to the two production nodes, which are named as `node1` and `node2`, we define and then use the custom tag, which is named as `"MajorAlarmsFromProdNodes"`, to configure PagerDuty.

```erlang
{"MajorAlarmsFromProdNodes", [all, ["major",
"alarm",
{any, ["node1",
"node2"]}]]}.
```

Assuming we are almost satisfied with the built-in `dev` role, we just want to remove anything related to node `N1`, hide the `invalid_application_version` alarm and show `ets_limit` and `os_cpu_load` alarms. In this case we may customise the `dev` role as follows.

```erlang
{"dev", [all, ["dev",
{not, "invalid_application_version"},
{not, "N1"}],
"ets_limit",
"os_cpu_load"]}.
```

**Constructing filter expressions**
Conﬁguration

86

A valid ﬁlter expression can be typed as followed.
1
2
3
4
5
6

-type
.
-type
-type
-type
-type
-type

filter_expr() :: any_expr() | all_expr() | not_expr()
any_expr() :: {any, non_empty_list(expr())}.
all_expr() :: {all, non_empty_list(expr())}.
not_expr() :: {'not', expr()}.
expr() :: filter_expr() | object_attr().
object_attr() :: binary().

Deﬁning custom tags
All custom tags should be deﬁned as one list and loaded into
WombatOAM by adding the following line to wombat.config.
1
2

{set, wo_core, custom_tags, [{<<"MyTag1">>, {any, [<<"T1">>
, <<"T2">>]}},
{<<"MyTag2">>, {'not', <<"T3">
>}}]}.

Another example.
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

{set, wo_core, custom_tags,
[{<<"datadog_tag">>,
{any,
[
{all, [
{any, [<<"node1@127.0.0.1">>]},
{any, [<<"Process memory">>]}
]},
{all, [
{any, [<<"node2@127.0.0.1">>]},
{any, [<<"System memory">>, <<"Total memory">>]}
]}
]
}
}]}.



Configuration

wombat.config and sys.config

The main WombatOAM configuration file is rel/wombat/files/wombat.config. The following example shows its file format:

```erlang
%% Disable authentication for the REST interface.
{set, wo_rest, enable_authentication, false}.

%% Modify the shell_history_size thresholds of the alarm plugin.
{set, wo_plugins, plugins, alarm, system_checks, shell_history_size, [{minor, 10000}, {major, 20000}]}.

%% Disable the code_tracer plugin.
{unset, wo_plugins, plugins, code_tracer}.
```

See the "Configuration file format" section below for more information.

**Note:** After each configuration tuple in wombat.config, a period (.) is needed.

WombatOAM needs to be restarted for the changes in wombat.config to take an effect, but no other action (such as regenerating the release) is needed.

The configuration examples in the documentation always apply to wombat.config (unless specified otherwise).

The default values of the configuration parameters can be found in the rel/wombat/wombat/releases/<version>/sys.config file. Modifying this file is not recommended, since it is generated from rel/wombat/files/sys.config and rel/wombat/files/vars during release generation and thus easily overwritten.

Please note that configuring Wombat with the user.config file is no longer supported.

Changing WombatOAM's node name and IP address

WombatOAM runs on 127.0.0.1 by default and uses the wombat@127.0.0.1 node name, but you can change this behaviour to run WombatOAM on any required IP and using any node name.

**Important:** Changing the node name/IP address creates a new empty database. The new database will be placed in different directories within the wombat directory. It also overwrites any changes made to sys.config and vm.args. However, it is possible to restore the old database into the new one with the help of the changenodename.es and restore_backup.es scripts.

To change the name/IP address (e.g. from 'wombat@127.0.0.1' to 'wombat@192.168.1.1'), please follow these steps:

1. To retain the database execute scripts/change_node_name.es wombat@127.0.0.1 wombat@192.168.1.1 while WombatOAM is running. The script backs up the entire database so it's execution time depends on how many metrics WombatOAM collected so far.
2. Ensure that WombatOAM is stopped.
3. Export the environment variable: export WOMBAT_NODENAME=wombat@NEW_IP
4. [Optional step] If you are moving the WombatOAM installation to a new machine copy the entire installation in this step. The database is saved in rel/wombat/wombat directory so it will be copied along.
5. Restart WombatOAM with ./wombat start.
6. Restore the database by executing scripts/restore_backup.es
   mnesia_backup_wombat@192.168.1.1 wombat@192.168.1.1

If WombatOAM's cookie setting was changed from default value, you can specify that as the optional last argument. For example if the cookie is mycookie:

scripts/change_node_name.es wombat@127.0.0.1 wombat@192.168.1.1 mycookie

and

scripts/restore_backup.es mnesia_backup_wombat@192.168.1.1 wombat@192.168.1.1 mycookie

Local ports used by WombatOAM

WombatOAM binds to one local port. This can be changed e.g. if another application uses the same port or if you want to run two WombatOAM instances on the same machine. (In the latter case, the node name also needs to be changed; see the previous section.)

- Dashboard HTTP port (default: TCP port 8080): This can be changed in rel/wombat/files/wombat.config. Restarting WombatOAM is enough for this change to take an effect.

Logging configuration

The following options can be used to configure logging:

```erlang
% size of one log file in bytes
{set, wo_logs, wo_logs_disklog, log_file_size, 500000}.

% size of all logs in bytes
{set, wo_logs, wo_logs_disklog, log_max_size, 100000000000}.

% A list of filter expressions used to configure log entries
% which will never be deleted
{set, wo_logs, wo_logs_disklog, backup_filters, []}.

% Use one of the following expressions for filtering:
% {originator, Originator :: binary()}
% {node_id, NodeId :: binary()}
% {node_family_id, NodeFamilyId :: binary()}
% {message, Message :: binary()}

% Configuring the web interface

The default web dashboard can be accessed on port 8080 under HTTP.

{set, wo_rest, http_port, 8080}.

Change the value of http_port to the port on which the web dashboard and REST API should listen.

SSL is disabled by default, nonetheless, it is supported by WombatOAM. If SSL is enabled, the dashboard uses a self-signed root certificate by default that is considered 'not trusted' by browsers. If you need to
change any related configuration, you can edit the `wombat.config` file.

```{set, wo_rest, https_port, 8443}.```

Change the value of `https_port` to the port on which the web dashboard and REST API should listen.

```{set, wo_dashboard, enable_force_ssl, true}.```

To disable SSL, change the value of `enable_force_ssl` to `false`.

```{set, wo_dashboard, force_websocket_ssl, true}.```

To force a Websocket over SSL.

```{set, wo_rest, cacert, "/absolute_path/to/certificate/authority/file"}.```

Change the value of `cacert` to the absolute path of the file containing PEM encoded CA certificates (trusted certificates used for verifying a peer certificate).

```{set, wo_rest, cert, "/absolute_path/to/certificate/file"}.```

Change the value of `cert` to the absolute path of the file containing the PEM encoded user certificate.

```{set, wo_rest, key, "/absolute_path/to/private/key/file"}.```

Change the value of `key` to the absolute path of the file containing the user's private PEM encoded key.

```{set, wo_rest, enable_authentication, true}.```

In order to switch off authentication change the value of `enable_authentication` to `false`.

```{set, wo_rest, default_user_tags, all}.```

In order to change the tags that new users will be created with or the tags that are used by the REST layer when the authentication is switched off, change the value of `default_user_tags` to either `all` or a list of tags represented as binary strings. If `all` is set, all information available in WombatOAM will be retrieved, otherwise metrics, notifications and alarms that are not tagged with any of the listed tags will be removed from the results.

**Security modes**

WombatOAM can run in multiple security modes which present some requirements for its configuration or restrictions in its capabilities.

By default it starts up in `Local-only` security mode when there are no configuration requirements but usage is restricted to monitor only nodes running on the same host as WombatOAM. The Dashboard and REST
When WombatOAM node name is changed to include a hostname or IP address other than the loopback address (see Changing WombatOAM’s node name and IP address) WombatOAM will switch to operate in **Distributed** security mode. In this mode any node can be monitored but certain security settings are required in order to enforce production-level security:

- Authentication must be enabled (It is strongly advised to change the default password of the admin user at first login)
- If HTTPS is enabled certificates must be explicitly configured
- If the Dashboard should be accessible outside a private network and HTTPS is not enabled it is strongly advised to place a proxy (like Apache or NGINX) in front of WombatOAM. (See sections with proxy configuration examples below.)

**Sample configurations**

This is a sample configuration for Local-only mode turning off both https and authentication:

```erlang
%% customise http port
{set, wo_rest, http_port, 8080}.

%% disable authentication
{set, wo_rest, enable_authentication, false}.

%% disable SSL - use plain HTTP
{set, wo_dashboard, enable_force_ssl, false}.
```

And this is another one for Distributed mode with custom certificates:

```erlang
%% location of PEM encoded CA and host certificates
{set, wo_rest, cacert, "/absolute_path/to/certificate/authority/file"}.
{set, wo_rest, cert, "/absolute_path/to/certificate/file"}.

%% location of PEM encoded unencrypted private key
{set, wo_rest, key, "/absolute_path/to/private/key/file"}.

%% customise https port
{set, wo_rest, https_port, 8443}.

%% enable SSL/HTTPS
{set, wo_dashboard, enable_force_ssl, true}.

%% enable authentication
{set, wo_rest, enable_authentication, true}.
```

**Short node names**

WombatOAM can manage nodes with long names and short names at the same time. It is not necessary to change the name of the WombatOAM node to manage nodes with short names.

When adding short-named nodes, use the names that are returned by the `node()` function, i.e. `<short node name>@<host>` where `<host>` is the first part of the host name.

**Opening firewall ports**

This section describes which ports need to be open if there is a firewall
This section describes which ports need to be open if there is a firewall between WombatOAM and the managed nodes.

WombatOAM connects to the managed nodes via Erlang distribution, which needs the port used by EPMD (Erlang Port Mapper Daemon) to be open. EPMD uses TCP port 4369 by default.

You also need to make sure that the Erlang Distribution traffic can flow between WombatOAM and the managed nodes, for which there are two ways:

- Enable all incoming TCP ports from WombatOAM to the managed nodes. (i.e. WombatOAM should be able to open new TCP connections towards the managed nodes.)
- Set a custom TCP port range for Erlang Distribution on the managed nodes, and set those ports in the firewall as incoming ports towards the managed nodes.

Details for the second option: WombatOAM communicates with the managed nodes via Erlang Distribution. By default, Erlang does not specify any port range to be used for Erlang Distribution traffic. Instead, Erlang uses the first available port in the system. This can be changed by configuring the `inet_dist_listen_min` and `inet_dist_listen_max` configuration parameters of the kernel Erlang application:

```erlang
{kernel, [{inet_dist_listen_min, 9100}, {inet_dist_listen_max, 9200}, ...]}
```

This modification needs to be performed on the managed nodes. In most Erlang applications, it can be set in the `sys.config` file. In case of Riak 1.x nodes, this setting can be set in the `app.config` file. In case of Riak 2.x nodes, it can be set in `riak.conf`.

### Configuring Apache

If you would like to access WombatOAM via Apache as a proxy, the following Apache configuration file example shows how to forward both the HTTP and the web socket traffic to WombatOAM. It assumes WombatOAM is listening on `127.0.0.1:8080`, and that it will be accessed under `http://<my_server>/wombat/`). The `/wombat/` URL path can be changed by modifying the rules.

```apache
<VirtualHost *:80>
    ServerName localhost
    ProxyRequests Off
    ProxyPreserveHost On
    RewriteEngine On
    RewriteRule ^/wombat$ /wombat/ [R=301,L]
    ProxyPass /wombat/api/core/ws ws://127.0.0.1:8080/api/core/ws
    ProxyPassReverse /wombat/api/core/ws ws://127.0.0.1:8080/api/core/ws
    ProxyPass /wombat/ http://127.0.0.1:8080/
    ProxyPassReverse /wombat/ http://127.0.0.1:8080/
    LogLevel debug
</VirtualHost>
```

### Configuring NGINX

If you would like to access WombatOAM via NGINX as a proxy, the
following NGINX configuration file example shows how to forward both the HTTP and the web socket traffic to WombatOAM. It assumes WombatOAM is listening on 127.0.0.1:8080, and that it will be accessed under http://<my_server>/).

```
location / {
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
    proxy_pass          http://127.0.0.1:8080;
    proxy_read_timeout  1800;
}
```

**Running WombatOAM in the foreground**

Executing `./wombat start` will start the WombatOAM server in the background and return the prompt. Using `./start-fg.sh`, the WombatOAM server can be started in the foreground, so that it will not return the prompt, only when the server terminated. Instead, it will print the error and warning log entries to the standard output while the WombatOAM server is running. This is useful for example when running WombatOAM inside a Docker container.

**Configuration file format**

The configuration entries are organized into a tree. `wombat.config` contains a number of tuples, each of which modifies one leaf or branch of this tree. Leaves and branches can be replaced (with the `set` command) and deleted (with the `unset` command). The elements of the tuple (after the command) specify the path from the root of the configuration tree towards the leaf or branch to be modified or deleted. In case of the `set` command, the last element of the tuple is not part of the paths, but it defines the new value for the leaf or branch.

The available commands in `wombat.config` are the following:

1. `{set, AppName, ParamName, Path1, ..., PathN, Replacement}`.
   Sets the value of a given configuration entry to the given value.

2. `{replace, AppName, ParamName, Path1, ..., PathN, Replacement}`.
   Sets a given configuration entry to the given value. The difference between `set` and `replace` is that the former sets only the value, while the latter the entry itself. Compare `{set, myapp, myparam, a, 1}` and `{replace, myapp, myparam, a, {a, 1}}`, which perform the same task. The latter can be used to set not only a pair, but also a longer tuple as a configuration entry.

3. `{unset, AppName, ParamName, Path1, ..., PathN}`.
   Deletes the given configuration entry.
### Configuration reference

- **wo_dashboard/enable_topo_graph (boolean())**: Whether the dashboard should show the "View node graph" subtab in the Topology tab.

- **wo_rest/http_port (integer())**: The HTTP port of the REST interface and the Dashboard.

- **wo_rest/hide_cookie (boolean())**: Whether the cookie values of nodes should be hidden on the dashboard and REST interface. Default is false.

- **wo_license/license (string())**: The license key to be use by WombatOAM.

  ```erlang```
  ```
  erlang {set, wo_license, license, "MYLICENCE"}.
  ```
  ```

- **wo_core/node_monitor_max_wait (integer())**: The maximum number of milliseconds that WombatOAM should wait between trying to reconnect to "DOWN" nodes.

- **wo_metrics/backend (mnesia | seq_file)**: Specifies which back end to use.

- **wo_metrics/gauges**: Specifies what gauges should be visible on the "Metrics/Gauges" tab, and what their minimum and maximum values for different nodes should be. The node names are matched against the specified regular expressions, and the first match is used.

  Example:

  ```erlang```
  ```erlang```
  ```
  {set, wo_metrics, gauges,
   [[total_memory,
     [range,
      [{"wombat@127.0.0.1", {0, 1500000000}}, % 100MB
       {"riak.*", {0, 200000000}}, % 200MB
       {".*", {10000000, 30000000}}]}], % 10-30MB
     [cpu_avg1,
      [range,
       [{"{node}"}, {0, 5}],
       {"riak.*", {0, 2}}]}],
    [{riak_core, handoff_timeouts},
     [range,
      {"riak.*", {0, 10}}]]].
  ```
  ```erlang```

- **wo_metrics/graphite**: Configures the connection to Graphite. If not configured, WombatOAM will not push metrics to Graphite.

  Example:

  ```erlang```
  ```erlang```
  ```
  {set, wo_metrics, graphite,
   [{carbon_host, "127.0.0.1"},
    {carbon_udp_port, 2003}]}.
  ```
  ```erlang```

- **wo_metrics/datadog**: Configures the connection to Datadog. If not configured, WombatOAM will not push metrics to Datadog.

  Example:

  ```erlang```
  ```erlang```
  ```
  {set, wo_metrics, datadog,
   [{host, "localhost"},
    {port, 8125},
    {send_metrics, true},
    {send_notifications, true}].
  ```
  ```erlang```
• wo_metrics/threshold_templates: Configure templates for threshold based alarms. See details in Threshold based alarms

• wo_metrics/threshold_sets: Configure rule sets for threshold based alarms. See details in Threshold based alarms

• wo_alarms/repeat_notifications (boolean()): If an alarm is raised, WombatOAM sends a notification about it. If this option is true, WombatOAM will also send notifications when the alarm is re-raised. You may turn this off if you do not want to get notifications when alarms are re-raised without first being cleared.

• wo_alarms/ignore_missing_module (string): The modules matching the given regular expression will not appear in the list of missing modules

• wo_alarms/ignore_different_module_versions (string): No different module versions alarm will be raised for module names which match the given regular expression.

• wo_alarms/ignore_missing_application (string): No missing application alarm will be raised for application names which match the given regular expression.

• wo_alarms/ignore_different_application_versions (string): No different application versions alarm will be raised for application names which match the given regular expression.

• wo_alarms/ignore_different_timezones (boolean): No different_timezones alarm will be raised if it is true.

• wo_plugins/capabilities_period (integer()): Specifies the frequency used to retrieve the capabilities from the plugins running on the managed node. Metrics created dynamically on the managed node using Folsom or Exometer will only start appearing on the dashboard after a new capabilities check is performed.

• wo_plugins/sampling_default: Specifies the retention rules for gathering and keeping metrics from the nodes through the plugins running on the managed node.

The format of the retention rules is similar to the one used by Graphite. For example, "1m:1h" means that each data point represents 1 minute, and you want to keep enough data points so that they add up to 1 hour of data.

The web interface uses the metrics stored in the first retention period to serve the "last hour" graphs, the second period to serve the "last day" graphs, and the third period to serve the "last week" graph. Note that if you set the second period to keep only the last 12 hours, for example, the first part of the "last day" graph will always be empty.

It is strongly recommended to keep all the 3 metric series. However, you can freely modify the data frequency (the number before the colons).

Example:

{set, wo_plugins, sampling_default, "1m:1h 5m:1d 15m:7d"}.

This is the default value used for all the plugins, but it is also possible to specify a value following this format by adding a tuple \{sampling, SamplingValue\} to the list of options for the plugin in order to override the default value for that specific plugin.

• wo_plugins/hidden_config_params: Specifies which configuration parameters to hide. E.g. if you have a myapp application that has a password configuration parameter, and you don't want its value to be shown to the WombatOAM users, you can add the following line
to wombat.config:
{set, wo_plugins, hidden_config_params, [{myapp, password}]}.

If you want a plugin to use a different list than the other plugins, that can
be achieved by setting the hidden_config_params plugin option:
{set, wo_plugins, plugins, myplugin, hidden_config_params, [{myapp, password}]}.

- wo_plugins/hidden_ets_tables: Specifies which ETS tables to hide.
  E.g. if you have an ETS table called passwords, and you don't want
  its content to be shown to the WombatOAM users, you can add
  the following line to wombat.config:

{set, wo_plugins, hidden_ets_tables, [passwords]}.

If you want a plugin to use a different list than the other plugins, that can
be achieved by setting the hidden_ets_tables plugin option:
{set, wo_plugins, plugins, myplugin, hidden_ets_tables, [passwords]}.

- wo_plugins/plugin_action_verbosity: This is a list that specifies
  the verbosity level for different plugins. When a plugin starts, sends
  metrics, etc, we might generate a notification based on the log
  level that belongs to this action and verbosity configured for the
  plugin.

Each plugin has a verbosity level and there is also a default verbosity
level. The following example configuration will generate plugin
notifications for events on a warning level or above for all plugins, except
for the builtin_metrics plugin, where even debug-level notifications will be
generated:

erlang {set, wo_plugins, plugin_action_verbosity, default, <<"warning">>>}. {set, wo_plugins, plugin_action_verbosity, builtin_metrics, <<"debug">>>}.

Note that the value of this configuration option is always read on-
demand by WombatOAM when needed, so modifying it in a running
WombatOAM will take immediate effect. For example executing the
following line in the WombatOAM shell will start generating debug-level
notifications for the observer_ets plugin, while setting back the verbosity
of all other plugins to the default value:

erlang application:set_env(wo_plugins, verbosity, [{observer_ets, <<"debug">>>}).

Or, if you manage WombatOAM with WombatOAM, you can open the
WombatOAM Dashboard, go to Topology → WombatOAM node →
Applications → choose an application → "Set a configuration parameter"
→ Fill in the following data and click Add:

- "Configuration parameter name": verbosity
- "The value to be set": [{observer_ets, <<"debug">>>}

- wo_plugins/plugins: List of configuration items for the plugins
  that run on the managed node. There must be one
  configuration item for each available plugin. The configuration
  elements have the format {PluginName, Apps, ExtraModules,
  Options} where PluginName is the short name of the plugin,
  Apps is the list of OTP applications the plugin depends on
  (those apps must be running on the managed node for the
  plugin to start), ExtraModules is the list of additional modules
  included in the plugin besides the main module, and Options is
  the list of options specific for that plugin. Within that list of
  options it is possible to specify the sampling and retention
  rules specific to a plugin as explained in the previous section.
Regarding the list of Apps it is either a list or a tuple of two-tuples of the form \{(AppName, RegularExpression)\} where the regular expression provided is matched using the functionalities of the Erlang re module with the application version. (See Erlang documentation). The plugin will only start when there is a match as a result of the evaluation. If we intend to match any version of the app, the ".*" regular expression can be used, as in the following example.

Example:

```
{set, wo_plugins, plugins,
 [{mnesia,
   [{mnesia, ".*"}>,</mnesia>],
   [wombat_plugin_mnesia_metrics],
   [{activity_events, false},
    {table_events, none},
    {table_metrics, all},
    {metrics_period, 5000}]}
}.
```

- `wo_orch/private_key (string())`: The path to the private key to be used when connecting to an instance. Example: "/Users/myuser/.ssh/id_rsa"

- `wo_orch/public_key (string())`: The path to the public key to be used when connecting to an instance. Example: "/Users/myuser/.ssh/id_rsa.pub"

- `wo_orch/home_dir (string())`: The path to be contain the deployed nodes on the managed hosts. It is the home directory of the user by default.

- `wo_orch/cloud-prefix (string())`: A prefix to be used when naming instances, SSH keys, etc. in the cloud. For example, if you specify "mycloud-.", all instances will be named "mycloud-<UUID>".

- `wo_orch/hook_dirs ([string()])`: Directories that may contain modules (BEAM files) that should be loaded into WombatOAM. These modules can then be called from deployment hooks.

Example:

```
{set, wo_orch, hook_dirs, ["/home/user/my_wombat_hook_dir"]}
```

- `wo_logs/mail_sender (string())`: The email address used as the From field when emails are sent. Example: "user@mail.com"

- `wo_logs/mail_recipients ([string()])`: A list of email addresses where mails are sent. Example: ["person@mail.com", "another.person@mail.com"]

- `wo_logs/mail_severity (atom())`: A severity value specifying the minimum required severity a notification must have in order for it to be emailed. Example: error

- `wo_logs/gen_smtp_options`: The configuration values for the SMTP client used, gen_smtp.

Example:

```
{set, wo_logs, gen_smtp_options, 
 [{relay, "localhost"},
  {username, "username123"},
  {password, "password123"},
  {port, 25}]}.
```
- `wo_logs/graylog`: The configuration values for Graylog log management tool, `graylog`.

Example:

```erlang
{set, wo_logs, graylog, [
    {host, "localhost"},
    {port, 12201},
    {inet_family, inet}
    %% ipv4 = inet | ipv6 = inet6
    {level, info},
    %% level = debug | info | notice | warning | error | critical | alert | emergency
    {additional_fields, [{service, "WombatOAM"}]}],
    {send_notifications, all}
}.
```

- `wo_logs/number_of_logs_per_page`: By default, the number of logs that are returned is 20. But this value could be overridden by setting the parameter `number_of_logs_per_page` in the `wombat.config` file as follows

```erlang
{set, wo_logs, wo_logs_rest_handler, number_of_logs_per_page, 50}.
```

Though the maximum and minimum number of logs that could be returned is 100 and 20 respectively. So, any value for `number_of_logs_per_page` greater than 100 or less than 20, will not take effect.
Using Docker images containing WombatOAM

You should already have access to the Docker image we created, if not, please get in touch. The base of the Docker image is the official Erlang 17.5 built by Docker (erlang:17.5-slim) (see The Docker Hub for details). The prebuilt Docker image is also available on DockerHub, please contact your Erlang Solutions sales representative or wombat@erlang-solutions.com for the access details.

Due to limitations in the Erlang VM currently it's not possible to connect to nodes which are not in the same network as the Wombat container (as in, only exposing ports). Use docker-compose or docker stack overlay networks or similar solutions to start containers in the same network.

Starting WombatOAM in Docker for the first time

By default the script only creates the image file with a license and prints instructions on how to load and start it. An example:

1. Import the image locally by running
   
   ```
   docker load --input wombat_<version>.image.tar
   ```

2. To start the created Docker image, use the
   
   ```
   docker run -tid -p 8080:8080 --name wombat-2.9.0 wombat:<version>
   ```

   command. This will create and start a container with wombat:2.9.0 from the wombat:2.9.0 image.

   If you wish to use a Docker image file not containing a license key, then a directory containing the license key has to be mounted:

   ```
   docker run -tid -p 8080:8080 \
   -v $PWD/lic_dir:/wombat-2.9.0/lic_dir \
   --name wombat-2.9.0 wombat:2.9.0
   ```

   If you wish to store the logs and the data on the host, outside the Docker container, then two directories have to be mounted, one for the data and one for the logs:

   ```
   docker run -tid -p 8080:8080 \
   -v $PWD/data:/wombat-2.9.0/rel/wombat/wombat/data \
   -v $PWD/log:/wombat-2.9.0/rel/wombat/wombat/log \
   --name wombat-2.9.0 wombat:2.9.0
   ```

Starting already created container

If the Docker container is already created, just use

```
docker start wombat-2.9.0
```
To remove the Docker container and image from the system, use the

```
docker container rm wombat-2.9.0
docker image rm wombat:2.9.0
```

commands.

**Starting PyWombat commands**

To start the `wombat_cli` command, use the

```
docker exec -i -t wombat-2.9.0 /usr/local/bin/wombat_cli
```

command. In integrations using `wombat_cli`, `wombat_check` or `wombat_zabbixplugin` (Cacti, Nagios, Zabbix) you have to use a command like above instead of the simple `/usr/local/bin/wombat_cli`.

**Useful commands**

To "connect" to (look into) the running container, use the

```
docker exec -i -t wombat-2.9.0 /bin/bash
```

command. This command starts a (root) shell inside the container.
Using the official AMI containing WombatOAM

We build AMIs from Wombat. Please get in touch at wombat@erlang-solutions.com if you have no access to them.

When starting the Wombat AMI you should put the instance into a Security Group which has the 80 and 443 ports open to be able to open the Web dashboard in your browser.

Once you have started the AMI, Wombat should be up and running. Try connecting to the public IP of the instance in your browser. It will present a self-signed certificate which is advised to be changed.

By default Wombat is started as wombat@SHORT_HOSTNAME_OF_MACHINE, for example wombat@ip-172-30-3-248. You can use this hostname to add Wombat to itself by using the cookie wombat.

You can check the status of Wombat by running:

```
$ sudo service wombat status
pong
Wombat is running: wombat@ip-172-30-3-248
```

You can find the password for the Wombat Web Dashboard in the /home/centos/Wombat_README.md file.

If you want to change the Erlang Distribution Network Interface in Wombat, open the /etc/default/wombat file and change the WOMBAT_NODENAME environment variable. Please note that changing the node name will create an empty Wombat instance. Use the command sudo service wombat stop before changing the node name.

If you can not add nodes to Wombat please check if TCP connections can be created between the nodes, for further information check the section Configuration/Opening firewall ports. You may have to configure EC2 Security Groups and iptables as well.

Make sure EPMD is not open to the world. It's advised to use a whole separate network for communication between Erlang nodes.

All Wombat user data is saved to the directory /wombat-data.
Integration with third-party tools

WombatOAM can integrate with third-party OAM tools such as:

- Graphite, Grafana, Cacti for metrics;
- Graylog for notifications;
- AppDynamics, Splunk and Zabbix for metrics and notifications;
- Nagios, Slack and PagerDuty for alarms;
- Logstash and Datadog for metrics, notifications and alarms.

Integration with Cacti, Zabbix and Nagios is done through PyWombat, a Python interface for WombatOAM.

The following integrations can be configured directly from the GUI: Slack, Graphite, Grafana, Graylog, AppDynamics, PagerDuty, Logstash and Datadog.

Select the **Admin** menu then **Configuration** tab. Under **Data Export**, you can configure the available integrations.

PyWombat

PyWombat is a Python interface for WombatOAM, and is required for integration with Cacti and Nagios.

Prerequisites

Python 3 and the setuptools package for Python.

The following command installs these prerequisites on Debian Linux:

```bash
$ sudo apt-get install python3 python3-setuptools
```

Installing the PyWombat package

The PyWombat package is shipped with WombatOAM in the `integrations` directory.

To install the PyWombat package:

1. Go to the `integrations` directory and extract the `pywombat.tar.gz` archive:
   ```bash
   sh $ tar xzf pywombat.tar.gz
   ```
2. Change to directory of the extracted archive, then run the
The commands wombat_cli, wombat_nagioscfg and wombat_check will be installed in /usr/local/bin (on Debian).

For information on using the wombat_cli command, run wombat_cli --help:

```bash
$ wombat_cli --help

Script to access to WombatOAM functionality

optional arguments:
  -h, --help            show this help message and exit
  -H HOST, --host HOST  
  -p PORT, --port PORT  
  -s, --https
  --user USER
  --password PASSWORD
```

Use tags for narrowing down the set of metrics, notifications and alarms pushed to integrations using PyWombat. You can change the active tags set for the user that is used by PyWombat.

**Integration with Cacti**

Cacti is an open-source web-based tool using RRDTool as its backend for data storage and graphing. Integrating WombatOAM with Cacti requires PyWombat – see "PyWombat" above.

**The wombat_cli script**

This script provides basic command-line access to the REST API of WombatOAM.

To initiate a session, enter the following (substituting WombatOAM's IP address after host):

```bash
$ wombat_cli --host 33.33.33.1 --port 8080
```

If you're executing Wombat in a Docker container, you need to execute the wombat_cli inside the Docker container. Please see the instructions on starting PyWombat commands in Docker in the Docker chapter.

The following is a sample session to discover nodes and metrics provided by them:

```
WombatOAM@33.33.33.1> help
Documented commands (type help <topic>):
========================================
cacti_metrics  exit  get  help  list  metric  rawget  timequery
WombatOAM@33.33.33.1> help list
list [ nodes | alarms | families | node_metrics NODE-ID]
WombatOAM@33.33.33.1> list nodes
cookie          wo_test
deleted         False
domain_id       undefined
host            undefined
id              4f237968-8dbd-47b7-adbd-79ebbb9d2d20
```
Example for outputting data in a format expected by Cacti:

$ wombat_cli --host 33.33.33.1 --port 8080 cacti_metrics 4f237968-8dbd-47b7-adbd-79ebbb9d2d20 total_memory binary_memory atom_memory cpu
total_memory: 49457840 binary_memory: 394296 atom_memory: 662409 cpu: 12
$

Collecting data with Cacti

See also the "How to" section of the Cacti documentation.

1. Adding a new data input method

The first and the only WombatOAM-specific step of configuring Cacti:选
Select **Data Input Methods** in the menu on the left of the Cacti web console, then click **Add** and select **Script/Command** as **Input Type**.

Cacti needs the exact path to the *wombat_cli* script, and it must be executable by the user running Cacti. The command can be parametrised by using input fields in <-brackets. The script output must either be a single value (numeric) or a space-separated list of Key: Value pairs.

The screenshot below shows a fairly general setup for retrieving any single metric for a managed node from WombatOAM:
2. Create a new data template

This step is not strictly necessary, but speeds up adding new data sources later.

3. Adding data sources

4. A more complex example -- collecting several metrics in one step

First, we'll need an input source with multiple fields:
Then a template:

The graph in WombatOAM:

The graph in Cacti:
Integration with Nagios

Nagios is a system and network monitoring application. Integrating WombatOAM with Nagios requires PyWombat – see "PyWombat" above.

Configuration

1. Generate configuration for Nagios:
   ```
   sh $ wombat_nagioscfg --host <WOMBAT-HOST> --port <WOMBAT-PORT> > wombat.cfg
   ```
   Specify the -s option to use SSL. If you're executing Wombat in a Docker container, you need to execute the wombat_nagioscfg inside the Docker container. Please see the instructions on starting PyWombat commands in Docker in the Docker chapter.

2. Inspect the generated wombat.cfg and modify it if necessary, especially if you're executing Wombat in a Docker container, because in that case the wombat_check command will not be available and have to use docker exec (see the instructions on starting PyWombat commands in Docker in the Docker chapter).

3. Put wombat.cfg in the directory where your Nagios configuration files are stored (for example, etc/nagios/conf.d/ on Debian).

4. Restart Nagios (service restart nagios3 or /etc/init.d/nagios3 restart).

Troubleshooting

If you experience problems, try the following:

- Check if the installation of Pywombat is correct: the commands wombat_check and wombat_nagioscfg should be in the PATH.
- Check if WombatOAM can be accessed. Make sure to use the correct setting for connecting. ```
  ```
  ```
  ```
  Script to access to WombatOAM functionality
  ```
  ```
  optional arguments: -h, --help show this help message and exit -H HOST, --host HOST -p PORT, --port PORT -s, --https -user USER --password PASSWORD
  ```
  ```
  $ wombat_check --host badhost check wombat WOMBAT CRITICAL
  Error getting data from WombatOAM at https://badhost:8443/api/help/about. Reason:
  ```
  ```
  $ wombat_check --https --host goodhost check wombat WOMBAT OK pre-1.5.1 - 0 alarm(s) present - https://goodhost:8443/```
  ```
  ```
  Make sure that Nagios has restarted and has loaded the new configuration file.

Integration with Graphite
WombatOAM can push all collected metrics into Graphite via UDP.

After installing Graphite, you need to modify `carbon.conf` to listen for UDP packages:

```
ENABLE_UDP_LISTENER = True
UDP_RECEIVER_INTERFACE = 0.0.0.0
UDP_RECEIVER_PORT = 2003
```

The Graphite web dashboard listens on port 8002 by default, so open `http://localhost:8002/`.

Before trying to view WombatOAM data in Graphite, test that it actually captures UDP data by trying to send some UDP packages. You can use either of the following methods (socat is a more complex variant of netcat that is larger and more flexible):

```
1   2   3
4   5   6
7   8   9
10  11  12
13  14  15
16  17  18
19  20  21
22  23  24
25  26  27
28  29  30
31  32  33
34  35  36
37  38  39
40  41  42
43  44  45
46  47  48
49  50  51
52  53  54
55  56  57
58  59  60
61  62  63
64  65  66
67  68  69
70  71  72
73  74  75
76  77  78
79  80  81
```

Then add the following entries to `wombat.config`:

```
{set, wo_metrics, graphite,
  [{carbon_host, {127, 0, 0, 1}},
   {carbon_udp_port, 2003},
   {tags, all}]}.
```

Provide a list of tags to narrow down the set of metrics pushed to Graphite or use `all` to switch off filtering. If the `tags` config option is not set, all metrics will be pushed to Graphite.

### Integration with Datadog

WombatOAM can push all collected metrics, notifications and alarms into Datadog via UDP. WombatOAM alarms are forwarded to Datadog both as events and service checks. WombatOAM will push one event and one service check message for each raised alarm and another pair for each cleared alarm.

The installation of Datadog will add a local agent listening at UDP port localhost:8125. This agent will send to your personal Dashboard at the Datadog web site.

Before trying to view WombatOAM data in Datadog, test that it actually captures UDP data by trying to send some UDP packages. You can use either of the following methods (socat is a more complex variant of netcat that is larger and more flexible):

```
1   2   3
4   5   6
7   8   9
10  11  12
13  14  15
16  17  18
19  20  21
22  23  24
25  26  27
28  29  30
31  32  33
34  35  36
37  38  39
40  41  42
43  44  45
46  47  48
49  50  51
52  53  54
55  56  57
58  59  60
61  62  63
64  65  66
67  68  69
70  71  72
73  74  75
76  77  78
79  80  81
```

Then add the following entries to `wombat.config`:

```erlang
{set, wo_metrics, datadog, [[host, "localhost"],
  {port, 8125},
  {send_metrics, all},
  {send_notifications, all},
  {send_alarms, all},
  {environment, prod}]}.  
```

The environment will act as a tag, and the metric will be tagged either from prod or dev. (By default the dev is set).

You can disable sending metrics, notifications or alarms to Datadog by setting the corresponding config option to `false`.

Provide a list of tags to narrow down the set of metrics, notifications and alarms pushed to DataDog or use `all` to switch off filtering.

**Alarm events**

All events have the title "Wombat error"/"Wombat warning" and the following tags:

- **node**: erlang node name
- **hostname**
- **alarm_id**: the type of alarm
- **unique_alarm_id**: an integer or slug that uniquely identifies the raised and cleared pairs of a given alarm instance (eg. on the same node clearing `disk_almost_full` for `/tmp` won't clear the raised alarm for `/home`)
- **presence**: `new` or `clear`
- **severity**: WombatOAM alarm severity (same as lager severity levels)
- **origin**: WombatOAM subsystem that triggered the alarm

There is a separate event for each cleared alarm with `alert_type`: `success`.

**Alarm service checks**

For all alarms, service check name is `wombat.alarm` and have the same tags as the events.

**Monitoring WombatOAM alarms**

- As an example the following Custom Monitor can be set up:

  ```erlang
  "wombat.alarm".over("*").by("alarm_id","node","unique_alarm_id").
  last(2).count_by_status() 
  ```

  - It will have a separate trigger for each `unique_alarm_id`
  - `alarm_id` would not be strictly needed, it is only included for readable identification of the alarm type

**Integration with Logstash**

WombatOAM can push notifications, alarms and metrics to Logstash in JSON format via UDP.

First, configure Logstash to be prepared for receiving data pushed by WombatOAM. Note that the Logstash's config file should have the following content in its input block to parse WombatOAM's messages.
Then add the following entries to `wombat.config`:

```plaintext
{set, wo_int, logstash,
  [{host, "localhost"},
   {udp_port, 8126},
   {send_notifications, all},
   {send_alarms, all},
   {send_metrics, all}].
```

You can disable sending metrics, notifications or alarms to Logstash by setting the corresponding config option to false.

Provide a list of tags to narrow the set of metrics, notifications and alarms pushed to Logstash or use `all` to switch off filtering.

### Integration with Graylog

WombatOAM can push all collected logs into Graylog via UDP.

The installation of Graylog will add a local server listening by default for UDP GELF format messages at localhost:12201 and syslogs plain text at localhost:514. The Graylog admin web is accessible at localhost:9000.

Before trying to view WombatOAM data in Graylog, test that it actually captures UDP data by trying to send some UDP packages. You can use either of the following methods (socat is a more complex variant of netcat that is larger and more flexible):

```
MSG=\"{\"version\": \"1.1\", \"host\": \"server\", \"short_message\": \"graylog gelf test\"}\"

# After sending the package, netcat hangs, press Ctrl-C
```

Then add the following entries to `wombat.config`:

```plaintext
{set, wo_logs, graylog,
  [{host, "localhost"},
   {port, 12201},
   {inet_family, inet},
   % inet_family will tell WombatOAM if it should try to resolve host to an IPv4 or IPv6 address. - level describes the lowest severity level for which logs should be forwarded to Graylog. - additional_fields contains a key-value list that can be appended to each GELF message as additional
```
fields. - send_notifications a list of tags to further narrow the set of notifications pushed to Graylog or use all to switch off this optional filtering.

**Integration with Splunk**

Splunk is a data collection, indexing and visualization engine for operational intelligence. WombatOAM ships with an add-on for integrating with Splunk.

**Installing the Splunk add-on**

The Splunk add-on wombat.spl is located in WombatOAM’s integrations directory. To install the add-on, consult the following page of the Splunk documentation:

http://docs.splunk.com/Documentation/PCI/2.1.1/Install/InstallTechnologyAdd-ons

Follow the procedure under "Add a technology add-on from a local file"; select and upload wombat.spl.

This Splunk add-on facilitates defines a new data source type for forwarding metrics and notifications collected by WombatOAM to Splunk.

To set up data forwarding in Splunk:

1. In the **Settings** menu, select **Add Data**.
2. Choose the **monitor** option, then **WombatOAM**.
3. Fill in the form with the required information about the WombatOAM server. Splunk will fetch data using the REST API of WombatOAM. It therefore needs a valid user name and password if authentication is enabled in WombatOAM.
4. Click **Next** to finish the setup process.

The data source will forward all notifications and metrics to Splunk that are tagged with any of the active tags set for the user.

Alarms cannot be directly represented in Splunk, but each state transition of an alarm generates a notification too, so it is possible to reconstruct the active alarm list from the available information.

**Notifications**

WombatOAM notifications will have the wombat_notification sourcetype. A sample event may look like this:

```json
{  
  familyId:  b910348c-cdc9-4b8b-9f83-4e314dc8e959  
  familyName:  wombat 1.4.1  
  message:  New alarm raised.  
  Alarm id:  system_memory_high_watermark.  
  Additional information:  <<"High total memory usage of all OS processes">>  
  nodeId:  bf0a9542-0243-483a-b152-bbfa8e82c3a8  
  nodeName:  wombat@10.100.0.133  
  originator:  alarm  
  severity:  indeterminate  
  type:  logEvent
}
```

**Metrics**

WombatOAM metrics will have the wombat_metric sourcetype. A sample event may look like this:
It is also possible to chart a metric's value with Splunk. For example, the following search will show the median and maximum values from histogram *foo* over time for each managed node:

```
sourcetype="wombat_metric" metricName=foo | chart first("value.median") first("value.max") over _time by nodeName
```

**Integration with PagerDuty**

PagerDuty is an alarm aggregation and dispatching service for system administrators and support teams. WombatOAM can send notifications of alarms that are raised on the monitored nodes to PagerDuty. The alarms can then be forwarded to the user, using one of the following two PagerDuty services:

- **Email**: An email is sent to an address checked by PagerDuty. If a mail is found describing an alarm, the notification is forwarded.
- **PagerDuty API**: Using the PagerDuty API directly, REST calls are sent to a configured URL.

Use tags for configuring alarm subscriptions. Set the value `all` to send all alarms, or provide a list of tags (including custom tags) to send only those alarms that are tagged with *any* of the listed tags.

The following example configuration defines two rules:

1. All major and critical alarms should be forwarded to both services to trigger incidents with high priority;
2. All alarms related to any of the two production nodes should be sent as email to the given email address.

```
{set, wo_alarms, pagerduty_config, [
  [rule, [[tags, ["major"], "critical"]],
   [services,
    [pagerduty_api,
     [address, "Address of Pagerduty API"],
     [service_key, "Valid Pagerduty service key"],
     [client, "WombatOAM monitoring service"]],
    [pagerduty_mail,
     [address, "valid integration email address"],
     [sender, "sender email address"],
     [smtp_opts,
      [relay, "localhost"],
      [port, 2525],
      [ssl, true]]]]],
  [rule, [[tags, ["prodnode1"], "prodnode2"]],
   [services,
    [pagerduty_mail,
     [address, "valid integration email address"],
     [sender, "sender email address"]],
```
We would draw your attention to the customisation possibility of the node down alarm. For further details please refer to the Alarms section.

Currently, there are two ways to send alarms to the PagerDuty service: email and using the PagerDuty API.

**Email**

Note that using the email configuration requires a running SMTP server. Using the API functionality instead is recommended.

Emails to PagerDuty are configured in the section `pagerduty_mail` with the following options:

- **address**: The integration email address created in PagerDuty
- **sender**: The address of the sender
- **smtp_opts**: A proplist for configuring the SMTP server

**PagerDuty API**

Using the PagerDuty API is configured in the `pagerduty_api` section. The accepted parameters for using the PagerDuty API are as follows:

- **address**: URL of the PagerDuty API
- **service_key**: The service key generated when the PagerDuty service was created
- **client**: A string describing the service sending alarms

**Integration with Zabbix**

**Setting up Zabbix for WombatOAM**

1. **Install ZabbixAPI**

First you need to download and install Zabbix API which is available on github. You can install it easily with python's package management tool:

```
pip install zabbix-api
```

The latest version in 2015 September is 0.4 with pip installation and 0.3 on github, but sources don't differ from each other.

2. **Install pyWombat**

Integrating WombatOAM with Zabbix requires PyWombat – see "PyWombat" above.

3. **Create symlink in the External scripts' folder**

On Zabbix server create a symlink in the directory where Zabbix external checks are stored (by default: `/usr/local/share/zabbix/externalchecks`, see also the documentation). The symlink's name has to be `wombat_getter` and it points to the `wombat_zabbixplugin` executable which was installed with pywombat in the previous step. Check that execution and read permissions are allowed for the user who runs Zabbix server. Note that the script uses `zabbix_sender` which has to be installed on the server. If you're executing Wombat in a Docker container, instead of a symlink you might need to create a script that executes the `wombat_zabbixplugin` inside the Docker container. Please see the instructions on starting PyWombat commands.
in Docker in the Docker chapter.

4. Import the given template file in Zabbix

In the menu of the Zabbix web interface go to Configuration -> Templates, then click Import. Click on Choose file and select the Zabbix template file (zbx_wombat_template.xml) that can be found in the pywombat sources under doc/zabbix-integration. Click import.

5. Create a new host

In the menu of the Zabbix web interface go to Configuration -> Hosts, then click Create host. Host needs to be exactly in one group which corresponds with the host name. For example you can use wombat as host name and group name, visible name can be something generic, such as WombatOAM general. Before finishing this step you need to add this host to the template described in step 4. Click on the Templates tab (next to Host) in the 'Link new templates' section, and click on the Select button. In the popup window select our template, then click on the "Add on Templates" tab. After setting these up, go back to Host tab and click Add.

6. Setup wombat_data_getter item

Now you have a new host with two items, called wombat_data_getter and wombat_logger_for_data_getter. wombat_data_getter calls frequently wombat_zabbixplugin, which was referenced in step 3. It has the following parameters with the default values:

- WHOST 127.0.0.1: WombatOAM address
- WPORT 8880: WombatOAM port
- WUSER admin: WombatOAM user
- WPASSWORD admin: WombatOAM password
- ZGROUP wombat: Zabbix group that you defined in
- ZSERVER http://localhost/zabbix: Url of the zabbix web interface
- ZUSER Admin: Zabbix user
- ZPASSWORD zabbix: Zabbix password

If one or more doesn't fit to your configuration, you can add your own: In the menu of the Zabbix web interface go to Configuration -> Hosts, select your new host, and click Macros. Here you can define your values for the above listed macros. For example type macro {$ZPASSWORD} and value your_password

Collecting data with Zabbix

After completing the "Setting up Zabbix for WombatOAM" section, you need to set up new hosts which will refer to WombatOAM nodes. Go to Configuration -> Hosts, then click Create host. The host needs to be in the same host group as the host created in step 3. The host name needs to be the same as the node's id in WombatOAM, that you want to monitor. You can add a visible name, then click Add. In a few minutes you will see that lot of applications, items and graphs will be generated under the newly created host.

Integration with AppDynamics

WombatOAM can send metrics and notifications via HTTP to a Simple Machine Agent (SMA) which sends them to the AppDynamics Controller. In wombat.config you must map each Erlang node monitored by Wombat to the Simple Machine Agent associated with it. Below we configured a single Erlang node. You can specify as many as you want in the list.
Here is a breakdown of the above.

- **rabbit@localhost**: The Erlang node whose data you want to send to AppDynamics. This is used to lookup the SMA configuration.
- **address**: The address of the HTTP server (started by SMA) that Wombat data is sent to.
- **port**: The port number of the HTTP server (started by SMA) that Wombat data is sent to. Defaults to 8125 if not supplied.
- **metrics_prefix**: The path of every metric sent will begin with this. Defaults to `"Custom Metrics"` if not supplied.
- **send_metrics**: If an incoming metric matches the tag specified here then it is sent to the SMA. To forward all metrics choose all, otherwise `[Tag]` where tag is a custom metric. If `send_metrics` and `send_notifications` are false then no data for the Erlang Node will be sent to AppDynamics.
- **send_notifications**: The same as above.

You have the option to configure a timeout. When no metrics have been received in Wombat in a given interval (defaults to 60 seconds) then all metrics accumulated will be sent to the SMA.

```
{set, wo_int, appdynamics_timeout, 120000}
```

To apply the configuration you must restart Wombat. After this open AppDynamics and navigate to "Servers" in the top menu bar. To view metrics click on "Metric Browser" and for notifications click on "Events". The integration was developed against version PRO43 of the AppDynamics Controller.

You can disable sending metrics or notifications to AppDynamics by setting the corresponding config option to `false`.

Provide a list of tags to narrow the set of metrics and notifications pushed to AppDynamics or use `all` to switch off filtering.
Integration with Slack

WombatOAM can push alarms to a Slack channel via an incoming web hook in JSON format.

First, create the incoming web hook at https://api.slack.com/incoming-webhooks (follow "setting up an incoming webhook integration"). Do note the Webhook URL as this needs to be configured in Wombat. Setup the channel name, user name, icon, etc. that you wish to use.

Then add the following entry to `wombat.config`:

```json
{set, wo_int, slack,
 [{slack_url, "https://hooks.slack.com/services/..."},
  {send_alarms, all}].
}
```

Use the webhook URL in the above example. When an alarm is raised, Wombat sends a message with this format to Slack:

```json
{
 "text": "[<severity>] <node-name>: <alarm-id> (raised|cleared)
 ,
 "attachments": [
  {"fallback": "[<severity>] <node-name>: <alarm-id> (raised|cleared)"},
  "color": "<color>",
  "fields": [
   {"title": "Probable cause",
    "value": "<probable-cause-text>",
   },
   "title": "Proposed action",
   "value": "<proposed-action-text>",
   "title": "Additional information",
   "value": "<additional-information>",
  ],
  "ts": <seconds-since-unix-epoch>
 ]
}
```

The format of the text and the fields are configurable.

An example:

```json
{
 "text": "[major] 'wo_test@127.0.0.1': system_memory_high_watermark raised",
 "attachments": [
  {"fallback": "[major] 'wo_test@127.0.0.1': system_memory_high_watermark raised",
   "color": "danger",
   "fields": [
    {"title": "Probable cause",
     "value": "An upper boundary of total memory usage of all OS processes has been reached."},
    {"title": "Proposed action",
     "value": "Install more memory or find culprits of memory leaks."},
    {"title": "Additional information",
     "value": "High total memory usage of all OS processesd5400bb643730e19f9024b65a0fe245907eb506aqout;"},
   ],
   "ts": 1502202229
 ]
}
```
The example message looks like this in the Slack channel:
[Disk almost full] raised on 'wo_test@127.0.0.1' with severity minor

**Probable cause**
A lower boundary of free space on the indicated disk has been reached

**Proposed action**
Free up disk space by deleting unnecessary files

**Additional information**

Yesterday at 1:56 PM

The color of the message is based on the severity of the alarm: red for critical and major alarms, orange for minor, warning or indeterminate alarms, green for cleared alarms.

**Configuration**

The following configuration options are available:

- **slack_url**: *required* The URL of the webhook, e.g. "https://hooks.slack.com/services/XXX/YYYY/ZZZ"
- **text_format**: The format of the text message. The following variables are replaced in the actual text message by their values:
  - $severity: The severity of the alarm (e.g. critical).
  - $alarmid: The alarm ID of the alarm (e.g. disk_capacity_major).
  - $nodename: The name of the node where the alarm was raised (e.g. 'wo_test@127.0.0.1').
  - $status: It is raised if the alarm is new or cleared if the alarm is cleared. The default is "[$severity] $nodename: $alarmid $status".
- **send_alarms**: A list of tags to narrow the set of alarms pushed to Slack or use all to switch off filtering. By default all alarms are pushed to Slack.
- **fields**: A list of fields that will be shown as attachment fields in the message, in the order of their appearance. Possible values:
  - probable_cause: shows the Probable Cause of the alarm.
  - proposed_action: shows the Proposed Repair Action of the alarm, or None if there's none.
  - additional_information: shows any additional information, if present. The default is to show Probable Cause, Proposed Repair Action and Additional Information.

**Integration with Amazon Cloudwatch**

WombatOAM can push metrics to Amazon Cloudwatch.

There are two ways of setting up the integration. The first one is using an environment variable and the second one is filling the necessary textboxes.

**First method**
- **WombatOAM is running within an ECS cluster**: By enabling this option it results in most of the textboxes are become disabled. This option uses an environmental variable, called AWS_CONTAINER_CREDENTIALS_RELATIVE_URI, to generate the necessary credentials. WombatOAM has to be started in the ECS container because the environment variable gets defined inside the container during the setup procedure.
Second Method

By using this method three parameters needed for the credentials to be generated. These parameters are the Access Key, Secret Access Key and the Region.

* **Access Key ID:** It can be found, if there is one already, or can be created under the *My Security Credentials* submenu. Accessible by clicking on the signed in user’s name dropdown menu. Learn more.

* **Secret Access Key:** When creating a new Access Key ID the Security Access Key is created simultaneously. Be aware that it can be seen only once and cannot be accessed later. Learn more.

* **Region:** It is essential to let WombatOAM know where to send the metrics. List of Regions with their Region Names:
  - us-east-2: US East (Ohio)
  - us-east-1: US East (N. Virginia)
  - us-west-1: US West (N. California)
  - us-west-2: US West (Oregon)
  - ap-east-1: Asia Pacific (Hong Kong)
  - ap-south-1: Asia Pacific (Mumbai)
  - ap-northeast-3: Asia Pacific (Osaka-Local)
  - ap-northeast-2: Asia Pacific (Seoul)
  - ap-southeast-1: Asia Pacific (Singapore)
  - ap-southeast-2: Asia Pacific (Sydney)
  - ap-northeast-1: Asia Pacific (Tokyo)
  - ca-central-1: Canada (Central)
**Optional Parameter**

- **Custom Namespace**: Providing a namespace is not crucial to start the integration, but it does stay available by using any method. If it is empty the system uses the default custom namespace which is `Wombat`. These namespaces must contain valid XML characters, and be fewer than 256 characters in length. Possible characters are: alphanumeric characters (0-9A-Za-z), period (.), hyphen (-), underscore (_), forward slash (/), hash (#), and colon (:).

Once the configuration is completed the integration is ready to start. If a node is connected to WombatOAM, its metrics are sent to the Amazon Cloudwatch service. The metrics can be found under the **Custom Namespaces** area under `Wombat` or under the provided custom namespace.

### Integration with InfluxDB

InfluxDB is a time series database designed to handle high write and query loads. WombatOAM can send metrics via HTTP to InfluxDB v2.

**Setup**

Before starting the integration the following five parameters have to be added to WombatOAM: *Host* *Port* *Token* *Organization* *Bucket* After typing in the parameters the integration validates all of them and responds with error messages if one or more of them are not added correctly.

**Token**

The token can be found through the GUI of InfluxDB or using influx CLI. In the navigation bar under the **Load Data** menu there is a submenu called **Tokens**. Clicking on a user’s token shows the token with the enabled authorizations.
Using the influx CLI executing the following shows the token. With option flags it is possible to filter the tokens.

```
influx auth find
```

Read more...

### Organization and Bucket

Both values can be the names or the ids of the corresponding parameters. The integration validates the using API calls in the case of valid host and port. The parameters can found through API calls, GUI or influx CLI. The bucket can be retrieved the same way as the organization. Read more at the InfluxDB documentation page.

- **API**

  - http://localhost:9999/api/v2/orgs
  - http://localhost:9999/api/v2/buckets

- **influx CLI** Executing the following commands shows the buckets and their ids. Using flags option can help filtering the results. The organization name or id has to be specified before executing the command.

```
influx buckets find
```

Executing the following commands shows the organizations and their ids. Using flags option can help filtering the results.

```
influx org find
```

- **GUI** Buckets can be found under the Load Data menu, under the Buckets submenu.

Furthermore in the address the id of the organization and the id of bucket is visible if a bucket is selected to edit its parameters. It follows the pattern that looks like this:

```
http://localhost:9999/orgs/{orgID}/load-data/buckets/{bucketID}/edit
```

The current organization can be found under the top menu right next to the user's name.

### Starting the integration

After typing in the parameters click on the save button and if everything is valid the integration starts sending metrics to InfluxDB. Use the tags to narrow the set of metrics pushed to InfluxDB or use all to switch off filtering.
Custom topology

WombatOAM automatically generates the node names and node family names that are used in the dashboard. Family names are derived from the release name of the node monitored by WombatOAM, while the node names are simply the name of the Erlang node. Nodes for which the same family name is generated are listed together.

The release name and node name do not always logically identify where a given node belongs. For example, monitored nodes with the same version in multiple Riak clusters will belong to the same family, instead of being grouped in the clusters to which they belong. If a different naming and grouping arrangement would be more logical for the system you are monitoring, you can override the default naming in WombatOAM, using the methods described below.

To override WombatOAM's default naming, you have to specify the relevant options in the configuration, create a callback module, and optionally use a script for upgrading the current node structure. These steps are described in detail below.

Configuration options

The `wombat.config` file has the following configuration options for how nodes and node families are named in WombatOAM:

```erlang
{set, wo_core, additional_paths, ["path/to/alias_callback"]}.
```

The `additional_paths` option specifies the location of the callback module used for naming nodes and node families. The path should end in a directory.

```erlang
{set, wo_core, alias_callback, callback_module_name}.
```

The `alias_callback` option specifies the name of the callback module. This module should be placed in a directory specified by the `additional_paths` option. The callback module makes it possible to implement fairly complex logic to grouping nodes into families and naming them as required. Naming the nodes and families is performed separately, so the names cannot depend on each other.

```erlang
{set, wo_core, node_family_rules, [Rule]}.  
```

Where

```erlang
Rule :: {FamilyName, [Condition]}.  
FamilyName :: atom()|string().  
Condition :: {application, loaded|loading|started|starting|running|start_p_false, App}  
| {application, isnot, loaded|loading|started|starting|running|start_p_false, App}  
| {environment, App, EnvVar, set|not_set}  
| {environment, App, EnvVar, equals, Value}.  
App :: atom().  
EnvVar :: atom().  
Value :: term().
```

If specified and `alias_callback` is not defined or does not apply to a given node, then the family name is first tried to be derived based on the list of `Rules` in order. If no rule applies, then the default family name is used. The rules are applied in the order they are specified in the config...
and the first matching rule will be used. The list of Condition tuples define what conditions must apply to the node in order to belong to the given family FamilyName. If all conditions apply, then the node belongs to the family FamilyName. For example, the condition \{application, loaded, mnesia\} specifies that an application named mnesia is loaded on the node, while \{environment, my_app, my_var, not_set\} states that the application named my_app does not have the environment variable my_var set. To make sure that this environment variable is set to a specific value, for example, 5 the condition \{environment, my_app, my_var, equals, 5\} can be given. Rules in the form \{application, isnot, State, App\} are to be interpreted as the given application is not in the specified state on the node. Each rule can contain any number of conditions and by definition any node matches the empty condition list. Defining either-or conditions is not supported, but they can be specified by duplication. For example

```erlang
{set, wo_core, node_family_rules, [
  \{"Test nodes", \[{application, running, test},
    \{environment, test, test_type, equals, unit\}\}],
  \{"Test nodes", \[{application, running, test},
    \{environment, test, test_type, equals, feature\}\}\}].
```

which describes that nodes that have the test application running on them with the environment variable test_type set to either unit or feature belong to the node family "Test nodes".

```erlang
{set, wo_core, automatic_node_restructure_at_startup, true}
```

The automatic_node_restructure_at_startup option specifies whether the nodes should be restructured at startup according to the rules in the callback module. If the rules do not change and the callback functions run without any side effects, then no restructuring occurs. Existing nodes' UUIDs are never changed. If the restructuring is set to true, then it is advisable to not disable the node_info plugin, otherwise the node will be excluded from the node family at startup.

### Alias callback interface

The alias callback module has to implement and export any of the following functions (a missing implementation will be substituted by the default implementation):

```erlang
-spec generate_custom_family_name(Node :: atom(),
                                 ScriptId :: {string(), string()}
                                 | {error, term()}) ->
                    {ok, string()} | {error, term()}.  
-spec generate_custom_node_name(node()) -> {ok, string()} | {error, term()}.  
```

Used for naming node families. The parameters are (1) the name of the node, and (2) the release name (either the name and version of the permanent release or the script_id - see the documentation of the release_handler and init:script_id/0). If an error occurs when capturing the release name, the following term is returned: \{error, no_scriptid\}. If the callback function is not implemented or exported, or if an error (crash) occurs in the function, the default naming logic is used instead. The callback should be free of side effects.
Used for naming the nodes. The parameter is the name of the node. If the callback function is not implemented or exported, or if an error (crash) occurs within the function, the default naming logic is used instead. The function should be free of side effects.

**Example module**

```erlang
-module(alias_callback).
%% API
-export([generate_custom_family_name/2,
          generate_custom_node_name/1]).
%%%=======================================================
======================
%%% External functions
%%%=======================================================
----------------------
%%--------------------------------------------------------
----------------------
%% @doc
%% Generate a name for a given node-family.
%% @end
%%--------------------------------------------------------
----------------------
-spec generate_custom_family_name(Node :: atom(),
                                   ScriptId :: {string(), string()}
                                   | {error, term()}) ->
                                   {ok, string()} | {error, term()}.
generate_custom_family_name(Node, {Name, Version}) ->
  case atom_to_list(Node) of
    "riak" ++ _ ->
      {ok, "Riak cluster"};
    "rabbitmq" ++ _ ->
      {ok, "Rabbit cluster"};
    _ ->
      %% Default behaviour
      {ok, Name ++ " " ++ Version}
  end.

%%--------------------------------------------------------
----------------------
%% @doc
%% Generate a name for a given node.
%% @end
%%--------------------------------------------------------
----------------------
-spec generate_custom_node_name(node()) ->
                                   {ok, string()} | {error, term()}.
generate_custom_node_name(Node) ->
  {ok, atom_to_list(Node)}.```

Adding a new node to WombatOAM

If the configuration settings are in place, and the callback module is compiled and located in the path, WombatOAM will load the module at startup.

All new nodes will be added according to the logic implemented in the
callback module. Existing nodes will not be affected. To restructure existing nodes, follow the workflow in the next section.

Restructuring existing nodes

**Workflow**

1. **Prepare the callback module and add it to the wombat.config.** The callback module will be loaded into the WombatOAM VM. This requires the configuration options to be present at startup. For changes to the configuration file to take effect, WombatOAM must be restarted. Do NOT set the `automatic_node_restructure_at_startup` option before performing a successful upgrade if the old WombatOAM version is lower than 1.14.0.

2. **Upgrade WombatOAM to the new version containing the feature.** Configuration changes will take effect after a WombatOAM restart. You can modify the configuration before performing the WombatOAM upgrade. If the node has already been upgraded, restart WombatOAM to ensure the configuration options are loaded.

3. **Run the node_restructure command-line script to review the proposed modifications.** The command-line script provides a fast feedback loop to enable modifying the callback module to mirror the desired node structure. The script automatically reloads the callback module.

4. **Enable restructuring in WombatOAM configuration.** The restructuring is only done at WombatOAM startup if the corresponding configuration option is enabled. If it is disabled, the restructuring is not performed, but new nodes are added according to the specified callback module.

5. **Restart WombatOAM.** On-the-fly restructuring would impose problems in some WombatOAM subsystems due to the concurrent nature of events in WombatOAM. Restarting WombatOAM and performing the restructuring during the initial phase eliminates the risk of clashing cached data between various subsystems.

Step 4 and 5 is done by the provided script.

**Command-line script to preview and apply restructuring**

The `node_restructure.escript` script lets you check whether the restructuring works as intended, reload the callback module, and commit changes. To run the script, you need `escript` in your PATH. If you have modified the port of EPMD for WombatOAM, you have to set the shell variable `ERL_EPMD_PORT` accordingly. E.g. set it to port 1234 in case of bash:

```
export ERL_EPMD_PORT=1234
```

To preview the changes, run the following: `escript node_restructure.escript preview`

```
$ escript node_restructure.escript preview
"Family1":
  'node1@domain': "Node 1",
  'node2@domain': "Node 2"
"Family2":
  'node3@domain': "Node 3",
  'node4@domain': "Node 4",
  'node5@domain': "Node 5"
```
Previewing has no effect on the WombatOAM node: no data is modified, no beam files are changed, and new nodes will keep using the callback module that is already loaded.

To reload the callback module, run the script with reload option:

```
$ escript node_restructure.escript reload
Reloaded.
```

Reloading the callback module performs a code change in the WombatOAM VM. Its effect is permanent: the new callback module will be used to name and group new nodes added to WombatOAM. Nodes that are already present are not changed. The script also shows the existing nodes in the new structure.

If you are satisfied with the result, set the above parameter in `wombat.config` and restart WombatOAM. The restructuring is done during the startup, after which WombatOAM starts normally. The configuration changes and restart procedure is automated with the `commit` option. Before the actual changes are made you will be prompted for confirmation.

```
$ escript node_restructure.escript commit
New nodes will be added according to new rules!
Existing nodes would look like the following after committing:
"Family 1":
  'node1@domain' : "Node 1"
  'node2@domain' : "Node 2"
  'node3@domain' : "Node 3"
"Family 2":
  'node4@domain' : "Node 4"
  'node5@domain' : "Node 5"
++-------------------------+
|WombatOAM will be RESTARTED!|
++-------------------------+
The following config option will be added to the `wombat.config`:
  {set, wo_core, automatic_node_restructure_at_startup, true}.
Are you sure to proceed? (yes/no) yes
Stopping wombat... ok
Starting wombat... ok
Changes are committed.
```

Note: if the automatic_node_restructure_at_startup is explicitly set to false in `wombat.config`, it will not be modified but a warning will be printed.

**Reverting your changes**

After the commit, the changes cannot be undone, but removing the callback module configuration option and committing again would restore the structure to system defaults. Preview is also available with system defaults.
WombatOAM Plugin Guide

What is a plugin?

Plugins for WombatOAM are user-supplied modules that extend its capabilities to monitor new Erlang applications. Plugins provide a way to extend WombatOAM and give it the ability to collect application-specific metrics, notifications, raise custom alarms and implement services. They can also be added to a binary WombatOAM release; this requires a restart of the system. Plugins run on the managed node and communicate with the WombatOAM server.

Viewing and managing plugins

The web dashboard shows the plugins that are running on each node. Click **Topology** → select a node → **Agents**. You can turn individual agents on or off on a per-node basis.

Configuring & deploying plugins

WombatOAM will start a plugin automatically on the managed node if the node is running the matching versions of the dependant Erlang applications declared for that plugin in one of the WombatOAM config files (`sys.config` or `wombat.config`).

For WombatOAM to find a plugin, the compiled BEAM files for the plugin need to be present in the `plugins` directory of the WombatOAM release when WombatOAM is started. This directory can be overridden: in the `wombat.config` file, set the environment variable `plugin_dir` of the `wo_utils` application:

```
{set, wo_utils, plugin_dir, "...path/to/plugins"}.
```

The main module of the plugin should be named `wombat_plugin_<APPNAME>.beam`. For example, a plugin for the `wo_test` application should be named `wombat_plugin_wo_test.beam`.

A plugin's settings need to be declared in WombatOAM's `wombat.config` file. For example, to enable the plugin for the `wo_test` application, `wombat.config` needs the following:

```
%% {PluginName, DependantApplications, ExtraModules, Options}
{replace, wo_plugins, plugins, wo_test,  
{wo_test,  
[[myapp1, "1.3"], [{myapp2, "."}, {myapp3, "."}]],  
[wombat_plugin_dummy_module],  
[\(test\_option, 42\),  
{required_wombat_apps, [{kernel, "^[3-9][0-9]*"}]}}]}.
```

The main plugin module is `wombat_plugin_wo_test` and there is a library module called `wombat_plugin_dummy_module` which gets loaded into the managed node. Module names must also have a `wombat_plugin_` prefix.

A dependant application is an application that must be running on the managed node in order to be able to use the plugin. A plugin can have more than one dependant application. In the example above, the `wo_test` plugin depends on `{myapp1, "1.3"}` and `{myapp2, "."}, `{myapp3, "."}`. This means that `wo_test` can be used if either:

- version 1.3 of `myapp1`; or
- any version of both `myapp2` and `myapp3`
are available on the target node. [{myapp2, ".*"}, {myapp3, ".*"}] is
known as a dependant application group, which means that all the
dependencies defined within the group must be met in order to be able
to use the plugin.

Extra options to the plugins can be passed via the Options property list.

This list should be used in case of the plugin's binaries requires some
extra applications to be loaded into the Wombat node. That can be
declared by the required_wombat_apps property that should list the
dependant applications. In the example above, the wo_test plugin will be
loaded into Wombat, only if Wombat is running on Erlang/OTP 17 or
newer versions as its binary contains op codes that can't be interpreted
by older Erlang versions (e.g. uses maps). If the property is not set,
Wombat always tries to load the plugin's binaries.

When developing a plugin, it is often useful to set the verbosity that
belongs to the plugin to the debug level:

```erlang
{set, wo_plugins, plugin_action verbosity, my_plugin, <<"debug">>}.
```

This means that notifications will be generated for many plugin actions,
such as starting, terminating, sending metrics and alarms to
WombatOAM, handling requests, etc. See more information about this
settings in the Configuration section.

Note that this setting may have a negative impact on WombatOAM's
performance. It should be only used while you are developing a plugin or
experiencing issues.

**Writing plugins for WombatOAM**

The main module of the plugin must implement the wombabt_plugin
behaviour by implementing its callback functions. Each plugin will run in
its own process and it has a state in which it can store information. The
callback functions init and terminate are called when the plugin is
started/stopped. The callback handle_info/2 is called when the plugin
process receives a message.

A plugin can report metrics, notifications, alarms and other information
by:

- Implementing the wombabt_plugin callbacks.
- Using the library functions in wombabt_plugin and
  wombabt_plugin_utils.

Reporting **metrics** is mostly callback-oriented: the callbacks
capabilities/1, live_metrics2comp_units/2 return the list of available
metrics, while the callbacks collect_metrics/1 and
collect_live_metrics/1 return the metric samples. If the list of available
metrics change, the plugin should call the
wombabt_plugin:announce_capabilities/1 function.

The **notifications, alarms** and **other information** are reported not via
callbacks but via calling library functions: wombabt_plugin:report_log/2 for
reporting notifications, wombabt_plugin:raise_alarm/2 and
wombabt_plugin:clear_alarm/1 for reporting alarms, and
wombabt_plugin:report_internal_data/2 for reporting other information.

A plugin can also implement services. First it has to report as capabilities
which services it implements (see Capabilities section). Then it has to
implement 3 callbacks to implement a request for a certain service (see
Callbacks section).

There are many useful library functions too, organised into 3 modules.
- wombat_plugin contains functions to implement the core
  wombat_plugin behaviour,
- wombat_plugin_utils includes general utility functions,
- wombat_plugin_services provides functions to create new services
  by implementing the wombat_plugin_services behaviour.

The plugin is started on the managed node, supervised by other
WombatOAM processes. Some general guidelines:

- Plugins are expected to generate a moderate amount of data;
currently, WombatOAM doesn't try to throttle plugins.
- As WombatOAM supervises the plugin processes, start-up
  notifications for the plugins and their supervisors might show up in
  logs on the managed node, for example, when SASL is started.

Types

The types used by the plugins are defined in the wombat_types module
and exported:

```
-define(CapabilityId, [binary()]).
-define(CapabilityInfo, metric_info() |
  notification_info() |
  alarm_info() |
  service_info()).
-define(CapabilityTag, binary()).
-define(CapabilityTags, list(CapabilityTag)).
-define(CapabilityInfoItem, {type, metric} |
  metric_info_item()).
```
%% The last element of the capability id of a metric, but as a term and not as a binary. E.g. 'Folsom cpu'.
%% From the plugin's point of view, this data type is used to administer live metrics: e.g. when a live metric is enabled on the dashboard, a list of metric_cap_id_last() items are send to the plugins to start sending the appropriate metric periodically.

%%% Capabilities: Notifications

%%% Capabilities: Alarms

%% Capabilities: Services

- type metric_cap_id_last() :: term().

- type notification_info() :: [notification_info_item()].

- type notification_info_item() :: {type, notification}
  | {description, binary()}

- type alarm_info() :: [alarm_info_item()].

- type alarm_info_item() :: {type, alarm}
  | {probable_cause, binary()}
  | {proposed_repair_action, binary()}
  | {severity, alarm_severity()}
  | capability_tags_item().

- type alarm_severity() ::
critical | major | minor | warning | indeterminate | cleared.

- type service_info() :: [service_info_item()].

- type service_info_item() :: {type, service_capability_type()}
  | {description, binary()}
  | {label, binary()}
  | {feature, term()}
  | {is_internal, boolean()}
  | {is_exclusive, boolean()}
  | {priority, integer()}
  | {arguments, [service_option_name()]
  | {options, [service_option()]

- type service_capability_type() :: configurator | explorer | executor.
-type service_option() :: [service_option_item()].

-type service_option_item() ::

  %% This key is used in the request.
  {option_name, service_option_name()}

  %% The label of the option on the Dashboard.
  | {option_label, binary()}

  %% The type of the option. It determines whether
  %r `option_values' or
  %% 'listitem_type' need to be also specified.
  %%
  %% If the option type is a list, then the list
  item_type field contains
  %% the types in the list. A list type will actu
  ally form a table. E.g.
  %% if the listitem_type field describes key1 an
d key2, then the user
  %% can fill in a table with two columns (key1 a
nd key2) and any number
  %% of rows.
  | {option_type, service_option_type()}

  %% The default value of the option.
  | {option_default, binary()}

  %% Whether the option can be seen and directly
set by the users.
  | {option_enabled, boolean()}

  %% The list of possible values that the option
can have. When type is
  %r not enum, it is an empty list.
  | {option_values, [OptionValue :: binary()]}

  %% This definition is recursive, but in reality
only the top service
  %% option can be a list, its children cannot. S
o the listitem_type
  %% will be empty for the children.
  | {listitem_type, [service_option()]}.

-type service_option_name() :: binary().

-type service_option_type() :: string | number | enum | l
ist.

%%%------------------------------------------------------
-----------------------
%%% Reporting
%%%------------------------------------------------------
-----------------------

--type capability_data() :: metric_data().

%%%------------------------------------------------------
-----------------------
%%% Reporting: Metrics
%%%------------------------------------------------------
-----------------------

-type metric_data() :: {metric, capability_id(), metric_t
ype(), metric_value()}

-type live_metric_data() :: {live_metric, capability_id()'
, metric_type(), metric_value()}.

-type live_metric_comp_unit() :: term().

-type metric_type() :: gauge | counter | histogram | mete
r | spiral | duration.
-\textbf{type} metric\_unit() :: numeric | byte | percentage.
-\textbf{type} metric\_value() :: term().

---

\textbf{Reporting: Notifications}

-\textbf{type} severity() :: binary().
\% The severity of a notification. The recommended values are the following:
\% critical, error, warning, info, debug.

-\textbf{type} log\_message() :: binary().
\% The text of a notification.

---

\textbf{Reporting: Alarms}

-\textbf{type} alarm\_id() :: term().
\% This is the same as alarm\_id() in elarm.hrl.

-\textbf{type} alarm\_add\_info() :: term().
\% This is the same as additional\_information() in elarm.hrl.

---

\textbf{Implementing: Services}

-\textbf{type} request\_args() ::
\[
\{\text{KeyBinStr} :: \text{binary()},
\text{ValueBinStr} :: \text{binary()},
\text{inner\_list} :: \text{proplist}()\}
\%
\]
\% A piece of input given by the user, which is used to execute a certain request.
\% The type of the key (as defined in the service capability of this request\_args term) defines the type of the value:
\% * For keys that have string, number, enum type, ValueBinStr is a binary.
\% * For keys that have list type, ValueBinStr is a list that contains inner lists. Each inner list is a proplist, and each inner list has the same keys (as defined in the capability).

-\textbf{type} display\_info() :: #display\_info{}.
\% A display info term describes for the GUI how to display streamed data. It is created by the plugin process.

-\textbf{type} display\_info\_option\_item() :: \{is\_interactive, boolean()\} |
\% Modifiers for the display\_info().

-\textbf{type} execution\_info() :: #execution\_info{}.
\% An execution info term describes for the wombat\_plugin behaviour how to
Capabilities

The metrics and services plugin interfaces use a list of capabilities to return information back to WombatOAM. When WombatOAM asks for information on available metrics, the capabilities/1 function of all plugins will be queried.

A plugin exposes a list of capabilities (capabilities()). Currently this is used to report the metrics that the plugin can report and services that can be requested from the plugin. Optionally, the alarms the plugin may raise can be reported. Each capability has an id (capability_id()) and a list that contains further information about the capability (capability_info()). An id is made up of a list of binary UTF-8 strings.

Metrics capabilities

In case of metrics capabilities the final component of the id is the name of the metric. The prefix list of the id (i.e. the list containing all elements of the capability id except the last one) becomes the name of the metric group.

Actual metrics samples either have the type metric_data() (in case of so-called collected metrics that are collected automatically) or live_metric_data() (in case of live metrics that are collected on-demand).
Note about the order of the entries in the capability list: When WombatOAM presents the metrics to the user, it shows them in the order they were received from the plugin in the capabilities callback or by calling `wombat_plugin:announce_capabilities`. If new metrics are added and reported later (either by capabilities or by `wombat_plugin:announce_capabilities`), WombatOAM will insert the new metrics. If metrics are deleted, they will be still shown to the user at least as long as WombatOAM stores samples from the metric. If the metrics are reordered, WombatOAM will prefer the new ordering. The recommendation is to keep a consistent order and not to reorder existing metrics though for two reasons:

1. It is better user experience to see the metrics always at the same place.
2. When some metrics are reordered and some are removed, WombatOAM is not always able to locate the correct position of the removed metrics, so they will be moved to unexpected locations.

Adding and removing metrics causes no problem, unless the same metrics appears at different places different times.

**Services capabilities**

For each service announced the plugin should declare its identifier, priority, type (configurator, explorer, executor), a label (displayed name on the dashboard) and description, whether the service is internal and exclusive, specification of the arguments of the service (label, type, default value; options field) and a subset of these which are the mandatory argument names (arguments field).

The feature field is the identifier of a service. Multiple plugins can implement the same service. WombatOAM aggregates the announced implementations of a service using the feature field. Then the generalised interface of the service is available for users to submit new requests. When a request arrives WombatOAM will try to initiate the request by asking the satisfied implementations in order. The implementations are ordered by priority and mandatory arguments count. This mechanism allows a way to override built-in services or implement more specific ones (e.g. a custom configuration service which does not use the OTP application environment).

The plugin should use the `wombat_plugin_services:create_capability/6` function to create a service capability. (see the `wombat_plugin_services` API section)

**Alarms capabilities**

Announcing alarms capabilities is optional. Alarms capabilities defines the assigned tags and provides additional information about the severity, the probable cause and the proposed repair action.

Alarm capabilities are matched to alarms using two identifiers, namely, using capability id and alarm id. If there is no matching alarm capability for a certain alarm, only information included by the alarm will be available and it will be tagged with the default tags. Although alarm id can be an arbitrary Erlang term, the matching algorithm works only with atoms and only those tuples whose first element is an atom. This atom is converted to an UTF-8 binary string, that is matched against the list item of the capability id.

As examples for the matching, consider the ETS limit and the missing application alarms. The alarm id of the ETS limit alarm is `ets_limit` that matches to the following capability id: `<<ets_limit>>`. Considering the missing application alarm that has a parametric id `{missing_application, App}` the correct capability id is `<<missing_application>>`.  

The `wombat_plugin_utils:create_alarm_capability/5` utility function (refer to the Useful functions in `wombat_plugin_utils` section for further detail) should be used to create an alarm capability, which should be returned by the `capabilities/1` callback or be announced using the `wombat_plugin:announce_capabilities/1` function.

**Callbacks of the `wombat_plugin` behaviour**

The following callbacks are defined in the `wombat_plugin` behaviour:

1. `callback init(Arguments :: [term()]) -> {ok, wombat_types:plugin_state()} | {skip, Msg :: binary()} | {error, _}.`

   When the plugin is started, its `init` function is called with the arguments specified for the plugin in `sys.config/wombat.config`. It either returns the initial state, which is typically a record (just like in case of a `gen_server` module) or an error to indicate an unexpected problem. Alternatively, `skip` can be returned to gracefully exit without generating a crash report on the managed node. `Msg` will be shown as a notification from the plugin on Wombat dashboard.

2. `callback capabilities(wombat_types:plugin_state()) -> {wombat_types:capabilities(), NewState :: wombat_types:plugin_state()}.`

   After the plugin is started, WombatOAM will retrieve the list of capabilities provided by this plugin by calling the `capabilities` function. Currently only metrics, alarms and services are handled as capabilities. In the future, this function might be called on other occasions too. A typical pattern is to calculate the capabilities in `init`, store it in the state record and simply read and return them in `capabilities`.

3. `callback handle_info(Message :: term(), wombat_types:plugin_state()) -> {noreply, wombat_types:plugin_state()}.`

   This function is called when the plugin process receives a message, just like in case of a `gen_server`.

4. `callback terminate(wombat_types:plugin_state()) -> any().`

   This function is called when the plugin is terminated. This can happen for a number of reasons: the plugin is disabled by the user; WombatOAM is stopped; the node is removed from WombatOAM; the connection between WombatOAM and the node is stopped; etc.

5. `callback collect_metrics(wombat_types:plugin_state()) -> {ok, [wombat_types:metric_data()], NewState :: wombat_types:plugin_state()}.`

   This function is called periodically for those plugins whose `capabilities` function reported that they have at least one metric. (Those plugins who reported that they didn't have metrics but later on realized that they do have them can use the `wombat_plugin:announce_capabilities` function.) The function should return the list of metric samples (i.e. metric values).
The order of the samples is irrelevant. This function should **not** report metrics that have not been announced beforehand via `capabilities` or `wombat_plugin:announce_capabilities`.

```
4   -callback live_metrics2comp_units([wombat_types:metric_cap_id_last()],
5       wombat_types:plugin_state(())->
6       
7       {ok, 
8       [wombat_types:live_metric_comp_unit()],
9       NewState :: wombat_types:plugin_state()} |
10      {error, term(), wombat_types:plugin_state()}.  
```

When handling live metrics, metrics are divided into "computation units" for the sake of optimization. There will be one `collect_live_metrics` call for each computation unit (i.e. the metrics in the same computation unit are calculated in one `collect_live_metrics` call).

As an example, let’s assume for example that metric `a` and metric `b` are computed by calling a costly function that computes a proplist, and `a` returns one item of the proplist while `b` returns another. Metric `c` is a different independent metric. In this case, we would put `a` and `b` into the same computation unit (let’s call it `ab_group`), which would then to be calculated in a common `collect_live_metrics` function call. `c` could be in different computation unit (let’s call it `c_group`), so it would be calculated independently.

The `live_metrics2comp_units` function gets the list of metrics that the user currently wants to monitor as live metrics, and it should return the computation units that include those metrics. The data type of the computation units is up to the plugin.

In the example above, the function would return `{ok, [ab_group, c_group], State}`.

```
3   -callback collect_live_metrics(wombat_types:live_metric_comp_unit()) ->
4       {ok, [wombat_types:live_metric_data()]} | {error, term()}.
```

For each computation unit that is returned by `live_metrics2comp_units`, a process will be started. Each process will periodically (once per second by default) call `collect_live_metrics` with one of the computation units.

In the example above, if the user wanted to monitor metrics `a`, `b` and `c`, we would have two processes, one of them calling `collect_live_metrics(ab_group)` and the other calling `collect_live_metrics(c_group)`.

When the process for a computation unit crashes, the plugin won’t be stopped, only the live collection of the metrics handled by the computation unit.

Notes:

- All the above callbacks are obligatory.
- If any of the functions throw an exception, the terminate function is called and the plugin is stopped. (Note that terminate will receive the state data that was given to the function that threw the exception; changes applied to the state but not returned by that function are lost.)

**Services callbacks**

A request for a service is fulfilled by executing a suitable implementation
of the service. A certain request is identified by ReqId, while the implementation to be executed is specified by CapabilityId. State is the current plugin state, it is shared among concurrent requests being served by a certain plugin. The execution consists of the following 3 phases.

\[
\text{init\_request} \rightarrow (\text{execute\_request})+ \rightarrow \text{cleanup\_request}.
\]

1. The process begins with asking the implementations whether they are willing to serve the request (init\_request/4). The first implementation which accepts the request will be executed.

2. Real execution takes place by calling the execute\_request/4 callback of the implementation. Data pushed back to WombatOAM is created during this phase. In case of periodic requests this callback will be called multiple times (once every period).

3. After the execution has finished, the implementation is allowed to clean up. Releasing resources, cleaning the plugin state should be the part of the cleanup\_request/3.

For each phase a callback is defined in the wombat\_plugin\_services behaviour described below. All 3 callbacks must be implemented to implement a new service.

```erlang
-spec init_request(ReqId :: binary(),
                   CapabilityId :: wombat_types:capability_id(),
                   ReqArgs :: wombat_types:request_args(),
                   State :: wombat_types:plugin_state()) ->
                   {out_of_scope, 
                    ReasonBinStr :: binary(),
                    NewState :: wombat_types:plugin_state()} |
                   {error, 
                    ReasonBinStr :: binary(),
                    NewState :: wombat_types:plugin_state()} |
                   {ok,
                    DisplayInfo :: wombat_types:display_info(),
                    ExecutionInfo :: wombat_types:execution_info(),
                    NewState :: wombat_types:plugin_state()}. 
```

This callback initializes a request (identified by ReqId) for a service (announced as CapabilityId by the plugin) based on the input arguments. The validation can have the following outcomes:

- **Serving the request is out of the plugin’s scope.** For instance, consider a special configurator plugin that changes only the configs of the MongooseM application. If it is asked to change the config of a Riak application, it is simply not capable of performing the change.

- **The plugin is capable of serving such a request (it has all necessary input) but the provided arguments are incorrect.** For instance, consider the Etop service that receives the "$ETC" binary as the value of its interval argument for which it only accepts binaries that can be converted to non-negative integers.

- **The plugin is capable and willing to serve the request (all mandatory arguments are given, have been checked and considered to be valid).** In this case it initialises the request by storing any necessary data in its state, provides information about how to display the result of the request (refer to wombat\_plugin\_services:create\_display\_info/3) and how to execute the request (refer to wombat\_plugin\_services:create\_execution\_info/3).
The goal is to really execute the request (identified by ReqId), to provide data to be streamed and then to be displayed based on the previously given DisplayInfo, and to define what will happen to the stream (should be closed or kept open to continue the execution).

It can return:

- **continue** to continue a periodic request. (Non-periodic requests cannot return this.) Can return stream data or no_data.
- **close** to indicate that the request completed. Can return stream data or no_data.
- The plugin can indicate to reply_later. This is useful to execute longer jobs in a separate worker process. In this case it can use the wombat_plugin:spawn_worker/1 function to initiate a worker and the From reference received as input argument and the wombat_plugin_services:request_reply/2 function to send stream data back to WombatOAM later.
- The plugin can indicate an error with a human readable reason to be displayed on the dashboard. In this case depending on the specified restart strategy and the number of previous retries the execution can continue or finish.

Data to be pushed to WombatOAM fall into the following 3 categories.

1. plain_value(). The simplest category. This will be displayed as is.
2. stream_data_plain_table(). List of lists built up from plain_value(). This will be rendered as a table on the dashboard.
3. stream_data_interactive_table(). List of lists built up from interactive_value(). To each value a list of actions is assigned which will be listed under the value's local menu on the dashboard. 2 general API functions and a utility function are available to construct such data, which are wombat_plugin_services:create_interactive_value/2, wombat_plugin_services:create_action/4, wombat_plugin_services:create_process_actions/1.

This callback can do any cleanup necessary after the execution of the request has finished. It will always be called, regardless of how the
execution finished (successfully completed, failed, or runtime error occurred).

Notes:

- All these callbacks are evaluated in the plugin process. That means while the callbacks are being evaluated the plugin process cannot handle other tasks (i.e.: cannot push metrics, logs, alarms).

- The execution of periodic requests can always be stopped by the users. It is stopped by finalising the request instead of scheduling its next execution. Requests being executed are not effected by stop commands, they are allowed to normally terminate. Non-periodic requests cannot be stopped by the users.

- Information provided in the capabilities is used by
  - WombatOAM to create services by aggregating the capabilities that describe different implementations of the same feature.
  - WombatOAM to categorise the services. Services will be displayed under their category group (configurator, explorer, executor) on the dashboard.

- Information provided in the display info (DisplayInfo) is used by
  - The \texttt{wombat\_plugin} behaviour to control the execution of requests.
  - The dashboard to display data to be streamed by the plugins.
  - The dashboard to decide whether users are allowed to stop requests.

**The wombat\_plugin\_services API**

The following functions in the \texttt{wombat\_plugin\_services} module can be used to implement services (for example to create structures).

```erlang
-spec create_capability(CapabilityID :: binary(),
                      Type :: wombat_types:service_capability(),
                      Description :: binary(),
                      Label :: binary(),
                      Feature :: term(),
                      Options :: [wombat_types:service_info_item()]) ->
                      wombat_types:capability().
```

Create a service capability. Properties given in \texttt{Options} override the default properties of the service. These properties together with their defaults are:

- \texttt{is\_internal} (false)
- \texttt{is\_exclusive} (false)
- \texttt{priority} (0)
- \texttt{arguments} ([])
- \texttt{options} ([])

**Note**

- The options defined by the same capability should have unique names.
- The mandatory options specified by listing their names should be defined as options.
ion_name(),
    Label :: binary(),
    Default :: binary(),
    IsEnabled :: boolean()) ->
  wombat_types:service_option().

-spec create_number_option(Name :: wombat_types:service_optio
n_name(),
    Label :: binary(),
    Default :: binary(),
    IsEnabled :: boolean()) ->
  wombat_types:service_option().

-spec create_enum_option(Name :: wombat_types:service_optio
n_name(),
    Label :: binary(),
    Default :: binary(),
    IsEnabled :: boolean(),
    OptionValues :: [binary()]) ->
  wombat_types:service_option().

These 3 functions create a scalar option. An empty binary (""")
means no default value. Note that the Default value for enums should
be the member of OptionValues (or an empty binary).

-spec create_list_option(Name :: wombat_types:service_optio
n_name(),
    Label :: binary(),
    Components :: [wombat_types:servic
e_option()] ) ->
  wombat_types:service_option().

Create a list option. The components of the list are specified as options.
For instance, consider that a list of module names should be given by
users. Then, a list option with one component, which is a string option,
would be suitable to require this input. For another example, check the
built-in configurator service allowing to change a batch of configs at
once.

-spec create_display_info(DataStructure :: value | table,
    Label :: binary(),
    Options :: [display_info_option_i
tem()]) ->
  wombat_types:display_info().

Create a display info about a service. Properties given in Options override
the default properties of display info. These properties together with
their defaults are:

- is_interactive (false)
- table_headers ([])
by the init_request/4 callback is used by the framework to know how to execute a request.

- The Period specifies how often data will be streamed. It can be once or a non negative number. once means that data will be streamed only once and users are not allowed to stop the execution of the requests. Periodic requests can always be stopped by the users. The period of executing such requests is defined by the value of this option, namely, the given value defines the number of milliseconds elapsed between two executions.

- The RetryAfter specifies how the failures should be handled. It can be never or a non negative integer. never means the evaluation of the request should be never retried, whilst the given number defines the number of milliseconds after the evaluation can be retried.

- The MaxRetries defines the maximum number of attempts to evaluate the request in a row. If the number of attempts reaches the defined maximum, the plugin process gives it up and finalises the request. If its value is 0, the plugin process will never retry the evaluation and gives up immediately after the first failure occurs.

```
-spec create_interactive_value(Data :: binary(),
                                Actions :: [wombat_types:action()]) ->
                              wombat_types:interactive_value().
```

Create an interactive value within a stream data, can be used to construct a cell in an interactive tables. Data will be displayed on the dashboard as the content of the cell. Actions specifies the content of the local menu. To construct an arbitrary action, use create_action/4. If the Data is a pid, use create_process_actions/1 utility function to define the same local menu that appears for processes in the Etop service's output.

```
-spec create_action(Label :: binary(),
                    ObjectType :: node | family,
                    FeatureName :: term(),
                    FeatureArgs :: wombat_types:request_args()) ->
                   wombat_types:action().
```

Create an arbitrary action for an interactive value. Imagine an action as a zero-arity fun expression, which will be evaluated when the user request for it. The body of the fun expression is a complete request for an other, already implemented service. The target of the request can be the node creating the action or this node's family. This is specified by ObjectType. FeatureName is the feature identifying the service, which is implemented as a capability by a plugin. (Same as the 5th argument passed to create_capability/6). FeatureArgs are the request arguments, with which the request will be initialised. Label will be shown as the link of this action in the local menu.

```
-spec create_process_actions(pid()) -> [wombat_types:action()].
```

Create a list of actions related to the given process. The list of actions can be directly used as the actions of interactive values. The actions are Terminate process, process info, process messages, process dictionary, process state, process stack trace.
This function can be used by a plugin to explicitly send stream data to WombatOAM. When the `execute_request/4` callback wants to return and send stream data only later, it can return `reply_later` and use this function later to send the stream data. The `From` parameter received in `execute_request/4` must be provided to this function. Note well that one `From` value can be used only once (i.e. it cannot be used to send back multiple stream data messages).

**The `wombat_plugin` API**

The following functions in the `wombat_plugin` module can be used by plugins.

```erlang
-spec request_reply(From :: wombat_types:from_ref(),
                     Reply :: wombat_types:async_reply()) -> ok.
```

Report a notification entry.

```erlang
-spec report_log(Severity :: wombat_types:severity(),
                 LogMessage :: wombat_types:log_message()) -> ok.
```

Raise/clear an alarm.

```erlang
-spec raise_alarm(AlarmId :: wombat_types:alarm_id(),
                  AddInfo :: wombat_types:alarm_add_info()) -> ok.
-spec clear_alarm(AlarmId :: wombat_types:alarm_id()) -> ok.
```

Push the list of capabilities to WombatOAM. It needs to be called with the list of all capabilities of the plugin (not only the new ones).

**Calling the `wombat_plugin` API from outside of the plugin process**

The `wombat_plugin` API is simple because when its functions are called, WombatOAM's plugin infrastructure knows who the caller is. But when a plugin calls these functions, WombatOAM will not know who they are; therefore calling these functions from other processes is not allowed. Instead, those processes need to obtain the counterparts of these functions in the `wombat_plugin_utils` module:

```erlang
-spec report_log_cb(Options :: plugin_options()) ->
                  fun((Severity :: wombat_types:severity(),
                        LogMsg :: wombat_types:log_message()) -> ok)
                  | undefined.
```

Report a notification entry.

The `Options` parameter that needs to be passed to these functions is the same as the `Arguments` parameter that is received by the `init` function of the module.

The following is an example that shows how this function can be used:
init(Options) ->
    LogCB = wo_plugin_utils:report_log_cb(Options),
    LogCB(<"error">, <"Test notification">),
    {ok, #state{}}.

-cleanup raise_alarm_cb(Options :: plugin_options()) ->
    fun((AlarmId :: term(), Message :: term()) -> ok) | undefined.
-cleanup clear_alarm_cb(Options :: plugin_options()) ->
    fun((AlarmId :: term()) -> ok) | undefined.

Raise/clear an alarm.

-cleanup announce_capabilities_cb(Options :: plugin_options()) ->
    fun((Capabilities :: wombat_types:capabilities()) -> ok) | undefined.

Push the list of capabilities to WombatOAM. It needs to be called with the list of all capabilities of the plugin (not only the new ones).

**Useful functions in `wombat_plugin_utils`**

- cleanup `binfmt(Fmt :: io:format(), Args :: [term()]) -> binary().` Print the given arguments into a binary.

- cleanup `spawn_worker(fun(() -> any())) -> pid().` Spawn a worker process from a plugin. The return value of the fun is ignored. The process is linked to the plugin process and has special treatment. (Never use plain `erlang:spawn_link` from a plugin process!)

- cleanup `create_metric_capability(MetricId :: wombat_types:capability_id(),
    Description :: binary(),
    Type :: wombat_types:metric_type(),
    Unit :: wombat_types:metric_unit(),
    Tags :: wombat_types:capability_tags()) ->
    wombat_types:capability().` Create a metric capability term. Note that the `create metric_capability/4` function is deprecated, kept only for backward compatibility. It uses the dev and the op tags to create the metric capability.

- cleanup `cap_id_to_cap_id_last(wombat_types:capability_id()) ->
    wombat_types:metric_cap_id_last().` Return the last element of the capability id as an atom.

- cleanup `create_alarm_capability(CapabilityId :: wombat_types:capability_id(),
    Severity :: wombat_types:alarm_severity(),
    WombatOAM Plugin Guide
141
Create an alarm capability term.

**Starting periodic jobs**

The types used are defined in `wombat_types.erl`:

```erlang
-type task_fun() :: fun(() -> ok | stop).
-type millisecs() :: non_neg_integer().
```

```erlang
-spec periodic(Period :: wombat_types:millisecs(),
                Job :: wombat_types:task_fun()) -> pid().
```

Start a periodic job from a main wombat plugin module. The process either stops when the fun doesn't return `ok` or the plugin is stopped.

```erlang
-spec stream_task_data(term()) -> ok.
```

Function to be called by the job (task) processes in order to stream results back to the WombatOAM plugin process. Streamed data format is `{'$task_data', TaskPid, Data}`.

**Using wombat_tracer as a service**

If you want to write a plugin that collects trace information, you should use the tracing service provided by the `wombat_plugin` application. The service is implemented as a server that is locally registered under the name `wombat_plugin_tracer`.

To subscribe, call `wombat_plugin_tracer:subscribe(Who, Topic, Filter)`, where:

- **Who** is the pid of the receiver,
- **Topic** is `{FlagList, MFA}`, where the variables share the types defined in the documentation of `erlang:trace_pattern`.
- **Filter** has the type `fun((TraceMsg) -> boolean() | {true, Msg})`. It needs to pre-select the trace messages that shall be delivered to the receiver. The type of `TraceMsg` is defined in the documentation of `erlang:trace`. If the filter returns true the `TraceMsg` is forwarded to the subscriber as is. In case the filter returns a custom `Msg` that will be then sent to the subscriber instead of the original `TraceMsg`.

Optionally also a **FinishFlag** can be provided when calling `wombat_plugin_tracer:subscribe(Who, Topic, Filter, FinishFlag)` which can have the following values:

- **undefined** (default): only call trace messages are sent to the tracer
- **return_trace**: apart from the call messages also a `return_from` trace message is sent upon return from the traced function.
- **exception_trace**: same as return_trace, plus; if the traced function exits due to an exception, an `exception_from` trace message is generated, whether the exception is caught or not.

The result of the call can be:
- ok, meaning the subscription was successful and the tracing is active.
- {warning, Reason}, meaning the subscription was okay but the tracing is not active.
- {error, Reason}, meaning the subscription wasn't done due to bad arguments were passed as parameters.

The tracer sends messages that have the form {wombat_plugin_tracer, Msg}, where Msg is one of the following:

- TraceMsg as defined in the documentation of erlang:trace.
- tracer_inactived, meaning no trace messages can be expected, tracing is not active.
- tracer_actived, meaning trace messages can be expected, tracing is active.

A strong recommendation is to link the receivers – the plugins – to the tracer. Hence, the plugins can restart in case the tracer restarts, simplifying the implementation of the plugins.

Notes:

- There is no need for unsubscribing from the tracer, as the plugin is monitored by the tracer.
- Tracing calls towards the functions of a module that is reloaded or loaded after the trace pattern has been enabled is supported. However, there is one exception. Trace patterns matching to any modules ("_") won't receive traces for modules that have been reloaded or loaded after the pattern has been activated. Also note that when loading a module is triggered by a first call towards that module, then this first call will not be traced.

**Example**

Assume you want to keep track which modules are loaded into the VM. Then, first subscribe to trace erlang:load_module/2 calls during init:

```erlang
init(_Args) ->
  Topic = [{[], {erlang, load_module, 2}},
            Filter = fun({trace, _, call, {erlang, load_module, [_, _]}}) -> true;
                  (_) -> false
            end,
  case wombat_plugin_tracer:subscribe(self(), Topic, Filter) of
    ok ->
      ok;
    {warning, Warning} ->
      Formatted =
        wombat_plugin_utils:binfmt("Tracers response: ~p", [Warning]),
        wombat_plugin:report_log(<"warning">>, Formatted);
    {error, Reason} ->
      Formatted =
        wombat_plugin_utils:binfmt("Tracers response: ~p", [Reason]),
        wombat_plugin:report_log(<"error">>, Formatted)
  end,
  link(whereis(wombat_plugin_tracer)).
```

To receive the collected trace messages and other system messages sent by the tracer, add the following function clause to handle_info/2.
Example of a complete plugin

```erlang
%%%======================================================
%%% @copyright 2015-2016, Erlang Solutions Ltd
%%% @doc Example WombatOAM plugin.
%%% This example plugin demonstrates how to write a simple WombatOAM plugin. It does
%%% the following:
%%% - It provides two metrics: nodes_count and hidden_nodes_count.
%%% - It raises an alarm and sends a notification if there is a process
%%%   registered with the name "Troublemaker". This is checked once a second.
%%%   (In a real plugin, this value would be much higher to avoid overloading
%%%   the system, e.g. one minute.)
%%% To activate this plugin, the following entry needs to be added to
%%% wombat.config:
%%% ```
%%% ```
%%% `%% module(wombat_plugin_example).
%%% `%% -copyright("2015-2016, Erlang Solutions Ltd.").
%%% `%% -behaviour(wombat_plugin).
%%% `%% -behaviour(wombat_plugin_services).
%%% % wombat_plugin callbacks
%%% -export([init/1, capabilities/1,
%%%         handle_info/2, terminate/1,
%%%         collect_metrics/1, live_metrics2comp_units/2, collect_live_metrics/1]).
%%% % wombat_plugin_services callbacks
%%% -export([init_request/4, execute_request/4, cleanup_request/3]).
%%% -define(CHECK_INTERVAL, 1000). % 1 second
```
-record(state,
   {metric_info_tuples = [],
    capabilities = [],
    troublemaker_exists = false,
    requests = []}
   ).

type state() :: #state{}.

type metric_internal_id() :: atom().

type metric_info_tuple() :: {MetricInternalId :: metric_internal_id(),
                           MetricNameBin :: binary(),
                           Type :: wombat_types:metric_type(),
                           Unit :: wombat_types:metric_unit(),
                           Tags :: wombat_types:capability_tags()}.

type tm_mode() :: binary().

---

%% Plugin state.

---

%%%---------------------------------------------
%% wombat_plugin callbacks
%%%---------------------------------------------

---

-spec init(Arguments :: [term()]) -> {ok, state()} | {error, _}.

init(_) ->
   Metrics = get_metric_info_tuples(),
   MetricCapabilities =
      [ wombat_plugin_utils:create_metric_capability(           
         metric_name_to_capability_id(Name), Name, Type, Unit, Tags) 
      || {Id, Name, Type, Unit, Tags} <- Metrics ],
   ServiceCapabilities = service_capabilities(),
   AlarmCapabilities = alarm_capabilities(),
   Capabilities =
      MetricCapabilities ++ ServiceCapabilities
% Note that announcing alarm capabilities is optional. ++ AlarmCapabilities,
%
% Alarms and notifications pushed to WombatOAM based on periodic checks
% The process started as periodic task will check whether a process
% registered as 'troublemaker' exists. The result of each check is
% streamed to the plugin process to perform any necessary further actions.
% wombat_plugin:periodic(
% ?CHECK_INTERVAL,
% fun() ->
%   %% Determine the current status of the troublemaker process.
%   TroubleMaker = erlang:whereis(troublemaker),
%   %% Inform the plugin process about the troublemaker process.
%   ok = wombat_plugin:stream_task_data(TroubleMaker),
% end),
%
% Perform the initial check.
% TroublemakerExists =
% case erlang:whereis(troublemaker) of
% undefined ->
%   wombat_plugin:clear_alarm(there_is_a_troublemaker),
%   false;
% Pid ->
%   wombat_plugin:raise_alarm(there_is_a_troublemaker, [{pid, Pid}]),
%   true
% end,
% {ok, #state{metric_info_tuples = Metrics,
%   capabilities = Capabilities,
%   troublemaker_exists = TroublemakerExists}}.
%
%-spec capabilities(state()) -> {wombat_types:capabilities(), state()}.
capabilities(#state{capabilities = Capabilities} = State) ->
{Capabilities, State}.
%
%-spec handle_info(Message :: term(), state()) -> {noreply, state()}.
handle_info({'$task_data', _Pid, Troublemaker},
#state{troublemaker_exists = TroublemakerExistsOld} = State) ->
NewState =
  case {TroublemakerExistsOld, Troublemaker} of
    {false, undefined} ->
      %% No troublemaker.
      State;
    {true, undefined} ->
%% The troublemaker disappeared.
wombat_plugin:clear_alarm(there_is_a_troublemaker),
State#state{troublemaker_exists = false};
{false, Pid} ->
  %% The troublemaker appeared.
wombat_plugin:raise_alarm(there_is_a_troublemaker),
{false, Pid} =>
  [{pid, Pid}]),
Msg = wombat_plugin_utils:binfmt(
  "We have a troublemaker: -p", [Pid]),
wombat_plugin:report_log(<"warning">, Msg),
State#state{troublemaker_exists = true};
{true, Pid} =>
  %% The troublemaker is still there.
Msg = wombat_plugin_utils:binfmt(
  "The troublemaker is still there: -p", [Pid]),
wombat_plugin:report_log(<"warning">, Msg),
State.
end,
{noreply, NewState};
handle_info(_Message, State) ->
{noreply, State}.

%-------------------------------------------------------
-----------------------
% @doc Terminate the plugin.
% @end
%-------------------------------------------------------
-----------------------
-spec terminate(state()) -> any().
terminate(_State) ->
  ok.

%-------------------------------------------------------
-----------------------
% @doc Return the metrics' values belonging to the already announced
% capabilities.
% @end
%-------------------------------------------------------
-----------------------
-spec collect_metrics(state()) -> {ok, [wombat_types:metric_data()], state()}.
collect_metrics(#state{metric_info_tuples = Metrics} = State) ->
  Samples = [{metric, metric_name_to_capability_id(Name), Type, get_metric_value(Id)}
           || {Id, Name, Type, _Unit, _Tags} <- Metrics],
            {ok, Samples, State}.

%-------------------------------------------------------
-----------------------
% @doc Convert live metrics into computation units.
% @end
%-------------------------------------------------------
-----------------------
-spec live_metrics2comp_units(LiveMs :: [wombat_types:metric_cap_id_last()],
state()) ->
  {ok, [metric_info_tuple()], state()} | {error, term(), state()}.
live_metrics2comp_units(LiveMs, #state{metric_info_tuples = Metrics} = State) ->
  %% Return those metric_info_tuples whose cap_id_last
  %% is present in LiveMs
  % (i.e. those metrics that shall be collected).
CompUnits = [{MetricInfoTuplue || MetricInfoTuple <- Metrics, lists:member(metric_info_tuple_to_cap_id_last(MetricInfoTuple), LiveMs)}, {ok, CompUnits, State}].

%%-------------------------------------------------------
%% Initialization of a service request.
%%-------------------------------------------------------
init_request(ReqID :: binary(), CapabilityID :: wombat_types:capability_id(), ReqArgs :: wombat_types:request_args(), State :: wombat_types:plugin_state()) ->
    ok, wombat_plugin_services:create_display_info(_DataStructure = value, _Label = <<"Troublemaker status">>, _Options = []), wombat_plugin_services:create_execution_info(_Period = once, _RetryAfter = never, _MaxRetries = 0), State;
init_request(ReqID, [<<"troublemaker watcher">>], _ReqArgs, State) ->
    ok, wombat_plugin_services:create_display_info(_DataStructure = table, _Label = <<"Troublemaker status">>, _Options = [
        {is_interactive, true},
        {table_headers, [<<"Status">>]}
    ]), wombat_plugin_services:create_execution_info(_Period = 3000, _RetryAfter = never, _MaxRetries = 0), State;
init_request(ReqID, ["troublemaker start"], ReqArgs, State) ->
    case proplists:get_value("mode", ReqArgs) of
        undefined ->
            {error, "Mandatory argument 'mode' missing.", State};
        Mode when Mode =:= "Persistent"; Mode =:= "Temporary" ->
            ok, wombat_plugin_services:create_display_info(
                _DataStructure = table,
                _Label = "Troublemaker process id",
                _Options = [
                    {is_interactive, true},
                    {table_headers, ["Result", "Pid"]}
                ],
                wombat_plugin_services:create_execution_info(
                    _Period = once,
                    _RetryAfter = never,
                    _MaxRetries = 0),
                add_req_info(ReqID, Mode, State)};
        Mode ->
            out_of_scope,
            wombat_plugin_utils:binfmt("Unknown value for argument mode: ~p", [Mode]), State
    end;
init_request(_ReqID, ["troublemaker stop"], _ReqArgs, State) ->
    ok, wombat_plugin_services:create_display_info(
        _DataStructure = value,
        _Label = "Result",
        _Options = []),
    wombat_plugin_services:create_execution_info(
        _Period = once,
        _RetryAfter = never,
        _MaxRetries = 0),
    State}.

%-------------------------------------------------------
-----------------------
% @doc Execute a service request.
% @end
%-------------------------------------------------------
-----------------------
-spec execute_request(ReqID :: binary(),
    CapabilityID :: wombat_types:capability_id(),
    From :: wombat_types:from_ref(),
    State :: wombat_types:plugin_state()) ->
    {continue | close,
     no_data | {data, StreamData :: wombat_types:stream_data()},
    NewState :: wombat_types:plugin_state()}
    | {reply_later,
     NewState :: wombat_types:plugin_state()}
    | {error,
     ReasonBinStr :: binary(),
     NewState :: wombat_types:plugin_state()},
    execute_request(_ReqId, ["troublemaker status"], From, State) ->
    Status = troublemaker_status(),
    {close, {data, Status}, State};
    execute_request(_ReqId, ["troublemaker watcher"], From, State) ->

The worker is only spawned for the sake of example

```
wombat_plugin_utils:spawn_worker(
    fun() ->
        Data = watch_troublemaker(),
        wombat_plugin_services:request_reply(From,
            {continue, {data, Data}})
    end),

    {reply_later, State};
```

execute_request(ReqId, [<<"troublemaker start">>], _From, State) ->
    Mode = get_req_info(ReqId, State),
    {Result, Pid} = start_troublemaker(Mode),
    BinPid = wombat_plugin_utils:binfmt("-p", [Pid]),
    PidActions = wombat_plugin_services:create_process_actions(Pid),
    Data =
        [{wombat_plugin_services:create_interactive_value(Result, []),},
         {wombat_plugin_services:create_interactive_value(BinPid, PidActions),}]
    {close, {data, Data}, State};

execute_request(_ReqId, [<<"troublemaker stop">>], _From, State) ->
    case stop_troublemaker() of
        error ->
            {error, <<"No Troublemaker process running">>, State};
        ok ->
            {close, {data, <<"Done">>}, State}
    end.

```
CapabilityId = <<"there_is_a_troublemaker">>,
Severity = minor,
ProbableCause = <<"A process has been registered with the name troublemaker.">>,
ProposedRepairAction = <<"Use the Stop Troublemaker service to terminate the process.">>,
Tags = [[<<"op">>]],
[wombat_plugin_utils:create_alarm_capability(
   CapabilityId, Severity, ProbableCause, ProposedRepairAction, Tags)].
wombat_types:metric_cap_id_last().
metric_info_tuple_to_cap_id_last({_Id, Name, _Type, _Unit, _Tags}) ->
  wombat_plugin_utils:cap_id_to_cap_id_last(metric_name_to_capability_id(Name)).

% Internal functions - Services

service_capabilities() ->
  [wombat_plugin_services:create_capability(
      [<!"troublemaker status"!>], % CapabilityId
      explorer, % Type
      "Return whether the Troublemaker process is alive or not.">>, % Description
      <!"Get Troublemaker status"!>, % Label
      troublemaker_status, % Feature
      []), % Options - no arguments, use defaults
      wombat_plugin_services:create_capability(
      [<!"troublemaker watcher"!>], % CapabilityId
      explorer, % Type
      "Periodically return whether the Troublemaker process is alive or not.">>, % Description
      <!"Watch Troublemaker"!>, % Label
      troublemaker watcher, % Feature
      []), % Options - no arguments, use defaults
      wombat_plugin_services:create_capability(
      [<!"troublemaker start"!>], % CapabilityId
      executor, % Type
      "Start the Troublemaker process.">>, % Description
      <!"Start Troublemaker"!>, % Label
      troublemaker_start, % Feature
      [{is_internal, false},
       {options, [wombat_plugin_services:create_enum_option(
         <!"mode"!>, % Name
         <!"Mode"!>, % Label
         <!""!>, % No Default
         true, % IsEnabled
         [<!"Persistent"!>, <!"Temporary"!>]]}
       % EnumValues
       ]},
       {arguments, [<!"mode"!>]}]
     ), % Options
  wombat_plugin_services:create_capability(
      [<!"troublemaker stop"!>], % CapabilityId
      executor, % Type
      "Stop the Troublemaker process.">>, % Description
      <!"Stop Troublemaker"!>, % Label
      troublemaker_stop, % Feature
      [] % Options
    ],
    %-------------------------------------------------------
    % @doc Start a troublemaker process if one is not started already
    %-------------------------------------------------------
    -spec start_troublemaker(tm_mode()) -> {binary(), pid()}.
    start_troublemaker(Mode) ->
      Parent = self(),
      TMPid = spawn(fun() ->
          try register(troublemaker, self()) of
          {ok, Pid} ->
            {binary(), Pid}
          end,
          Parent
        end).

true ->
    Parent ! {started, self()},
    Timeout =
    case Mode of
        <<"Persistent">> ->
          infinity;
        <<"Temporary">> ->
          10000
    end,
    receive
      stop -> ok
    after
      Timeout -> ok
    end
    catch error:badarg ->
        Parent ! {already_started, self}
    end).

receive
    {started, TMPid} ->
        {{"Started"}, TMPid};
    {already_started, TMPid} ->
        {{"Already started"}}, whereis(troublemaker)
end.

%%-------------------------------------------------------
-----------------------
%% @doc Stop the troublemaker process
%% @end
%%-------------------------------------------------------
-----------------------
-stop_troublemaker() -> ok | error.
stop_troublemaker() ->
    case whereis(troublemaker) of
        undefined ->
        error;
        Pid ->
          exit(Pid, shutdown),
          ok
    end.

%%-------------------------------------------------------
-----------------------
%% @doc Check troublemaker status and create an interactive table data
%% accordingly.
%% @end
%%-------------------------------------------------------
-----------------------
-watch_troublemaker() -> wombat_types:stream_data_interactive_table().
watch_troublemaker() ->
    case whereis(troublemaker) of
        undefined ->
            [[wombat_plugin_services:create_interactive_value
            ("Not running", [])];
            _Pid ->
                Action = wombat_plugin_services:create_action
            ("Stop Troublemaker");
            _WombatStopValue -> wombat_plugin_services:create_interactive_value
            ("Running", [Action])]]
        end.
% @doc Return troublemaker status as a binstring % @end %-------------------------------------------------------
-troublemaker_status() -> binary().
troublemaker_status() ->
    case whereis(troublemaker) of
        undefined ->
            <<"Not running">>;
        _Pid ->
            <<"Running">>
    end.

%-------------------------------------------------------
% @doc Add info about a request to the state. % @end %-------------------------------------------------------
-add_req_info(ReqId :: binary(),
    ReqInfo :: tm_mode(),
    State :: state()) -> NewState :: state().
add_req_info(ReqId, ReqInfo, #state{requests = Requests} = State) ->
    State#state{requests = [{ReqId, ReqInfo}|Requests]}.

%-------------------------------------------------------
% @doc Get the info of a request from the state. % @end %-------------------------------------------------------
-get_req_info(ReqId :: binary(),
    State :: state()) -> tm_mode().
get_req_info(ReqId, #state{requests = Requests}) ->
    {_ReqId, ReqInfo} = lists:keyfind(ReqId, 1, Requests),
    ReqInfo.

%-------------------------------------------------------
% @doc Delete the info of a request from the state. % @end %-------------------------------------------------------
-delete_req_info(ReqId :: binary(),
    State :: state()) -> NewState :: state().
delete_req_info(ReqId, #state{requests = Requests} = State) ->
    NewRequests = lists:keydelete(ReqId, 1, Requests),
    State#state{requests = NewRequests}.

Rules about passing callback functions to non-WombatOAM processes

There are two important rules to keep in mind when passing a callback function to a non-WombatOAM process:

1. Agent modules (including plugin modules and plugin infrastructure modules) should never pass a reference to an anonymous or local function (e.g. sys:install(interesting_gen_server, {fun (FuncState, SysMsg, ServerState) -> ... end, FuncState0}) or sys:install(interesting_gen_server, {fun my_dbg/3, FuncState0})) to a non-agent process, because when the agent module is purged, the process with the reference to the unloaded module will be killed by code:purge.
Agent modules should pass only exported functions using the MFA syntax (e.g. `sys:install(interesting_gen_server, {fun Module:my_dbg/3, FuncState})`), because this way the non-WombatOAM process will keep only the MFA in its memory as opposed to a reference, so it is not affected by the agent module being purged. When the callback is called, the caller will get an "undefined function" error, but that can be caught easily by the non-WombatOAM process. The plugin developer should check whether the error is indeed caught by the process that the plugin is observing.

1. The callback functions should be very quick: they should not take more than 1 second even if the system is loaded heavily. This is because if a process is executing a callback function defined in an agent module, WombatOAM will give 1 second for that function call to finish before doing a hard purge (which would kill the process if it were still executing the callback).

The following snippet demonstrates the problem behind the first rule:

```erlang
-module(test).
-compile(export_all).
f() ->
  io:format("Finished f_fun").
f_fun() ->
  fun() ->
    io:format("Finished f_fun")
  end.
```

```erlang
$ cat test.erl
-module(test).
-compile(export_all).
f() ->
  io:format("Finished f_fun").
f_fun() ->
  fun() ->
    io:format("Finished f_fun")
  end.
```

```erlang
$ erl
% We load the test module.
1> c(test).
{ok,test}
% We create a reference to a function in the test module.
2> F = test:f_fun().
#Fun<test.0.124694843>
% We don't have old code yet (only new code), so check_process_code is false when called with the shell process.
3> erlang:check_process_code(self(), test).
false
% We mark the test module as old code.
4> code:delete(test).
true
% Now check_process_code says that we do have old code.
5> erlang:check_process_code(self(), test).
true
% Code purge calls check_process_code on each process to decide if it uses the old version of the purged module, and if so, it kills the % process. In this case it kills the shell process.
6> code:purge(test).
*** ERROR: Shell process terminated! ***
Eshell V5.10.4 (abort with ^G)
```

If line 2 is replaced with `F = fun test:f_fun/0`, then this problem will not occur:
% Now F only contains the information that is should call
test:f_fun/0, and not a real reference that points inside
code of the test module.
2> F = fun test:f/0.
#Fun<test.f.0>
7

% Therefore it doesn't use the test module...
5> erlang:check_process_code(self(), test).
false

% ...and therefore it is not killed by purge.
6> code:purge(test).
false

% If we now call F(), we will simply get an undef error that can be
% caught by 'catch'. Before doing that, let's set the path to an
% empty list, otherwise Erlang would automatically load test.beam
% when we call F.
7> code:set_path([]).
true

8> F().
** exception error: undefined function test:f/0
9>

9> catch F().
{'EXIT',{undef,[[test,f,[],[]],
{erl_eval,do_apply,5,[[file,"erl_eval.erl"
},{line,560}],
{erl_eval,expr,5,[[file,"erl_eval.erl"],[line,357]]},
{shell,exprs,7,[[file,"shell.erl"],{line,674}]},
{shell,eval_exprs,7,[[file,"shell.erl"],[line,629]]},
{shell,eval_loop,3,[[file,"shell.erl"],[line,614]]]}}}

A typical scenario is to pass an MFA (which points to a WombatOAM plugin) to a non-WombatOAM process that will use it as a callback. Examples include:

- Passing debug functions to gen processes using sys:install/sys:remove. This scenario is analysed below.
- Passing callback functions to event handler processes.

Let's say the plugin uses sys:install to install a debug function into the interesting_gen_server process. When WombatOAM wants to stop the plugin, the plugin shall call sys:remove, with 0 timeout:

terminate(_State) ->
...
catch sys:remove(interesting_gen_server, fun ?MODULE:my_dbg_function/3, 0),
...
alarm plugin

Description
The alarm plugin is intended to do some basic system status checks on the monitored system and report alarms to WombatOAM when any of the monitored parameters reach a certain threshold.

Applications it depends on
kernel

Modules
wombat_plugin_alarm

Reports
The plugin reports the following alarms:
- process_limit
- port_limit
- ets_limit
- atom_limit
- module_limit
- export_limit
- memory_limit
- open_file_limit
- open_socket_limit
- os_cpu_load
- disk_capacity
- shell_history_size
- process_message_queue
- system_information
- old_code

Configuration options
The interval at which the checks are performed is configurable, in case it is necessary to regulate plugin's moderate resource use:
- collection_interval (integer, default: 60000): Specifies how many milliseconds to wait between checking whether any process-related alarm (e.g. process_message_queue) should be raised or ceased.
- interval (integer, default: 60000): Specifies how many milliseconds to wait between checking whether any system limit-related alarm (e.g. atom_limit) should be raised or ceased.
- app_check_interval (integer, default: 60000): Specifies how many milliseconds to wait between checking whether the version of any application changed. In case of any change, WombatOAM will raise a different_application_versions alarm.

The "node info alarms" are raised by the WombatOAM server, based on the node info reported by this plugin:
- node_info_opts/app_version_alarms (default: true): If true, then alarm will be raised if nodes in the same family have different versions of the same application, or the application is not running on all nodes. Application started or stopped on nodes will be logged as notifications.
- node_info_opts/time_diff_alarms (default: true): If true, then
alarms will be raised if nodes in the same family are in different
time zones.

The `system_checks` option is a list of system checks that the plugin shall
perform.

- `process_limit`, `port_limit`, `ets_limit`, `atom_limit`, `module_limit`,
  `export_limit`, `memory_limit`, `open_file_limit`, `open_socket_limit`,
  `os_cpu_load`, `disk_capacity`: These system checks are a minor
  alarm limit and a major alarm limit. After these limits are reached,
an appropriate alarm is raised. The thresholds are expressed as
  percentages.
- `shell_history_size`, `process_message_queue`: These system checks
  are a minor alarm limit and a major alarm limit. After these limits
  are reached, an appropriate alarm is raised. The thresholds are
  absolute numbers.
- `system_information`, `old_code`: These system checks only have an
  alarm severity, which specifies the severity of the alarm that
  should be raised for them.

Example wombat.config entries

```
{set, wo_plugins, plugins, alarm, collection_interval, 60000}.
{set, wo_plugins, plugins, alarm, interval, 600000}.
{set, wo_plugins, plugins, alarm, app_check_interval, 60000}.
{set, wo_plugins, plugins, alarm, node_info_opts, app_version_alarms, true}.
{set, wo_plugins, plugins, alarm, node_info_opts, module_version_alarms, true}.
{set, wo_plugins, plugins, alarm, node_info_opts, time_diff_alarms, true}.
```

The system check configuration entries can be overridden individually:

```
{set, wo_plugins, plugins, alarm, system_checks, process_limit, minor, 80}.
{set, wo_plugins, plugins, alarm, system_checks, process_limit, major, 90}.
{set, wo_plugins, plugins, alarm, system_checks, port_limit, minor, 80}.
{set, wo_plugins, plugins, alarm, system_checks, port_limit, major, 90}.
{set, wo_plugins, plugins, alarm, system_checks, ets_limit, minor, 80}.
{set, wo_plugins, plugins, alarm, system_checks, ets_limit, major, 90}.
{set, wo_plugins, plugins, alarm, system_checks, atom_limit, minor, 80}.
{set, wo_plugins, plugins, alarm, system_checks, atom_limit, major, 90}.
{set, wo_plugins, plugins, alarm, system_checks, module_limit, minor, 80}.
{set, wo_plugins, plugins, alarm, system_checks, module_limit, major, 90}.
{set, wo_plugins, plugins, alarm, system_checks, export_limit, minor, 80}.
{set, wo_plugins, plugins, alarm, system_checks, export_limit, major, 90}.
{set, wo_plugins, plugins, alarm, system_checks,}
```

To disable all system checks or enable only a few of them, list only those that shall be performed:

```erlang
{% Enable only three system checks
{set, wo_plugins, plugins, alarm, system_checks, [{process_limit, [{minor, 80}, {major, 90}]}],
{port_limit, [{minor, 80}, {major, 90}]}],
{ets_limit, [{minor, 80}, {major, 90}]}].

{% Disable all system checks
{set, wo_plugins, plugins, alarm, system_checks, []}.  
```
application plugin

Description

The application plugin reports information about the OTP applications present on the managed node that the user can see on the Topology page. It reports the following information for each loaded application:

- **Name**: the name of the application.
- **Description**: the short description of the application.
- **Version**: the version of the application.
- **Running**:
  - The PID of the process started by the application callback if the application has a callback module and is running.
  - `library` if the application has no callback, but has been started and is running.
  - `Loaded, but not started` if the application has not been started yet.
  - `crashed` if the application has been started, but is not running currently. This usually indicates an error on the managed system.
  - `unknown` if the application has been not started, has no callback module, but is running. This indicates an error on the managed system.
- **Type**: The type of the application (permanent, transient, temporary or unknown).
  - If a permanent application terminates, all other applications and the entire Erlang node are also terminated.
  - If a transient application terminates with `Reason == normal`, this is reported but no other applications are terminated.
  - If a transient application terminates abnormally, all other applications and the entire Erlang node are also terminated.
  - If a temporary application terminates, this is reported but no other applications are terminated.
  - If the application has not been started yet, its type is `unknown`.

It is essentially the second parameter of `application:start/2`. If the application is running, but has been not started, **Error: application running, but not started!** is shown. This indicates an error on the managed system. *Configuration: the list of the environment variables of the application together with their current values.

Applications it depends on

kernel

Modules

wombat_plugin_application

Reports

The plugin doesn't report any metrics or alarms. It sends a log message when an error is detected in the application status (started, but not running; running, but not started). It is used for displaying information about the OTP applications present on the managed node.

Note: If this plugin is turned off, any values that change will not be updated on the dashboard.

Configuration options
app_info_period

The period (in ms.) used to report a message with the last status of the applications and their environment variables.

Example wombat.config entry

```
{set, wo_plugins, plugins, application, app_info_period, 20000},
```

builtin_metrics plugin

Description

The `builtin_metrics` plugin is responsible for providing the general available metrics for a managed node. These metrics are described in the section "Metrics".

Applications it depends on

kernel

Modules

- `wombat_plugin_builtin_metrics`
- `wombat_plugin_builtin_metrics_lib`

Reports

For descriptions of the metrics that are generated, see the section "Metrics".

Configuration options

`port_type_counters_mode` Specifies which metrics should be reported based on the types of Erlang ports. If `simple`, then only the following three metrics will be reported in the IO category:

- "Number of open SCTP sockets"
- "Number of open TCP sockets"
- "Number of open UDP sockets"

If `detailed`, then additionally each port type X will have its own metric called "Number of open X ports" in the "Node / Port info" category.

The default value is `simple`.

Example wombat.config entry

```
{set, wo_plugins, plugins, builtin_metrics, port_type_counters_mode, detailed}.
```
code_tracer plugin

**Note:** This plugin traces processes on the managed node, so it is advisable to turn it off if you want to start tracing yourself. Refer to the Erlang/OTP documentation on the `trace` function.

Applications it depends on

kernel

**Modules**

wombat_plugin_code_tracer

**Reports**

The following notifications are always enabled:

- Code loaded
- Code purged
- Shell command

The following notifications are disabled by default:

- Busy port
- Busy dist port
- Long GC
- Long schedule
- Large heap

Triggering limits for these notifications are configurable (see example section below).

The notifications are documented in the Notifications page.

**Metrics**

- Busy port Tags: dev, op
- Busy dist port Tags: dev, op
- Long GC Tags: dev
- Long schedule Tags: dev
- Large heap Tags: dev

The metrics are documented in the Metrics page.

**Configuration options**

- `report_system_monitor_notifs`(true or false, default: false): Whether the plugin should generate "Busy port", "Busy dist port", "Long GC", "Long schedule" and "Large heap" notifications (based on `erlang:system_monitor`). If disabled, none of these notifications are generated.

- `report_system_monitor_metrics`(true or false, default: true): Whether the plugin should collect "Process notifications" metrics (i.e. "Busy port", "Busy dist port", "Long GC", "Long schedule" and "Large heap").

- `code_clash_ignore_paths`([string()], default: ["patches", "consolidated"): Only a notification (and no module_clash alarm)
will be raised for clash paths that match any of the values. Each value can be the entire path of the module directory or part of it. For example the value "/db_client/patches" would match any directories that have this inside of them e.g.
/opt/my_app/db_client/patches,
/usr/local/opt/my_app/db_client/patches,
/db_client/patches/ebin/ etc.

- old_code_ignore_pattern (string): No old_code alarm will be raised for module names which match the given regular expression. By default WombatOAM ignores
  - Elixir's built-in protocol modules (i.e. the default pattern is: "Collectable|Enumerable|IEx.Info|Inspect|List.Chars|String.Chars" );
  - MongooseIM specific modules that are compiled and reloaded at runtime (i.e. the default pattern is: "^mod.*_backend|^mod_bosh_dynamic|^mod_mam_odbc_arch_params" ).

See more configuration options in the example section below.

**Example wombat.config entries**

```elixir
{set, wo_plugins, plugins, code_tracer, trace_modules, true}.
{set, wo_plugins, plugins, code_tracer, long_gc_limit, 500}.
{set, wo_plugins, plugins, code_tracer, long_schedule_limit, 100}.
{set, wo_plugins, plugins, code_tracer, large_heap_limit, 80000}.
{set, wo_plugins, plugins, code_tracer, long_gc_scheduler_limit, 4}.
{set, wo_plugins, plugins, code_tracer, report_system_monitor_notifs, false}.
{set, wo_plugins, plugins, code_tracer, report_system_monitor_metrics, true}.
{set, wo_plugins, plugins, code_tracer, code_clash_ignore_paths, ["/the/entire/path", "/start/of/path", "end/of/path", "single_dir"]}.
{set, wo_plugins, plugins, code_tracer, old_code_ignore_pattern, "^Elixir\..*"}.
```
**cowboy plugin**

**Description**

The *cowboy* plugin reports metrics from requests performed to the endpoints defined in the *cowboy* application. When the plugin starts, it processes all registered routes and creates a metrics combining each route with every single HTTP method (i.e. GET, POST, PUT). Additionally metrics for each type of response status code is also created (i.e. 100, 200, 300, 400 and 500).

**Applications it depends on**

cowboy

The plugin only works with version 1.x.

**Modules**

wombat_plugin_cowboy, wombat_plugin_cowboy_common

**Reports**

**Metrics reported**

**Cowboy Request Count** category: these metrics show the number of requests performed with a certain HTTP code or with a certain method and endpoint.

- HTTP 1xx Tags: dev
- HTTP 2xx Tags: dev, op
- HTTP 3xx Tags: dev
- HTTP 4xx Tags: dev, op
- HTTP 5xx Tags: dev, op
- [HTTP-METHOD] /ROUTE Tags: dev

Where HTTP-METHOD is one of GET, POST, PUT; and ROUTE is one of the routes defined through Cowboy.

**Cowboy Response Time [ms]** category: these metrics show the average response time performed with a certain method and endpoint.

- Avg. [HTTP-METHOD] /ROUTE Tags: dev

Where HTTP-METHOD is one of GET, POST, PUT; and ROUTE is one of the routes defined through Cowboy.
phoenix plugin

Description

The phoenix plugin reports metrics from requests performed to the endpoints defined in the phoenix application. The plugin exposes fine-grain metrics to give insight into your Phoenix application. The plugin captures the number of requests and the average response time for each registered endpoint, HTTP method (e.g. HTTP GET) and HTTP status code (e.g. HTTP 200).

Applications it depends on

phoenix and cowboy

The plugin only works with Phoenix versions 1.x that use Cowboy 0.x or 1.x versions.

Modules

wombat_plugin_phoenix

Reports

Metrics reported

Phoenix: Miscellaneous category

- Connected channels Number of connected PubSub channels Tags: dev, op.
- Open sockets Number of open sockets Tags: dev, op.

Phoenix: Request Count category: these metrics show the number of all requests and requests performed with a certain HTTP result code or with a certain method and endpoint.

- All requests Tags: dev, op.
- HTTP STATUS-CODE Tags: dev, op.
- [HTTP-METHOD] /ENDPOINT Tags: dev.
- [HTTP-METHOD] Tags: dev.

Phoenix: Response Time [ms] category: these metrics show the average response time of all requests and requests performed with a certain HTTP result code or with a certain method and endpoint.

- Average duration of all requests Tags: dev, op.
- Avg. STATUS-CODE Tags: dev, op.

Where HTTP-METHOD is one of GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH; ENDPOINT is one of the endpoints defined throughPhoenix; and STATUS-CODE is one of the official status codes (100, 101, 102, 200, 201, 202, 203, 204, 205, 206, 207, 208, 226, 300, 301, 302, 303, 304, 305, 306, 307, 308, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 421, 422, 423, 424, 426, 429, 431, 451, 500, 501, 502, 503, 504, 505, 506, 507, 508, 510, 511); one of RESTful error responses (103, 420, 450, 498, 499, 500, 509, 530); one of Internet Information Services responses (440, 449, 451); one of NGINX responses (444, 495, 496, 497, 499) or one of CloudFlare responses (520, 521, 522, 523, 524, 525, 526).
phoenix telemetry plugin

Description

The `phoenix_telemetry` plugin reports metrics from requests performed to the endpoints defined in the Phoenix application. The plugin exposes fine-grain metrics to give insight into your Phoenix application. The plugin captures the number of requests and the average response time for each registered endpoint, HTTP method (e.g. HTTP GET) and HTTP status code (e.g. HTTP 200).

Applications it depends on

phoenix and telemetry

The plugin only works with Phoenix versions 1.4.7 and beyond.

Modules

`wombat_plugin_phoenix_telemetry`

Reports

Metrics reported

**Phoenix: Request Count** category: these metrics show the number of all requests and requests performed with a certain HTTP result code or with a certain method and endpoint.

- All requests Tags: dev, op.
- HTTP STATUS-CODE Tags: dev, op.
- `[HTTP-METHOD] /ENDPOINT` Tags: dev.
- `[HTTP-METHOD]` Tags: dev.

**Phoenix: Response Time [ms]** category: these metrics show the average response time of all requests and requests performed with a certain HTTP result code or with a certain method and endpoint.

- Average duration of all requests Tags: dev, op.
- Avg. STATUS-CODE Tags: dev, op.

Where `HTTP-METHOD` is one of GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH; `ENDPOINT` is one of the endpoints defined through Phoenix; and `STATUS-CODE` is one of the official status codes (100, 101, 102, 200, 201, 202, 203, 204, 205, 206, 207, 208, 226, 300, 301, 302, 303, 304, 305, 306, 307, 308, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 421, 422, 423, 424, 426, 429, 431, 451, 500, 501, 502, 503, 504, 505, 506, 507, 508, 510, 511); one of RESTful error responses (103, 420, 450, 498, 499, 509, 530); one of Internet Information Services responses (440, 449, 451); one of NGINX responses (444, 495, 496, 497, 499) or one of CloudFlare responses (520, 521, 522, 523, 524, 525, 526).
trails plugin

Description

The trails plugin reports metrics from requests performed to the endpoints defined in the trails application. When the plugin starts, it processes all registered routes and creates a metric combining each route with every single HTTP method (i.e. GET, POST, PUT). Additionally metrics for each type of response status code is also created (i.e. 100, 200, 300, 400 and 500).

Applications it depends on

trails

The plugin requires version 0.2 or newer. Also trails depends on cowboy and only works with cowboy version 1.x.

Modules

wombat_plugin_trails, wombat_plugin_cowboy_common

Reports

Metrics reported

Cowboy-trails Request Count category: these metrics show the number of requests performed with a certain HTTP code or with a certain method and endpoint.

- **HTTP 1xx Tags**: dev
- **HTTP 2xx Tags**: dev, op
- **HTTP 3xx Tags**: dev
- **HTTP 4xx Tags**: dev, op
- **HTTP 5xx Tags**: dev, op
- **[HTTP-METHOD] /ROUTE Tags**: dev

Where HTTP-METHOD is one of GET, POST, PUT or DELETE; and ROUTE is one of the routes defined through Cowboy Trails.

Cowboy-trails Response Time [ms] category: these metrics show the average response time performed with a certain method and endpoint.

- **Avg. [HTTP-METHOD] /ROUTE Tags**: dev

Where HTTP-METHOD is one of GET, POST, PUT or DELETE; and ROUTE is one of the routes defined through Cowboy Trails.
**Ejabberd plugin**

**Description**

This plugin reports metrics of an Ejabberd node. It can be used to monitor a server. It does not report any alarms.

See the XMPP specification for more information about the XMPP protocol, specifically about the different stanza-types (*iq, presence, message*).

**Applications it depends on**

ejabberd

All metrics are compatible with Ejabberd versions 17.03 and newer. The backwards compatibility of the `registered_users` metric is heavily dependent on the auth backend used by the Ejabberd instance. Support for different databases varies between versions, consult the Ejabberd documentation or source code to ensure compatibility.

**Modules**

`wombat_plugin_ejabberd`.

**Reports**

This plugin only reports metrics.

**Metrics reported**

One Ejabberd instance can serve multiple XMPP domains. This plugin collects metrics for one such instance.

**Metrics related to user activity**

The following metrics relate to Client-To-Server (C2S) activity. They are reported for each collection interval and do not accumulate, i.e., they start at zero in the beginning of the interval.

- `user_iqs_sent` Tags: dev, op
- `user_iqs_received` Tags: dev, op
- `user_messages_sent` Tags: dev, op
- `user_messages_received` Tags: dev, op
- `user_presences_sent` Tags: dev, op
- `user_presences_received` Tags: dev, op
- `offline_messages_sent` Tags: dev, op

**Metrics related to S2S activity**

The following metrics relate to Server-To-Server (S2S) activity. They are reported for each collection interval and do not accumulate, i.e., they start at zero in the beginning of the interval.

- `s2s_iqs_sent` Tags: dev, op
- `s2s_iqs_received` Tags: dev, op
- `s2s_messages_sent` Tags: dev, op
- `s2s_messages_received` Tags: dev, op
- `s2s_presences_sent` Tags: dev, op
• s2s_presences_received Tags: dev, op

**Other metrics**

The following metrics accumulate, that is, they are not reset to zero in the beginning of the collection interval.

• online_users Tags: dev, op
• registered_users Tags: dev, op
• unknown_packets Tags: dev, op

  ◦ **Note:** This metric is used to detect misuse of the XMPP protocol, and should not be triggered. The different types of packets should all be caught by the user_ and s2s_ metrics.

**Notifications**

The plugin will send a notification to WombatOAM when something unexpected occurs, for example if a message is sent to the Ejabberd plugin main process which it does not recognise.

**Configuration options**

None.
Elixir Logger plugin

Description
The plugin subscribes to the Logger application running on the Elixir nodes that are managed by WombatOAM. The plugin is capable of reporting all the logs as notifications and the number of logs as metrics to WombatOAM. The different severity levels of logs generated by Logger are debug, info, warn and error (in ascending order of precedence). The default severity level is set to warn. So the plugin reports only the logs and metrics for the warn and error severity levels. The plugin also ignores Logger events originally form the error_logger (as those are handled by another plugin).

Application it depends on
elixir

Modules
- wombat_plugin_elixir_logger
- wombat_plugin_elixir_logger_handler

Reports
The plugin reports logs/notifications and the following metrics to WombatOAM.

- Logger debug events Tags: dev
- Logger info events Tags: dev
- Logger warn events Tags: dev, op
- Logger error events Tags: dev, op

These metrics show the number of log messages captured by the plugin on different severity levels, at any given point in time. As the default severity level is set to warn, the metrics for info and debug will be shown as 0.

Configuration options
- report_metrics (boolean, default: true): Specifies whether the WombatOAM server should receive the values for the above mentioned metrics.

Example wombat.config entry
```
{set, wo_plugins, plugins, elixir_logger, report_metrics, true}.
```
**error_logger plugin**

**Description**

The error_logger plugin subscribes to the error_logger event manager and reports the log events and the number of log events as metrics to WombatOAM. It does not deal with the events on error_logger that are reported to the SASL handler.

The metrics correspond to the number of events per severity level as generated by the error_logger. The various severity levels being info, error and warning. The plugin allows the user to enable or disable collecting notifications and metrics. The plugin will only report notifications and metrics that have a severity level either equal to or higher than the severity level chosen by the user.

**Applications it depends on**

kernel

**Modules**

- wombat_plugin_error_logger
- wombat_plugin_elogger_handler

**Reports**

The plugin reports logs/notifications and the following metrics to WombatOAM.

- SASL error events Tags: dev, op
- SASL warning events Tags: dev, op
- SASL information events Tags: dev

These metrics show the number of log entries captured by WombatOAM on different log levels. Note that WombatOAM collects log entries on warning level and above by default, so the value of the "SASL information events" metric will be zero. (Unless this default log level is set to a lower value.)

**Configuration options**

The decision to receive notifications or metrics or both is configurable.

- report_notifications (boolean, default: true): Specifies whether the WombatOAM server should receive the error_logger notifications.
- report_metrics (boolean, default: true): Specifies whether the WombatOAM server should receive the values for the above mentioned metrics.

**Example wombat.config entries**

```erlang
{set, wo_plugins, plugins, error_logger, report_notifications, true}.
{set, wo_plugins, plugins, error_logger, report_metrics, true}.
```
**exometer plugin**

**Description**
The `exometer` plugin will retrieve any metrics stored in the exometer application running on the managed node. It will periodically check for new metrics, therefore metrics dynamically added to exometer will also appear on the WombatOAM dashboard.

**Applications it depends on**
exometer OR exometer_core

**Modules**
wombat_plugin_exometer

**Reports**
The plugin reports all exometer metrics with the tag `dev`.

**Configuration options**
None.

**folsom plugin**

**Description**
This `folsom` plugin will retrieve any metrics stored in the folsom application running on the managed node. It will periodically check for new metrics, therefore metrics dynamically added to folsom will also appear on the WombatOAM dashboard.

**Applications it depends on**
folsom

**Reports**
The plugin reports all folsom metrics with the tag `dev`.

**Modules**
wombat_plugin_folsom

**Configuration options**
None.
lager plugin

Description
The lager plugin subscribes to the lager application running on the managed node and is capable of reporting all the notifications/logs generated by it to WombatOAM. By default, it starts itself reporting at the warning level; this can be changed using the options shown on the WombatOAM dashboard.

Applications it depends on
lager

Modules
wombat_plugin_lager and wombat_plugin_lager_handler

Reports
The plugin reports logs/notifications and the following metrics to WombatOAM in the "Lager notifications" category:

- Lager debug events Tags: dev
- Lager info events Tags: dev
- Lager notice events Tags: dev
- Lager warning events Tags: dev, op
- Lager error events Tags: dev, op
- Lager critical events Tags: dev, op
- Lager alert events Tags: dev, op
- Lager emergency events Tags: dev, op

These metrics show the number of log entries captured by WombatOAM on different log levels. Note that WombatOAM collects log entries on warning level and above by default, so the value of the debug, info and notice metrics will be zero. (Unless this default log level is set to a lower value.)

Configuration options
- report_metrics (boolean, default: true): Specifies whether the WombatOAM server should receive the values for the above mentioned metrics.

Example wombat.config entry

```plaintext
{set, wo_plugins, plugins, lager, report_metrics, true}.
```
mnesia plugin

Description
This plugin will monitor and extract information from the Mnesia database running on a particular node. It can generate various metrics, notifications for relevant changes, and raise alarms when Mnesia enters a problematic state.

Consult the official Mnesia documentation for a deeper understanding of various topics mentioned here.

Applications it depends on
mnesia

Modules
wombat_plugin_mnesia and wombat_plugin_mnesia_metrics

Reports
This plugin reports logs, metrics and alarms.

Alarms reported
- Mnesia has become inconsistent
- Mnesia is overloaded

Please find the detailed alarm description in the Alarms documentation.

Metrics reported

Metrics from mnesia:system_info/1
- checkpoints Tags: dev
- db_nodes Tags: dev
- dump_log_time_threshold Tags: dev, op
- dump_log_write_threshold Tags: dev, op
- extra_db_nodes Tags: dev
- held_locks Tags: dev
- local_tables Tags: dev
- lock_queue Tags: dev
- master_node_tables Tags: dev, op
- running_db_nodes Tags: dev, op
- subscribers Tags: dev
- tables Tags: dev
- transaction_commits Tags: dev
- transaction_failures Tags: dev, op
- transaction_log_writes Tags: dev
- transaction_restarts Tags: dev
- transactions Tags: dev
- no_table_loaders Tags: dev
Table-specific Metrics

For tables listed under the table_metrics configuration parameter the following values will be collected:

- **TABLE-size**: the number of records in the table
- **TABLE-memory**: memory use of the table

Notifications

WombatOAM always subscribes to and displays Mnesia system events. Activity and table-specific events can also be subscribed to by editing the configuration.

Configuration options

table_events Whether to generate events for the tables stored, it can be all or none or a list of tables. If a listed table doesn't exist when WombatOAM starts monitoring the node a warning will be generated.

table_metrics Whether to generate metrics for the tables stored, it can be all or none or a list of tables. If a listed table doesn't exist when WombatOAM starts monitoring the node a warning will be generated.

activity_events Whether to subscribe to activity events or not. It can be true or false. If true, this will report a notification to WombatOAM every time an activity is reported on the Mnesia app on the node.

Note that WombatOAM periodically checks for new tables until all the tables listed as either table_events or table_metrics exist. In case of all is set the periodic checks never end, whilst in case of none no check will be performed.

Collecting table metrics for many tables or enabling event reporting can induce serious load both for WombatOAM and the managed node. Those features are mostly meant for debugging a Mnesia cluster, not for simple monitoring; use them with care.

Example sys.config entry

```erlang
{set, wo_plugins, plugins, mnesia, activity_events, false}.
{set, wo_plugins, plugins, mnesia, table_events, none}.
{set, wo_plugins, plugins, mnesia, table_metrics, none}.
```
node_info plugin

Description

This plugin generates the information about added nodes displayed under the Topology page on the dashboard. This information is a list of mostly static parameters that relate to the characteristics of the node. These values are checked and reported to WombatOAM on a very low frequency once the node has started.

Applications it depends on

kernel

Modules

wombat_plugin_node_info.erl

Reports

The plugin doesn't report any metrics, alarms, or logs. It is used for displaying information about the nodes managed in WombatOAM.

Note: If this plugin is turned off, any values that change will not be updated on the dashboard.

Configuration options

node_info_period

The period used to report the list of values to the dashboard in milliseconds. Due to the values being mostly static, this should be a rather big number to avoid unnecessary calls on the managed node.

Example wombat.config entry

```
{set, wo_plugins, plugins, node_info, node_info_period, 600000}.
```
rabbitmq plugin

Description

The rabbitmq plugin exposes metrics describing the channels, the queues, the messages, the connections, the consumers, the exchanges and the activities related to the users on the monitored RabbitMQ node. There are four main groups.

- Overview metrics denotes the total number of objects that are currently active + the total number objects that were created since the last time, the metric was read + total number of active objects from the other RabbitMQ nodes in a cluster. By objects, we mean the AMQP connections, channels, consumers, queues and exchanges.

- Simple metrics that are cumulative metrics. They will be continuously increasing i.e. cumulative. If the plugin is restarted, all the metric values will be reset.

- Message rate metrics denotes the total number of messages handled by the RabbitMQ node per second. Some of the message types are publish, confirm, redeliver etc.

- Statistic metrics that characterise the current status of the RabbitMQ node from different point of views. The values are total sums derived by counting all instances of a certain Rabbit object, for instance, the 'Queue messages' metric shows the total count of messages in all queues. These metrics are cumulative.

Supported versions.

Official stable Rabbit releases newer than 2.3.0 are supported.

Applications it depends on

rabbit

Modules

- wombat_plugin_rabbitmq
- wombat_plugin_rabbitmq_handler
- wombat_plugin_rabbitmq_metrics

Configuration options

- priority: the priority of the RabbitMQ process. The following values can be used: low, normal, high, max. Default value: low.

- msgq_check_period: The plugin implements an overload protection against dealing with too many notifications arriving from the rabbit_event handler. This parameter specifies how frequently these overload conditions should be checked. Measured in milliseconds. Default value: 5000.

- msgq_normal: If the message queue length of the plugin process goes down below this level, all rabbit events will be enabled. Default value: 500.

- msgq_stats: If the message queue length of the plugin process reaches this limit, statistics events will be disabled. Default value: 2000

- msgq_threshold: When message queue length reaches this level all kinds of rabbit events will be disabled. Default value: 5000.

- check_overload_after: Overload conditions are checked also after having these many messages. Default value: 2000.
• **handle_not_existing_stats**: Denotes whether the plugin should handle the statistics events for those RabbitMQ objects that were created before WombatOAM started monitoring the node. Default value: true.

• **live_metrics_read_deadline**: Deadline (in milliseconds) before which the live value for any given metric should have been read. Failure to meet this deadline, will lead to the deletion of the corresponding count only for the closed channels. Default value: 30000.

• **rabbitmq_statistics**: Denotes whether the plugin should handle the statistics events. Default value: true.

• **rabbit_event_mod**: The RabbitMQ module that generates the events.

• **clean_stats_period**: The plugin implements a clean-up mechanism where the metric values of dead objects are removed. This parameter specifies on how frequently this clean-up mechanism should be triggerred. Measured in milliseconds. Default value: 5000.

• **clean_stats_after**: Statistics metrics values that are older than this time will be removed. Measured in milliseconds. Default value: 30000.

**Overload protection related options**

The plugin implements an overload protection mechanism which ensures the plugin won't add a huge amount of extra work, and it will be turned automatically on/off when needed.

The following relations must be true: `msgq_normal < msgq_threshold`

• **msgq_normal**: If the message queue length of the plugin goes down below that level all rabbit events will be enabled. Default value: 500.

• **msgq_threshold**: When message queue length reaches this level all kind of rabbit events will be disabled. Default value: 5000.

**Example wombat.config entry**

```erlang
%% Set the process priority to normal
{set, wo_plugins, plugins, rabbitmq, priority, normal}.

%% Set how frequently the overload conditions are checked to one minute
{set, wo_plugins, plugins, rabbitmq, msgq_check_period, 60000}.

%% Change below which level all events will be enabled
{set, wo_plugins, plugins, rabbitmq, msgq_normal, 1000}.

%% Above this level all kind of rabbit events are disabled
{set, wo_plugins, plugins, rabbitmq, msgq_threshold, 7000}.

%% After these many messages overload conditions are checked
{set, wo_plugins, plugins, rabbitmq, check_overload_after, 5000}.

%% Process priority of the rabbit plugin
{set, wo_plugins, plugins, rabbitmq, priority, normal}.
```

**Metrics reported**

If the plugin is restarted, all these metrics are reset.
Overview metrics:

- Total active connections: Tags: dev

  The total number of open TCP connections from producers and consumers connecting to the monitored RabbitMQ node + the total number of TCP connections that the monitored RabbitMQ node had opened from producers and consumers, since the last time this metric was read by the WombatOAM server.

  In a clustered environment, the total number TCP connections from consumers connected to any of the other RabbitMQ nodes in the cluster, are also included in the above total.

- Total active channels: Tags: dev

  The total number of open AMQP channels from producers and consumers connecting to the monitored RabbitMQ node + the total number of AMQP channels that the monitored RabbitMQ node had opened from producers and consumers, since the last time this metric was read by the WombatOAM server.

  In a clustered environment, the total number of AMQP channels from consumers connected to any of the other RabbitMQ nodes in the cluster, are also included in the above total.

- Total active consumers: Tags: dev

  The total number of consumers that are currently subscribed to queues on the RabbitMQ node + the total number of consumers who had subscribed to the RabbitMQ node since the last time the metric was read by the WombatOAM server.

  In a clustered environment, the total number of consumers that are subscribed with the other RabbitMQ nodes in the cluster, are also included in the above total.

- Total active exchanges: Tags: dev

  The total number of exchanges that have been declared and currently active in the RabbitMQ node + the total number of exchanges that were declared in the RabbitMQ node since the last time the metric was read by the WombatOAM server + the default number of exchanges that exists in RabbitMQ (As of RabbitMQ version 3.6.2 there are 8 default exchanges).

  In a clustered environment, the total number of exchanges that have been declared by the consumers connected to the other RabbitMQ nodes in the cluster, are also included in the above total.

- Total active queues: Tags: dev

  The total number of both durable and non-durable queues that currently exist in the RabbitMQ node + the total number of both durable and non-durable queues that were declared in the RabbitMQ node since the last time the metric was read by the WombatOAM server.

  In a clustered environment, the total number of both durable and non-durable queues created by the other RabbitMQ nodes in the cluster, are also included in the above total.

Simple metrics:

Note that the RabbitMQ's management plugin does not expose most of these metrics.

- Channels created: Total number of created channels. Tags: dev
• Channels closed: Total number of closed channels. Tags: dev
• Connections created: Total number of created connections. Tags: dev
• Connections closed: Total number of closed connections. Tags: dev
• Consumers created: Total number of created consumers. Tags: dev
• Consumers deleted: Total number of deleted consumers. Tags: dev
• Permission created: Total number of permissions created. Tags: dev
• Permission deleted: Total number of permissions deleted. Tags: dev
• Queue created: Total number of created queues. Tags: dev
• Queue deleted: Total number of deleted queues. Tags: dev
• User created: Total number of created users. Tags: dev, op
• User deleted: Total number of deleted users. Tags: dev, op
• User password changed: Total number of user password changes. Tags: dev, op
• User password cleared: Total number of user passwords being cleared. Tags: dev, op
• User authentication failure: Total number of user authentication failures. Tags: dev, op
• User authentication success: Total number of user authentication successes. Tags: dev, op
• Exchanges created: Total number of exchanges created. Tags: dev, op

**Message rate metrics:**

• Publish rate: Tags: dev The number of published messages handled by the RabbitMQ server per second.
• Deliver No Ack rate: The number of messages delivered to all the consumers (but not yet acknowledged) by the RabbitMQ server per second.
• Deliver rate: Tags: dev The number of messages delivered to all the consumers (with acknowledgments) by the RabbitMQ server per second.
• Get No Ack rate: Tags: dev The number of messages retrieved by the consumers (using the basic.get call and not yet acknowledged) by the RabbitMQ server per second.
• Get rate: Tags: dev The number of messages retrieved by the consumers (using the basic.get call with acknowledgements) by the RabbitMQ server per second.
• Deliver Gets rate: Tags: dev The number of Get + Get no ack + Deliver + Deliver no ack messages delivered to the consumers by the RabbitMQ server per second.
• Ack rate: Tags: dev The number of consumer acknowledgements handled by the RabbitMQ server per second.
• Return Unroutable rate: Tags: dev The number of published messages that could not be routed to any queue, handled by the RabbitMQ server per second.
• Confirm rate: Tags: dev The number of confirmations sent to the...
publishers (who are in confirm mode) by the RabbitMQ server per second.

- Redeliver rate: Tags: dev The number of messages re-delivered to consumers (e.g. when the intended consumer crashes before sending back an acknowledgement or through invoking basic.recover) by the RabbitMQ server per second.

**Statistic metrics:**

- Channel consumer count: Total number of consumers that have subscribed to the broker to consume messages. Same as "Consumers created". Tags: dev
- Channel messages unacknowledged: Total number of unacknowledged messages. Tags: dev
- Channel messages unconfirmed: Total number of messages unconfirmed by the broker. Tags: dev
- Channel messages uncommitted: Total number of messages uncommitted. Tags: dev
- Channel acks uncommitted: Total number of the messages uncommitted in all channels. Tags: dev
- Queue messages: Total number of messages in queues. Tags: dev, op
- Queue messages ready: Total number of messages ready in queues. Tags: dev
- Queue messages unacknowledged: Total number of messages unacknowledged in queues. Tags: dev
- Queue message bytes: Total number of bytes in messages in queues. Tags: dev, op
- Queue message bytes ready: Total number of bytes in ready messages in queues. Tags: dev
- Queue message bytes unacknowledged: Total number of bytes in unacknowledged messages in queues. Tags: dev
- Connection received octets: Total number of octets received. Tags: dev
- Connection received packets: Total number of packets received. Tags: dev
- Connection sent octets: Total number of octets sent. Tags: dev
- Connection sent packets: Total number of sent packets. Tags: dev
- Connection send pending packets: Total number pending packets to be sent. Tags: dev
- Connection active channels: Total number of active channels. Tags: dev, op

**Memory consumption measurements**

**System and RabbitMQ Configurations**

- System Memory = 8 GB
- Number of Cores = 2
- RabbitMQ version = 3.6.2
- Default clean stats period = 300 seconds. (This higher value was set to prevent any impact of the plugin's automatic clean-up mechanism.)
We are using the publish_deliver_ack message mechanism using fanout exchange. This means the message types that will be handled are publish, deliver, ack and deliver_gets.

All the memory values are in Kilobytes.

Some definitions

- **Producer** = A client application process/thread that sends messages to the RabbitMQ server. One producer corresponds to one AMQP channel.

- **Consumer** = A client application process/thread that consumes messages from the RabbitMQ server. One consumer corresponds to one AMQP channel.

- **Best case scenario** = A scenario where any given client application behaves either as a producer or as a consumer.

- **Worst case scenario** = A scenario where all the client applications behaves both as a producer and a consumer. So, in a worst case scenario,

\[\text{#Producers} + \text{#Consumers} = \text{#Client applications}\]

**Measurement Configs**

\[
\begin{align*}
C0 &= \{\text{#Consumers} = 0, \text{#Queues} = 0, \text{#Connections} = 0, \text{#Channels} = 0, \\
    &\quad \text{#Producers} = 0, \text{#publish} = 0, \text{#deliver} = 0, \text{#ack} = 0\} \\
C1 &= \{\text{#Consumers} = 1, \text{#Queues} = 1, \text{#Connections} = 2, \text{#Channels} = 2, \\
    &\quad \text{#Producers} = 1, \text{#publish} = 1, \text{#deliver} = 1, \text{#ack} = 1\} \\
C2 &= \{\text{#Consumers} = 2, \text{#Queues} = 2, \text{#Connections} = 3, \text{#Channels} = 3, \\
    &\quad \text{#Producers} = 1, \text{#publish} = 1, \text{#deliver} = 2, \text{#ack} = 2\} \\
C3 &= \{\text{#Consumers} = 3, \text{#Queues} = 3, \text{#Connections} = 4, \text{#Channels} = 4, \\
    &\quad \text{#Producers} = 1, \text{#publish} = 1, \text{#deliver} = 3, \text{#ack} = 3\} \\
C4 &= \{\text{#Consumers} = 4, \text{#Queues} = 4, \text{#Connections} = 5, \text{#Channels} = 5, \\
    &\quad \text{#Producers} = 1, \text{#publish} = 1, \text{#deliver} = 4, \text{#ack} = 4\} \\
C5 &= \{\text{#Consumers} = 5, \text{#Queues} = 5, \text{#Connections} = 6, \text{#Channels} = 6, \\
    &\quad \text{#Producers} = 1, \text{#publish} = 1, \text{#deliver} = 5, \text{#ack} = 5\}
\end{align*}
\]

Measurements for the various data structures

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
</tr>
</thead>
<tbody>
<tr>
<td>rabbit_metrics (proplist)</td>
<td>1.15</td>
<td>1.15</td>
<td>1.15</td>
<td>1.15</td>
<td>1.15</td>
<td>1.15</td>
</tr>
<tr>
<td>channel_stats (dict)</td>
<td>0.05</td>
<td>0.23</td>
<td>0.32</td>
<td>0.41</td>
<td>0.49</td>
<td>0.59</td>
</tr>
<tr>
<td>queue_stats (dict)</td>
<td>0.11</td>
<td>0.23</td>
<td>0.34</td>
<td>0.46</td>
<td>0.57</td>
<td>0.68</td>
</tr>
<tr>
<td>connection_stats (dict)</td>
<td>0.05</td>
<td>0.26</td>
<td>0.36</td>
<td>0.46</td>
<td>0.55</td>
<td>0.66</td>
</tr>
<tr>
<td>stat_counters (proplist)</td>
<td>0.45</td>
<td>0.45</td>
<td>0.45</td>
<td>0.45</td>
<td>0.45</td>
<td>0.45</td>
</tr>
<tr>
<td>message_counts (dict)</td>
<td>0.05</td>
<td>0.95</td>
<td>1.07</td>
<td>1.18</td>
<td>1.30</td>
<td>1.42</td>
</tr>
<tr>
<td>total_active_entities (dict)</td>
<td>0.58</td>
<td>0.58</td>
<td>0.58</td>
<td>0.58</td>
<td>0.58</td>
<td>0.58</td>
</tr>
<tr>
<td>closed_channels (list)</td>
<td>0.0</td>
<td>0.07</td>
<td>0.11</td>
<td>0.14</td>
<td>0.18</td>
<td>0.21</td>
</tr>
<tr>
<td>process_heap_size</td>
<td>6.61</td>
<td>5.59</td>
<td>25.47</td>
<td>5.59</td>
<td>2.52</td>
<td>6.61</td>
</tr>
<tr>
<td>process_memory</td>
<td>53.39</td>
<td>73.0</td>
<td>47.3</td>
<td>49.3</td>
<td>53.39</td>
<td>69.7</td>
</tr>
</tbody>
</table>

**Static Structures**

All the proplists and the total_active_entities dictionary remains almost constant. This is because they just increment/decrement integer values stored in them. So the memory consumed by these static structures does not grow.

**Dynamic structures**

- **channel_stats dictionary**

Between C0 to C5, the channel_stats increases by 0.09 kb per producer/consumer. So,

Memory consumption for channel stats =
\[0.05 + (0.09 \times (#\text{Producers} + #\text{Consumers})) \rightarrow [1]\]

**queue\_stats dictionary**

Between C0 to C5, the queue\_stats increases by 0.11 kb per producer/consumer which creates the new queue. So,

\[
\text{Memory consumption for queue\_stats} = 0.11 + (0.11 \times (#\text{Producers that created queues} + #\text{Consumers that created the queues})) \rightarrow [2]
\]

**connection\_stats dictionary**

Between C0 to C5, the connection\_stats increases by 0.10 kb per producer/consumer. So,

\[
\text{Memory consumption for connection\_stats} = 0.05 + (0.10 \times (#\text{Producers} + #\text{Consumers})) \rightarrow [3]
\]

**message\_counts dictionary**

Between C0 and C1, when 4 new message types were added to the dict, the size increased by 0.90 kb. So, for every new message type that is encountered, the message\_counts increases on an average by 0.22 kb. That is, one message type with one channel data occupies 0.22 kb. Given the fact that we process 10 types of messages, if we have at least one channel producing at least one type of message, with all the 10 message types, the dict will occupy 2.2 Kb.

Between C1 to C5, with four types of messages in the message\_counts dict, for every new consumer/producer added, the dict increases by 0.11 kb. So, for any new producer/consumer added to an existing message type, the dict increases by \(0.11 / 4 = 0.027\) Kb per message type.

So,

\[
\text{Memory occupied by one message type with one channel data} = 0.22 \text{ Kb}
\]
\[
\text{Memory occupied by one channel data} = 0.027 \text{ kb}
\]
\[
\text{Memory occupied by other terms for one message type} = 0.22 \text{ Kb} - 0.027 = 0.193 \text{ Kb}
\]

Message types can be classified into two categories.

- Producer types (publish, return_unroutable, confirm) and
- Consumer types (deliver_no_ack, deliver, ack, get_no_ack, get and redeliver)

So,

- Number of Producer type messages = 3
- Number of Consumer type messages = 6

So,

\[
\text{Memory consumed by one producer message type content} = 0.193 + (#\text{Producers producing the particular producer message type} \times 0.027) \rightarrow [4]
\]

\[
\text{Memory consumed by one consumer message type content} = 0.193 + (#\text{Consumers producing the particular consumer message type} \times 0.027) \rightarrow [5]
\]

As deliver\_gets = get + get\_no\_ack + deliver + deliver\_no\_ack

\[
\text{Memory consumed by deliver gets message type content} = 0.193 + (#\text{Consumers producing the consumer message types} \times 0.027) \rightarrow [6]
\]

In the best case scenario, [4], [5] and [6] are valid.
In the worst case scenario,

[4] becomes

Memory consumed by one producer message type content = 
0.193 + ((#Producers + #Consumers) * 0.027) ---> [7]

[5] becomes

Memory consumed by one consumer message type content = 
0.193 + ((#Producers + #Consumers) * 0.027) ---> [8]

and [6] becomes

Memory consumed by deliver_gets message type content = 
0.193 + ((#Producers + #Consumers) * 0.027) ---> [9]

So,

Memory consumed by the message_counts dict = 
0.05 + 
(Memory consumed by one producer message type content * 
  #Producer type messages) + 
(Memory consumed by one consumer message type content * 
  #Consumer type messages) + 
(Memory consumed by deliver_gets message type content) ---> [10]

closed channels

Every channel that gets closed, occupies 0.035 Kb in the closed channels list.

So,

Memory consumed by closed channels list = 
0.035 * #Channels closed ----> [11]

Finally,

Total memory occupied by the plugin process = 
(Constant memory space for proplists, 
total_active_entities dict and 
other variables in the process' State) + 
[1] + [2] + [3] + [10] + 

In reality, the set of Producers and the set of consumers could intersect with each other, though not completely overlapping.

Some calculations

Based on [12] above, we have the following outcomes on a 32-bit architecture system.

- CM = Constant Memory = 2.17 Kb
- PM = Default Process Memory = 1.32 Kb
- CSM = Channel Stats Memory
- QSM = Queue Stats Memory
- CNSM = Connection Stats Memory
- MCM = Message Counts Memory
- CCM = Closed Channels Memory
- TM = Total Memory
- P = Producer
- C = Consumer

In the best case scenario

<table>
<thead>
<tr>
<th>CSM</th>
<th>QSM</th>
<th>CNSM</th>
<th>MCM</th>
<th>CCM</th>
<th>TM</th>
</tr>
</thead>
</table>

rabbitmq channels plugin

Description

The rabbitmq_channels plugin provides the ability to monitor metrics for individual, active channels. When enabled, this plugin will acquire all active channels from the managed rabbit node and based on its configuration, it generates corresponding metrics for each acquired channel.

All channel metric names follow the following format:

\~

where:

<CHANNEL-NAME> is the native acquired RabbitMQ channel name.

<METRIC-ATTRIBUTE> is one of the following metric attributes;

- number
- reductions
- consumer_count
- messages_unacknowledged
- messages_unconfirmed
- messages_uncommitted
- acks_uncommitted
- prefetch_count
- global_prefetch_count


Supported versions.

Official stable Rabbit releases newer than 2.3.0 are supported.

Applications it depends on

rabbit

Modules

- wombat_plugin_rabbitmq_channels
- wombat_plugin_rabbitmq_metrics

Configuration options

- metric: The metric attribute to be applied and monitored on all channels. Default value: consumer_count.
- refresh_interval: Time period after which new metric values are acquired from the rabbit node. Default value: 1000.
• limit: Maximum of the number channel metrics the plugin is allowed to handle on each refresh interval. Default value: 1000.

Example wombat.config entry

```erlang
%% Set the metric attribute
{set, wo_plugins, plugins, rabbitmq_channels, metric, channels}.

%% Change refresh interval after which metric values are refreshed
{set, wo_plugins, plugins, rabbitmq_channels, refresh_interval, 1500}.

%% Limit the maximum number of processed channel metrics
{set, wo_plugins, plugins, rabbitmq_channels, limit, 1500}.
```

Metrics reported

If the plugin is restarted, all these metrics are reset.

Channel metrics:

- \sim \textbf{number}: Tags: dev
  
  The identity and count number of the channel \\

- \sim \textbf{reductions}: Tags: dev
  
  The number of reductions executed by the channel.

- \sim \textbf{consumer_count}: Tags: dev
  
  The number of active consumers on the channel.

- \sim \textbf{messages_unacknowledged}: Tags: dev
  
  The number of unacknowledged messages on the channel.

- \sim \textbf{messages_unconfirmed}: Tags: dev
  
  The number of unconfirmed messages on the channel.

- \sim \textbf{messages_uncommitted}: Tags: dev
  
  The number of uncommitted messages on the channel.

- \sim \textbf{acks_uncommitted}: Tags: dev
  
  The number of uncommitted message acknowledgements on the channel.

- \sim \textbf{prefetch_count}: Tags: dev
  
  The channel prefetch count limit.

- \sim \textbf{global_prefetch_count}: Tags: dev
  
  The global channel prefetch count limit.

rabbitmq channel messages plugin

Description

The rabbitmq channel messages plugin currently provides the ability to monitor message rates for individual, active channels. When enabled, this plugin will acquire all active channels from the managed rabbit node and based on its configuration, it generates corresponding message rate
metrics for each acquired channel.

All channel metric names follow the following format;

\~ \~ message rate \~ (<\METRIC-PERIOD\>s average)

where:

<CHANNEL-NAME> is the native acquired RabbitMQ channel name,

<VHOST> is the RabbitMQ VHost the channel belongs to,

<TYPE> is one of

- publish
- confirm
- unroutable
- get
- get no ack
- deliver
- deliver no ack
- redeliver
- ack
- deliver get

<METRIC-PERIOD> is 60 (seconds) by default, but can be changed through the configuration;

Example of a channel metric would be: 192.168.1.209:55605 \rightarrow 192.168.1.211:5672 (4)~/~message rate publish~(60s average).

Supported versions.

All official 3.7 releases are supported and tested.

Applications it depends on

rabbit

Modules

- wombat_plugin_rabbitmq_channel_messages
- wombat_plugin_rabbitmq_metrics

Configuration options

- metric_period: specifies the granularity of the samples in milliseconds. It is recommended to set it to a multiple of 5000. The minimum value is 5000. When set to a lower value, it will be ignored and 5000 will be used instead. The default value is 60000, that is 60 seconds.

Example wombat.config entry

```
1
2  \%
3  \% Change the metric period
4  {set, wo_plugins, plugins, rabbitmq_channel_messages, metric_period, 5000}.
```

Metrics reported

If the plugin is restarted, old metrics for non-existing channels are removed.

Channel message rate metrics:

- \~ \~ message rate get \~ (<\METRIC-PERIOD\>s average)
The average rate of Basic.get operations on \ in the last \ seconds.

- \( \sim \) ~ \textit{message rate get no ack} \( \langle \text{\textbackslash METRIC-PERIOD} \rangle \text{s average} \) Tags: dev, op

The average rate of Basic.get operations with auto acknowledgement on \ in the last \ seconds.

- \( \sim \) ~ \textit{message rate deliver} \( \langle \text{\textbackslash METRIC-PERIOD} \rangle \text{s average} \) Tags: dev, op

The average rate of messages delivered through \ in the last \ seconds.

- \( \sim \) ~ \textit{message rate deliver no ack} \( \langle \text{\textbackslash METRIC-PERIOD} \rangle \text{s average} \) Tags: dev, op

The average rate of messages delivered through \ with auto acknowledgement in the last \ seconds.

- \( \sim \) ~ \textit{message rate redeliver} \( \langle \text{\textbackslash METRIC-PERIOD} \rangle \text{s average} \) Tags: dev, op

The average rate of messages redelivered through \ in the last \ seconds.

- \( \sim \) ~ \textit{message rate ack} \( \langle \text{\textbackslash METRIC-PERIOD} \rangle \text{s average} \) Tags: dev, op

The average rate of message acknowledgements received for delivered messages in \ in the last \ seconds.

- \( \sim \) ~ \textit{message rate deliver get} \( \langle \text{\textbackslash METRIC-PERIOD} \rangle \text{s average} \) Tags: dev, op

The average rate of messages delivered with Basic.get operations through \ in the last \ seconds.

- \( \sim \) ~ \textit{message rate publish} \( \langle \text{\textbackslash METRIC-PERIOD} \rangle \text{s average} \) Tags: dev, op

The average rate of messages published to \ in the last \ seconds.

- \( \sim \) ~ \textit{message rate confirm} \( \langle \text{\textbackslash METRIC-PERIOD} \rangle \text{s average} \) Tags: dev, op

The average rate of publish confirms sent from \ in the last \ seconds.

- \( \sim \) ~ \textit{message rate unroutable} \( \langle \text{\textbackslash METRIC-PERIOD} \rangle \text{s average} \) Tags: dev, op

The average rate of messages sent through \ in the last \ seconds that could not be routed to any queue.

\textbf{rabbitmq connections plugin}

\textbf{Description}

The \texttt{rabbitmq connections} plugin provides the ability to monitor metrics for individual, active connections. When enabled, this plugin will acquire all active connections from the managed rabbit node and based on its configuration, it generates corresponding metrics for each acquired connection.

All connection metric names follow the following format;

\( \sim \) ~ \( \sim \)
where:

<CONNECTION-NAME> is the native acquired RabbitMQ connection name.

<METRIC-ATTRIBUTE> is one of the following metric attributes;

- frame_max
- channel_max
- recv_oct
- recv_cnt
- send_oct
- messages_ram
- send_cnt
- send_pend
- channels


**Supported versions.**

Official stable Rabbit releases newer than 2.3.0 are supported.

**Applications it depends on**

rabbit

**Modules**

- wombat_plugin_rabbitmq_connections
- wombat_plugin_rabbitmq_metrics

**Configuration options**

- metric: The metric attribute to be applied and monitored on all connections. Default value: channels.
- refresh_interval: Time period after which new metric values are acquired from the rabbit node. Default value: 1000.
- limit: Maximum of the number connection metrics the plugin is allowed to handle on each refresh interval. Default value: 1000.

**Example wombat.config entry**

```
1 2 3 4 5 6 7 8
%% Set the metric attribute
{set, wo_plugins, plugins, rabbitmq_connections, metric, channels}.

%% Change refresh interval after which metric values are refreshed
{set, wo_plugins, plugins, rabbitmq_connections, refresh_interval, 1500}.

%% Limit the maximum number of processed connection metrics
{set, wo_plugins, plugins, rabbitmq_connections, limit, 1500}.
```

**Metrics reported**

If the plugin is restarted, all these metrics are reset.

**Connection metrics:**

- \~ frame_max : Tags: dev
The maximum allowed frame size for connection \.

- \( channel\_max \) : Tags: dev

The maximum allowed channel number allowed for the connection.

- \( recv\_oct \) : Tags: dev

The total number of received octets for the connection.

- \( recv\_cnt \) : Tags: dev

The total number of received packets for the connection.

- \( send\_oct \) : Tags: dev

The total number of sent octets by the connection.

- \( send\_cnt \) : Tags: dev

The total number of sent packets for the connection.

- \( send\_pend \) : Tags: dev

The total number of packets awaiting to be sent by the connection.

- \( channels \) : Tags: dev

The total number of channels on the connection.

**rabbitmq queues plugin**

**Description**

The rabbitmq queues plugin provides the ability to monitor metrics for individual queues on the rabbit node being managed. Based on its configuration, the plugin provides monitoring of static and dynamic queues. Static queues are individually defined in config, whereas dynamic queues are defined by a regular expression, from which matching queue names are acquired and used to create metric capabilities. This implies that metric names for dynamic queues may or may not be known when Wombat is started-up.

All queue metric names follow the following format;

\( \~ \~ \~ \\)

where:

<QUEUE-NAME> is the configured static queue name, or acquired dynamic queue name from the regular expression. Queue names vary based on the rabbit node deployment.

<VIRTUAL-HOST> is the virtual host name on which the queue, \( \_ \) resides. The virtual host also varies based on the specific rabbit node deployment.

<METRIC-ATTRIBUTE> is one of the following metric attributes;

- messages
- messages_ready
- messages_unacknowledged
- messages_ready_ram
- messages_unacknowledged_ram
- messages_ram
- messages_persistent
- message_bytes
Example of a queue metric: test.queue-test.vhost-messages.

**Supported versions.**

Official stable Rabbit releases newer than 2.3.0 are supported.

**Applications it depends on**

rabbit

**Modules**

- wombat_plugin_rabbitmq_queues
- wombat_plugin_rabbitmq_metrics

**Configuration options**

- **static_queues**: Defines a list of tuple definitions for queues. Each queue definition is a tuple {QueueName1, VirtualHost, MetricAttribute}. Default value: [].

- **dynamic_queues**: Defines two configuration tags used to generate dynamic queue metrics. These tags are match_spec and metric, defined next.

  - **dynamic_queues.match_spec**: This is the regular expression used for acquiring all matching queue names on the rabbit node, from which metric names are generated from. Default value: ".*".

  - **dynamic_queues.metric**: This is the metric attribute to be applied and monitored to all matched queues. Default value: messages.

- **refresh_interval**: Time period after which new metric values are acquired from the rabbit node. Default value: 1000.

- **ordering**: Defines the manner in which queue metrics will be ordered. The following values can be used: true, ascending, ascend, descending, descend. Value true simply means ascending. Default value: ascending

**Example wombat.config entry**

```erlang
%% Set the static queues
{set, wo_plugins, plugins, rabbitmq_queues, static_queues,
  [{<<"test.queue.1">>, <<"/">>, messages},
   {<<"test.queue.2">>, <<"/">>, memory}]}.

%% Set the dynamic queues
{set, wo_plugins, plugins, rabbitmq_queues, dynamic_queues,
  [{match_spec, "^test"},
   {metric, consumers}]}.

%% Change refresh interval after which metric values are refreshed
{set, wo_plugins, plugins, rabbitmq_queues, refresh_interval, 1000}.
```

%% Set the ordering or metrics to ascending
{set, wo_plugins, plugins, rabbitmq_queues, ordering, ascending}.

Metrics reported
If the plugin is restarted, all these metrics are reset.

Overview metrics:
- \(\sim\) \(\sim\) messages : Tags: dev
  The total number messages in queue \(\sim\) residing in vhost \(\sim\).
- \(\sim\) \(\sim\) messages_ready : Tags: dev
  The number of messages within the queue, ready for consumption.
- \(\sim\) \(\sim\) messages_unacknowledged : Tags: dev
  The number of messages within the queue, that have not yet been acknowledged.
- \(\sim\) \(\sim\) messages_ready_ram : Tags: dev
  The number of queue messages held in RAM, which are ready for delivery.
- \(\sim\) \(\sim\) messages_unacknowledged_ram : Tags: dev
  The number of queue messages held in RAM, that have not been acknowledged.
- \(\sim\) \(\sim\) messages_ram : Tags: dev
  The total number of queue messages held in RAM.
- \(\sim\) \(\sim\) messages_persistent : Tags: dev
  The total number of persistant messages of the queue.
- \(\sim\) \(\sim\) disk_reads : Tags: dev
  The total number of disk read operations carried out by the queue.
- \(\sim\) \(\sim\) disk_writes : Tags: dev
  The total number of disk write operations carried out by the queue.
- \(\sim\) \(\sim\) consumers : Tags: dev
  The total number of clients consuming messages from the queue.
- \(\sim\) \(\sim\) memory : Tags: dev
  The total memory utilisation of the queue.

rabbitmq queue messages plugin

Description
The rabbitmq_queue_messages plugin currently provides the ability to monitor message rates for individual queues. When enabled, this plugin will acquire all existing from the managed rabbit node and based on its configuration, it generates corresponding message rate metrics for each acquired queue.

All queue metric names follow the following format;
where:

<QUEUE-NAME> is the name of the acquired RabbitMQ queue,

<VHOST> is the RabbitMQ VHost the queue belongs to,

<TYPE> is one of
  
  • get
  • get no ack
  • deliver
  • deliver no ack
  • redeliver
  • ack
  • deliver get

<METRIC-PERIOD> is 60 (seconds) by default, but can be changed through the configuration;

Example of a queue metric would be: test-queue ~ / ~ message rate publish ~ (60s average).

Supported versions.

All official 3.7 and later releases are supported and tested.

Applications it depends on

rabbit

Modules

• wombat_plugin_rabbitmq_queue_messages
• wombat_plugin_rabbitmq_metrics

Configuration options

• metric_period: specifies the granularity of the samples in milliseconds. It is recommended to set it to a multiple of 5000. The minimum value is 5000. When set to a lower value, it will be ignored and 5000 will be used instead. The default value is 60000, that is 60 seconds.

Example wombat.config entry

```plaintext
1 2
1 % Change the metric period
2 {set, wo_plugins, plugins, rabbitmq_queue_messages, metric_period, 5000}.
```

Metrics reported

Queue message rate metrics:

• \ ~ \ ~ message rate get ~ (\METRIC-PERIOD\)s average) Tags: \ dev, \ op

The average rate of Basic.get operations on \ in the last \ seconds.

• \ ~ \ ~ message rate get no ack ~ (\METRIC-PERIOD\)s average) Tags: \ dev, \ op

The average rate of Basic.get operations with auto acknowledgement on \ in the last \ seconds.

• \ ~ \ ~ message rate deliver ~ (\METRIC-PERIOD\)s
average) Tags: dev, op

The average rate of messages delivered from \ in the last \ seconds.

- \ ~ \ ~ message rate deliver no ack ~ (</METRIC-PERIOD)>s average) Tags: dev, op

The average rate of messages delivered from \ with auto acknowledgement in the last \ seconds.

- \ ~ \ ~ message rate redeliver ~ (</METRIC-PERIOD)>s average) Tags: dev, op

The average rate of messages redelivered from \ in the last \ seconds.

- \ ~ \ ~ message rate ack ~ (</METRIC-PERIOD)>s average) Tags: dev, op

The average rate of message acknowledgements received for delivered messages in \ in the last \ seconds.

- \ ~ \ ~ message rate deliver get ~ (</METRIC-PERIOD)>s average) Tags: dev, op

The average rate of messages delivered with Basic.get operations from \ in the last \ seconds.

**RabbitMQ Recovery from netsplit:**

**About:**

This service enables the recovery for a RabbitMQ cluster after having a netsplit in the network which leads to inconsistency in the mnesia database.

**Requirements:**

In order to perform the recovery, it is required that: - All the nodes of the cluster are added to Wombat. - All the nodes are running. - All the nodes have Rabbit application running - Network connection is stable

**Structure:**

Every RabbitMQ Node added to wombat will have the rabbitmq/fix netsplit plugin enabled, which in turn, going to be responsible for handling the orders coming from the woservices rabbitmq/fix netsplit service which manages the recovery process.

The recovery process can be started from any node in the cluster, and then Wombat will gather information about the cluster nodes and inform the user with the cluster partitioning situation: - Number of partitions - Nodes in every Partition

The user can choose between two ways of recovery: Automatic recovery (Default Button) Manual recovery: by giving the user the ability to choose which partition to be the winner and restart all the nodes in other ones.

Depending on the number of partitions exists, buttons for every partition will be shown in order to make the user select which partition is the winner.

**Important:**

It is important to know that by choosing a partition, this partition is the winner and all nodes in the other ones are going to be restarted (losers).
The Default button is responsible for performing the automatic recovery based on the Auto-heal algorithm (which is going to select the losers based on the number of nodes on every partition and the number of local connections in every node).

In case this request is started and no partitioning problem exists, the user is going to be informed about it.

**riak_core plugin**

**Description**

The `riak_core` plugin report metrics and notifications from `riak_core`. This plugin only focuses on `riak_core` metrics, so it is possible to use it with any product using `riak_core`.

**Applications it depends on**

`riak_core`

**Modules**

- `wombat_plugin_riak_core`
- `wombat_plugin_riak_core_metrics`

**Reports**

**Alarms reported**

**Riak Node down**

Please find the detailed alarm description in the Alarms documentation.

**Notifications**

- `vnode_started`
- `vnode_terminated`
- `node_down`
- `node_up`
- `handoff_initiated`
- `handoff_completed`
- `handoff_cancelled`
- `vnode_state_transfer_initiated`
- `vnode_state_transfer_completed`
- `command_received`

Traced functions of Riak, which are called when executing Riak commands, (for example calling `riak-admin transfers`). These function calls are reported to WombatOAM including the actual arguments of the function call. Calls to the following functions are reported:

```
riak_core_console:member_status/1
riak_core_console:ring_status/1
riak_core_console:print_member_status/2
riak_core_console:stage_leave/1
riak_core_console:stage_remove/1
riak_core_console:stage_replace/1
riak_core_console:stage_resize_ring/1
riak_core_console:stage_force_replace/1
riak_core_console:print_staged/1
riak_core_console:commit_staged/1
riak_core_console:clear_staged/1
riak_core_console:transfer_limit/1
riak_core_console:pending_claim_percentage/2
riak_core_console:pending_nodes_and_claim_percentages/1
```
riak_core_console:transfers/1
riak_core_console:add_user/1
riak_core_console:alter_user/1
riak_core_console:del_user/1
riak_core_console:add_group/1
riak_core_console:alter_group/1
riak_core_console:del_group/1
riak_core_console:add_source/1
riak_core_console:del_source/1
riak_core_console:grant/1
riak_core_console:revoke/1
riak_core_console:print_users/1
riak_core_console:print_user/1
riak_core_console:print_sources/1
riak_core_console:print_groups/1
riak_core_console:print_group/1
riak_core_console:print_grants/1
riak_core_console:security_enable/1
riak_core_console:security_disable/1
riak_core_console:security_status/1
riak_core_console:ciphers/1
riak_core_console:stat_show/1
riak_core_console:stat_info/1
riak_core_console:stat_enable/1
riak_core_console:stat_disable/1
riak_core_console:stat_reset/1
riak_core_console:command/1
riak_kv_console:staged_join/1
riak_kv_console:bucket_type_status/1
riak_kv_console:bucket_type_activate/1
riak_kv_console:bucket_type_create/1
riak_kv_console:bucket_type_update/1
riak_kv_console:bucket_type_list/1
riak_kv_console:join/1
riak_kv_console:leave/1
riak_kv_console:remove/1
riak_kv_console:down/1
riak_kv_console:status/1
riak_kv_console:vnode_status/1
riak_kv_console:ringready/1
riak_kv_console:aae_status/1
riak_kv_console:ensemble_status/1
riak_kv_console:repair_2i/1
riak_kv_console:cluster_info/1
riak_kv_console:reload_code/1
riak_kv_console:reip/1
riak_kv_console:reformat_indexes/1
riak_kv_console:reformat_objects/1
yz_console:aae_status/1
yz_console:switch_to_new_search/1
riak_core_cluster_cli:register_cli/0
riak_core_cluster_cli:status/2
riak_core_cluster_cli:partition_count/2
riak_core_cluster_cli:partitions/2
riak_core_cluster_cli:partition/2
riak_core_handoff_cli:register_cli/0
riak_core_handoff_cli:set_transfer_limit/3
riak_core_handoff_cli:set_transfer_limit/5

**Metrics reported**

- Vnode count (total) Tags: dev, op
- Vnode count (primary) Tags: dev, op
- Vnode count (secondary) Tags: dev, op
- Member count (total) Tags: dev, op
- Member count (active) Tags: dev, op
- Member count (down) Tags: dev, op
Member count (joining) Tags: dev, op
Member count (leaving) Tags: dev, op
Member count (unreachable) Tags: dev, op
Active handoff count (inbound) Tags: dev, op

Configuration options
None.

**riak_kv plugin**

**Description**
A plugin for riak_kv which includes counters for the different backends used by Riak.

**Applications it depends on**
riak_kv

**Modules**
- wombat_plugin_riak_kv
- wombat_plugin_riak_kv_metrics

**Reports**
Metrics reported:
- Disk used by bitcask Tags: op
- Disk used by leveldb Tags: op
- Number of active hooks Tags: dev

Per index counters:
- AAE repairs Tags: dev

**Configuration options**
None.

**riak_repl plugin**

**Description**
The riak_repl plugin monitors information from the Multi-Datacenter Replication subsystem in Riak.
It can give an overview of the current status of the realtime queue, which can give indications that a full sync might be needed.
During fullsync it is possible to see the progress and information on previous fullsyncs.

**Applications it depends on**
riak_repl

**Modules**
- wombat_plugin_riak_repl
- wombat_plugin_riak_repl_metrics
**Reports**

The plugin reports the metrics listed below.

- Realtime queue percent used Tags: dev, op
- Realtime queue bytes used Tags: dev, op
- Realtime queue overload drops Tags: dev, op
- Realtime sent kbps Tags: dev, op
- Realtime recv kbps Tags: dev, op
- Fullsync sent kbps Tags: dev, op
- Fullsync recv kbps Tags: dev, op

Counters per inter-cluster connection:

- Busy nodes Tags: dev, op
- Fullsyncs completed Tags: dev
- Last fullsync duration Tags: dev

Counters per partition state:

- Queued partitions Tags: dev
- Successful exit partitions: When this reaches ring_size the fullsync is finished. Tags: dev
- Starting partitions Tags: dev
- Partitions in progress Tags: dev
- Partitions waiting for retry Tags: dev
- Error exits partitions Tags: dev, op
- Retry exits partitions Tags: dev, op
- Soft retry exits partitions Tags: dev, op

**Configuration options**

None.
sasl plugin

Description
The sasl plugin subscribes to the SASL alarm_handler included in OTP, and forwards those alarms to WombatOAM.

Please find the detailed alarm descriptions in the Alarms documentation.

Applications it depends on
sasl

Modules
- wombat_plugin_sasl
- wombat_plugin_sasl_handler

Reports
The plugin reports the following SASL alarms (in pretty-printed format):
- disk_almost_full
- system_memory_high_watermark
- process_memory_high_watermark

It will also report any other alarm raised by the alarm_handler.

Configuration options
None.

worker_pool plugin

Description
The worker_pool plugin reports metrics from the active pools running on the node.

Applications it depends on
worker_pool

The plugin can only monitor worker_pool version 2.2.1 and above.

Modules
wombat_plugin_worker_pool

Reports
The plugin reports the following metrics:
- pools The total number of pools on the system Tags: dev, op
- [POOL].size The number of workers the pool has Tags: dev, op
- [POOL].queue_size The size of the task queue the pool has yet to run Tags: dev, op
- [POOL].total_memory Total memory used by the pool Tags: dev, op

These metrics allow you to monitor the state of your worker pools, aiding the optimisation of pool sizes and pool strategies.

yokozuna plugin
Description
A plugin for Riak's yokozuna application. Yokozuna is the new implementation of Riak Search built on top of Apache Solr.

Applications it depends on
yokozuna

Modules
wombat_plugin_yokozuna_metrics

Reports

Alarms reported

Metrics reported
The following metric is reported:
  - AAE repairs Tags: dev

Configuration options
MongooseIM plugins

Description

The MongooseIM plugins help you prevent outages and understand your MongooseIM nodes. The plugins expose the most important MongooseIM metrics that are handy for both manual and automated system health checks. Also, the provided services allow you to explore the configured hosts and to observe your MongooseIM's users.

Applications it depends on

mongoose

The plugin exposing metrics (wombat_plugin_mongooseim_metrics) is compatible with MongooseIM version 1.6 and above, while the other two plugins can work with MongooseIM version 1.3 and above.

Modules

The functionalities are separated into 3 independent plugins:

- wombat_plugin_mongooseim_metrics
- wombat_plugin_mongooseim_hosts
- wombat_plugin_mongooseim_users

Metrics reported

Metrics reported under MongooseIM tree can be divided into 4 categories based on the exposed values.

Gauges

- Online users: The current number of online users on a given MongooseIM node. Tags: dev

Counters

The following metrics count events that happened within a minute.

- Auth errors last minute: Number of authentication failures. Tags: dev, op
- Logins last minute: Number of successful logins. Tags: dev, op
- Logouts last minute: Number of logouts. Tags: dev
- XMPP messages received last minute: Number of XMPP message stanzas received by the server. Tags: dev
- XMPP messages sent last minute: Number of XMPP message stanzas sent by the server. Tags: dev
- XMPP messages sent to offline user last minute: Number of XMPP message stanzas addressed to offline users. Tags: dev
- XMPP presence received last minute: Number of XMPP presence stanzas received by the server. Tags: dev
- XMPP presence sent last minute: Number of XMPP presence stanzas sent by the server. Tags: dev
- XMPP error stanzas last minute: Number of stanza errors (all stanzas with type='error'). Tags: dev, op
- XMPP stanzas dropped last minute: Number of dropped stanzas (mainly due to privacy settings). Tags: dev, op
Median execution time

These metrics are the derived median execution times of core authentication functions.

- Auth check password time median: Derived from all the measured time taken by the check_password action. In milliseconds. Tags: dev, op
- Auth does user exist time median: Derived from the milliseconds spent on evaluating the does_user_exists action. Tags: dev, op

Median data size

These metrics derive the median values of XMPP stanzas' properties.

- XMPP stanzas received size median: Derived from all the XMPP stanzas' sizes *received* by the server. Tags: dev
- XMPP stanzas sent size median: Derived from all the XMPP stanzas' sizes *sent* by the server. Tags: dev

Provided services

All the provided services are explorers, available under the Services page:

- Display configured hosts: Displays all configured XMPP hosts on the MongooseIM node.
- Display registered hooks for host: Displays all registered hooks for all or the given XMPP host.
- Display started modules for host: Displays all started modules with their configuration for all or the selected XMPP host.
- MongooseIM users: Shows the top N MongooseIM's user ordered by values such as: Memory Size, Message Queue, Data sent, Data received, Created. Data sizes are displayed in kilobytes. *This action may put significant load on the MongooseIM node. Use it with caution.*
- MongooseIM user info: Displays detailed information about the specified MongooseIM user.

Threshold based alarms

To prevent outages, you may configure the following threshold based alarms. The configuration below, which can be copied and pasted to your wombat.config file, uses the default thresholds. The best values may vary among MongooseIM clusters.

To get the best out of your MongooseIM nodes (and to properly configure WombatOAM), you may contact the MongooseIM team (mongoose-im@erlang-solutions.com).

```erlang
{set, wo_metrics, threshold_sets,
  [{nodes, all},
   {rules,
     [{name, "auth_check_overload_check_password"},
      {metric, {"MongooseIM", "Auth check password time median"}},
      {raise_level, 50000},
      {cease_level, 40000},
      {unit, numeric},
      {direction, warn_above}],
     [{name, "auth_check_overload_user_exist"},
      {metric, {"MongooseIM", "Auth does user exist time median"}},
      {unit, numeric},
      {direction, warn_above}]}
}
```
Poolboy plugin

This plugin monitors the pools managed by Poolboy. The plugin exposes health metrics about the pools, sends early warning signs as notifications, and raises an alarm when a pool turns into a critical state.

Description

The plugin is based on Poolboy's concepts. The status of a pool is a key property, which can be:

- **ready** When the number of workers are less than or equal to the initial `Size` parameter. `Size` refers to the pool size initially configured.

- **overflow** When the number of workers are greater than the pool `Size` but less than the `Size + MaxOverFlow`. `MaxOverFlow` refers to the maximum number of workers created if pool is empty.

- **full** When the number of workers are equal to the `Size + MaxOverflow`.

Applications it depends on

poolboy

The plugin only works with version 1.x.

Modules

`wombat_plugin_poolboy.erl`

Reports

The plugin reports metrics, a notification, and an alarm.

Metrics

Each pool is depicted by the following two metrics, which are gauges:

- **Number of available workers for POOL** The number of workers can be checked out. Tags: dev.

- **Percentage of running workers for POOL** The utilisation of the pool. Calculated as follows: `RunningWorkerCount * 100 / Size`. Tags: dev, op.

Notification
Pool is overloaded when the workers launched initially got used and additional workers have been launched. This happens when the pool transits into overflow state, by leaving ready state.

**Alarm**

An alarm is raised when a pool becomes full: all workers are used and no more workers can be launched.

Please find the detailed alarm description in the Alarms documentation.

**Configuration options**

None.

**Ecto plugin**

**Description**

The plugin gives insight into how Ecto performs. Its various metrics gives fine-grain information about the operations carried out by Ecto. For each operation type (e.g. SQL SELECT query) it measures the number of executed operations and the average execution time. To provide a high level overview of the operational status of Ecto, the plugin exposes 3 more metrics. The total number of executed operations, the average time required to execute the operations, and the average time while the operations were in the queue.

**Applications it depends on**

ecto

The plugin works with all Ecto versions (latest 3.x).

**Modules**

wombat_plugin_ecto

**Metrics reported**

The three global metrics providing the high level overview:

- Request Tags: dev, op.
- Request Avg ms Tags: dev, op.
- Queue Avg ms Tags: dev, op.

And the fine-grain metrics for common operations:

- OPERATION Tags: dev.
- OPERATION Avg ms Tags: dev.

Where OPERATION is one of SQL SELECT, SQL INSERT, SQL UPDATE, SQL DELETE, SQL BEGIN, SQL COMMIT, SQL ROLLBACK.

**telemetry plugin**

**Description**

The plugin detect every event that has been executed at least once with tracing. Metrics are extracted from the event_measurement, 2nd argument of the telemetry:execute/2-3 function. The first metrics appear 7 seconds after the plugin started. The values are stored in the wombat_plugin_telemetry ets table, therefore after restarting WombatOAM only the available metrics will remain visible in the Metrics
page. In order to see the proper values of the counters, it is recommended to visualize them in delta mode if necessary.

**Reporting Metrics**

The plugin detects if the event uses the `telemetry_metrics` library and based on the information the plugin generates the proper name for that event and gathers its values to report them as metrics to WombatOAM. Furthermore, the `telemetry` plugin detects counters as well. Those metrics are stored in a separate ets table called `wombat_plugin_telemetry_counters`. In addition, WombatOAM reports events that has no numeric values assigned or not recognisable as counters. All metrics are reported with `dev` tag.

**Name Of The Metrics**

As mentioned above the plugin detects if the event uses the `telemetry_metrics` library. In that case the name parameter is used to define the metrics.

In other cases the event name and the key of event measurement map are used. The key does not have to be a atom. The atom list (the name of the event and the key of the event measurement map) is converted to a binary where the atoms are connected with a `_` character.

**Application it depends on**

- `telemetry` Recommended version is 0.4.0 or above.

**Modules**

- `wombat_plugin_telemetry`
- `wombat_plugin_telemetry_handler`

**Reports**

The plugin reports the following metric to WombatOAM:

- **Number of Events**

  Tags: `dev`, `op`

  The metrics shows all available events stored in the `telemetry_handler_table`. Additional metrics are going to appear as they are getting executed with the same tags.

**Notifications**

WombatOAM sends two types of notifications:

- **info:** All events have values assigned!
- **warning:** Telemetry module metrics have no values assigned: `[]` shows a list of events without numeric values. These are not gauges, nor counters.

**Configuration options**

None.

**SumoDB plugin**

**Description**

The plugin provides info about the performance of SumoDB. Metrics is extracted by subscribing to SumoDB events. This plugin also provides a service showing the data stored for a given model.

**Applications it depends on**

- `sumo_db`

  The plugin only works with SumoDB versions 0.7.1 and above.

**Modules**

- None.
wombat_plugin_sumo_db

Metrics reported
The plugin provides the following 11 metrics for each model defined in SumoDB, i.e., for each entry listed in the docs environment variable of the sumo_db application:

- schema_created Count of schema creation. Actually, the time graph shows a single event at the time of schema creation if it happens during the measurement.
- Avg time schema_created Duration of schema creation.
- persisted Number of persisted entities (sumo:persist/2).
- Avg time persisted Average duration of persisting entities (create or update).
- deleted Number of deleted entities (sumo:delete/2).
- Avg time deleted Average duration of deleting entities.
- deleted_all Number of sumo:delete_all/1 calls.
- Avg time deleted_all Average duration of delete_all calls.
- deleted_total Number of sumo:delete_by/2 calls.
- Avg time deleted_total Average duration of delete_by calls.
- count Number of entities stored for the model.

Service
The plugin also provides a Service that displays the current data for a particular model in a grid or table view.

Yaws Plugin
The Yaws plugin collects and publishes Yaws metrics.

Description
Yaws supports virtual servers and publishes statistics on a per virtual server basis. The statistics published for each virtual server are: connections count, sessions count, and requests count.

The plugin retrieves the list of virtual servers published by Yaws and for each of the virtual servers creates three metrics, one for each of the statistics discussed above.

Dependencies
The Yaws plugin depends on the Yaws application and is compatible with the version 1.9 and higher.

Modules
wombat_plugin_yaws

Reports

Metrics
The following metrics are published in the Yaws metrics group for each Yaws virtual server:

- [Address:Port] Connections count
  Tags: dev, op
- [Address:Port] Sessions count
  Tags: dev, op
• [Address:Port] Requests count
  Tags: dev, op

An Address:Port pair represents the IP address and port number of the Yaws virtual server to which they apply to.
WombatOAM Orchestration

WombatOAM can deploy Erlang nodes in the cloud or on specified computers. The following features can be used via WombatOAM's REST interface and from the web dashboard:

- Add cloud providers. Currently, Amazon and local deployment are supported.
- Upload Erlang releases.
- Define node families. A node family is an entity which uses a certain Erlang release and can deploy it to certain providers.
- Deploy and start new Erlang nodes (this includes creating the virtual machine instances on the given provider).

Orchestration v1 and v2

There are two versions of Orchestration implemented in WombatOAM, with regards to handling node families and nodes:

- Orchestration v1 is the legacy implementation. This is the default, which will change in the future.
- Orchestration v2 is the new implementation. The aim is to be more consistent, user-friendly, fault-tolerant and scalable than v1.

Orchestration v2 can be used with the following `wombat.config` setting:

```
1 {set, wo_orch, version, v2}.
```

Providers and releases are handled in the same way regardless of the Orchestration version. On the other hand, node families and nodes are not only handled differently by the two versions, but they store the Orchestration-related data in different data tables. (So e.g. if a node family is created with v1, then WombatOAM is restarted with v2 configured, v2 won't show this node family as a node family handled by Orchestration.)

Configuring WombatOAM Orchestration

For using Orchestration, Libcloud needs to be installed from source as described in [https://github.com/esl/elibcloud#installation](https://github.com/esl/elibcloud#installation).

Before being able to deploy your distributed Erlang application via WombatOAM, some configuration is required in the `wombat.config` file.

First, enable the Orchestration tab on the dashboard:

```
1 {set, wo_dashboard, enable_orch, true}.
```

Create SSH public and private keys (using the `ssh-keygen` program) and specify their path in `wombat.config`. Assign a custom string to `cloud_prefix` (which will be prepended to the name of all your virtual machine instances, security rules, etc. on the cloud providers).

```
1 {set, wo_orch, private_key, "~/.ssh/wombat"}.
2 {set, wo_orch, public_key, "~/.ssh/wombat.pub"}.
3 {set, wo_orch, cloud_prefix, "joe-smith-"}.
```

When using physical providers, make sure that an SSH server is running on them, and that the provided private key has been added to its list of authorized keys. This includes the scenario when you deploy nodes on the same machine on which WombatOAM is running.
Using WombatOAM Orchestration

If you are interested in using WombatOAM Orchestration, it is recommended that you contact the WombatOAM team.
RESTful API

The WombatOAM Web Dashboard communicates via the WombatOAM server via its RESTful API (or REST interface) and WebSocket. Each service has a different URL prefix, e.g. the topology service is served under the `<wombat_server>/api/topo/...` URL.

WombatOAM is shipped with the ability to restrict access to the services by enforcing authentication. The authentication can be switched to turn off in the configuration file. If the authentication is turned on, this is the default, each request must contain valid credentials to be processed, otherwise HTTP 401 code is sent as response. Otherwise credentials are not required to be sent, although the server accepts non-existing credentials as well.

The REST interface of each service is documented on a separate page:

- Authentication (including user management)
- Role based information retrieval
- Topology
- Node information
- Alarms
- Notifications
- Metrics
- Services
- Orchestration
- Bookmarks
- Front page
- Plugins
- Other parts
Authentication and user management
REST interface

- Get the authentication status
- Get username of the current user
- Get websocket pass of the current user
- Get whether current user is admin
- Get role of the current user
- Get rights of the current user
- Set new password for the current user
- Set role of the current user
- Set rights of the current user
- Get the list of registered users
- Add new user
- Delete user

Get the authentication status

Retrieve whether authentication is turned on.

Definition

GET /api/users/auth_status

Example request

```bash
curl -X GET "http://127.0.0.1:8080/api/users/auth_status" \\
-u admin:admin
```

Example response

```
true
```

Returns

Boolean value describes the status of the authentication.

Get username of the current user

Retrieve the username of the current user.

Definition

GET /api/users/username

Example request

```bash
curl -X GET "http://127.0.0.1:8080/api/users/username" \\
-u admin:admin
```

Example response

```
"admin"
```

Returns

String representation of the username.
Get WebSocket pass of the current user
Retrieve the pass used to authenticate the current user via WebSocket.

Definition
GET /api/users/pass

Example request
curl -X GET "http://127.0.0.1:8080/api/users/pass" \\n  -u admin:admin

Example response
"7f9449854c2fef8d14de44f84a753c49"

Returns
String representation of the WebSocket pass.

Get whether current user is admin
Retrieve whether the current user is admin (only when authentication is enabled).

Definition
GET /api/users/is_admin

Example request
curl -X GET "http://127.0.0.1:8080/api/users/is_admin" \\n  -u admin:admin

Example response
true

Returns
Return true if authentication is enabled and the current user is admin. Otherwise return false.

Get role of the current user
Retrieve the role of the current user. This can be one of the following: guest, user, admin.

Definition
GET /api/users/role/USERNAME

Example request
curl -X GET "http://127.0.0.1:8080/api/users/role/admin" \\n  -u admin:admin

Example response
admin
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Required. The string value of the username.</td>
</tr>
</tbody>
</table>

Returns

If successful, the response is the role of the user.

Limitations

Request is restricted to admin and the specified user. If a non-admin user sends this request with another username then HTTP 400 is sent with an error object:

```
1 {"error" : "ERROR_REASON_STR"}
```

Get rights of the current user

Retrieve a list of rights. These can be read, write or execute.

Definition

GET /api/users/rights/USERNAME

Example request

```
curl -X GET "http://127.0.0.1:8080/api/users/rights/admin" \ 
-u admin:admin
```

Example response

```
[]
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Required. The string value of the username.</td>
</tr>
</tbody>
</table>

Returns

If successful, the response is a list of rights belongs to the user specified in the request.

Limitations

Request is restricted to admin and the specified user. If a non-admin user sends this request with another username then HTTP 400 is sent with an error object:

```
1 {"error" : "ERROR_REASON_STR"}
```

Set new password for the current user

Change the password of the current user.

Definition
POST /api/users/change_password

Data

```json
{
  "password": "PASSWORD"
}
```

Example request

```
curl -X POST "http://127.0.0.1:8080/api/users/change_password"
  -H "Content-Type: application/json"
  -d "{\"password\":\"newPassword12\"}"
  -u admin:admin
```

Example response

```
true
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>Required. The string value of the new password.</td>
</tr>
</tbody>
</table>

Returns

If successful, the HTTP response is 200 OK. The new password is stored. Otherwise, HTTP 400 is sent with an error object:

```
{"error": "ERROR_REASON_STR"}
```

Limitations

Password can consist of letters and numbers.

Set role of the current user

Set the role of the specified user. Request is restricted to admin.

Definition

POST /api/users/change_role

Data

```json
{
  "username": "USERNAME",
  "role": "ROLE"
}
```

Example request

```
curl -X POST "http://127.0.0.1:8080/api/users/change_role"
  -H "Content-Type: application/json"
  -d "{\"username\":\"admin\",
       \"role\":\"admin\"}"
  -u admin:admin
```
Example response

true

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Required. The string value of the username.</td>
</tr>
<tr>
<td>role</td>
<td>Required. The string value of the role.</td>
</tr>
</tbody>
</table>

Returns

If successful, the HTTP response is 200 OK. Otherwise, HTTP 400 is sent with an error object:

```
{ "error" : "ERROR_REASON_STR" }
```

Limitations

Request is restricted to admin. If other user sends this request then HTTP 400 is sent with an error object:

```
{ "error" : "ERROR_REASON_STR" }
```

Set rights of the current user

Set the rights of the specified user. Request is restricted to admin.

Definition

```
POST /api/users/change_rights
```

Data

```
{
  "username" : "USERNAME",
  "rights" : ["RIGHT"]
}
```

Example request

```
curl -X POST "http://127.0.0.1:8080/api/users/change_rights" \
   -H "Content-Type: application/json" \
   -d "{\"username\":\"admin\",\n        \"rights\":[]}" \
   -u admin:admin
```

Example response

true

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Required. The string value of the username.</td>
</tr>
<tr>
<td>rights</td>
<td>Required. A list of rights as string values.</td>
</tr>
</tbody>
</table>
Returns
If successful, the HTTP response is 200 OK. Otherwise, HTTP 400 is sent with an error object:

```json
{"error": "ERROR_REASON_STR"}
```

Limitations
Request is restricted to admin. If another user sends this request then HTTP 400 is sent with an error object:

```json
{"error": "ERROR_REASON_STR"}
```

Get the list of registered users
Retrieve a list of registered users.

**Definition**

```
GET /api/users/list_users
```

**Example request**

```bash
curl -X GET "http://127.0.0.1:8080/api/users/list_users" \
-u admin:admin
```

**Example response**

```json
["admin","user2","user"]
```

Returns
The array of the registered users's username. Usernames are represented as strings.

Limitations
Request is restricted to admin. If other user sends this request then HTTP 400 is sent with an error object:

```json
{"error": "ERROR_REASON_STR"}
```

Add new user
Add a new user with the specified credentials.

**Definition**

```
POST /api/users/add_user
```

**Data**

```json
{"username": "USERNAME" "password": "PASSWORD"}
```
Example request

```bash
curl -X POST "http://127.0.0.1:8080/api/users/add_user" \
-H "Content-Type: application/json" \
-d "{"password":"newPassword12", \
  "username":"newUser"}" \
-u admin:admin
```

Example response

`true`

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>username</code></td>
<td>Required. The string value of the username.</td>
</tr>
<tr>
<td><code>password</code></td>
<td>Required. The string value of the password.</td>
</tr>
</tbody>
</table>

Returns

If successful, the HTTP response is 200 OK. The new user is stored with the specified credentials. Otherwise, HTTP 400 is sent with an error object:

```json
{"error": "ERROR_REASON_STR"}
```

Limitations

Username and password can consist of letters and numbers.

Request is restricted to admin. If other user sends this request then HTTP 400 is sent with an error object:

```json
{"error": "ERROR_REASON_STR"}
```

Delete user

Delete the specified user.

**Definition**

DELETE /api/users/delete_user/USERNAME

or

DELETE /api/users/delete_user/USERNAME?with_pages=true

**Data**

none

**Example request**

```bash
curl -X DELETE "http://127.0.0.1:8080/api/users/delete_user/newUser?with_pages=true" \
-u admin:admin
```
## Example response

```
true
```

## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>username</code></td>
<td>Required. The string value of the username.</td>
</tr>
<tr>
<td><code>with_pages</code></td>
<td>Optional. If defined and the value is true then the front pages owned by the user will be deleted. Otherwise, the new owner will be the admin user.</td>
</tr>
</tbody>
</table>

## Returns

If successful, the HTTP response is 200 OK. The user is deleted. Otherwise, HTTP 400 is sent with an error object:

```
{
  "error": "ERROR_REASON_STR"
}
```

## Limitations

Username can consist of letters and numbers.

Request is restricted to admin. If other user sends this request then HTTP 400 is sent with an error object:

```
{
  "error": "ERROR_REASON_STR"
}
```
Tags REST interface

GET requests for getting information about tags:

- Get all available tags
- Get the active tags set for a user

POST requests to change settings:

- Change the tags set for a user

**Note** that the tags assigned to users are used only if authentication is switched on. When authentication is disabled the tags defined by the default_user_tags configuration parameter are used by the REST layer.

Get all available tags

**Definition**

GET `/api/tags`

**Example request**

```bash
curl -X GET "http://127.0.0.1:8080/api/tags"
```

**Example response**

```
["CustomTag", "alarm", "dev", "metric", "op"]
```

**Returns**

A minimal list of known tags that can be assigned to users or be used to reload only the current view on the WombatOAM Dashboard. It excludes object attributes, such as alarm IDs, but includes custom tags, data type tags (e.g. "alarm" and "metric") and the tags announced by the plugins, which are "dev" and "op" by default.

Get the active tags set for a user

**Definition**

GET `/api/tags/user/USERNAME`

**Example request**

```bash
curl -X GET "http://127.0.0.1:8080/api/tags/user/admin"
```

**Example response**

```
["dev", "op"]
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>The username of the user. If it is not given, the username of the current user will be used.</td>
</tr>
</tbody>
</table>

**Returns**
If the user exists either the list of tags set for the given user, or the "all" string is returned. The latter means no filtering is switched on for the user. In case of an error (e.g. the user doesn't exist), the {error, REASON_STR} object is returned.

Change the tags set for a user

Definition

POST /api/tags/user

Example request

curl -X POST "http://127.0.0.1:8080/api/tags/user" \
-H "Content-Type: application/json" \
-d '{"UserName": "admin", "tags": ["dev", "op"]}'

Example response

```
1    "ok"
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserName</td>
<td>Required. The username of the user.</td>
</tr>
<tr>
<td>tags</td>
<td>Required. Either the list of tags that should be set for the given user, or the &quot;all&quot; string to switch filtering off.</td>
</tr>
</tbody>
</table>

Returns

Either an "ok" string or the {error, REASON_STR} object that describes the reason of the failure.
**Topology REST interface**

- Get metadata about nodes
- Get metadata about node families
- Get more information about a node
- Get more information about a family
- Add new node
- Add new node and discover connected nodes
- Add new family
- Remove node
- Remove family
- Move a node to another family
- Analyse reachability of a node
- Get node statistics

**Get metadata about nodes**

Retrieve information about all the nodes managed in WombatOAM.

**Definition**

```
GET /api/topo/node
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/topo/node"
```

**Example response**

```
[
  {
    "id": "298548c9-f5f8-464d-9a2c-07d494c3ecd4",
    "name": "wombat@127.0.0.1",
    "node_family_id": "01a85d4a-7af3-43bd-b96e-21f0ffbc2e3",
    "host": "undefined",
    "cookie": "wombat",
    "domain_id": "undefined",
    "state": "UP",
    "tref": "undefined",
    "deleted": false,
    "plugins_opts": []
  },
  {...},
  {...}
]
```

**Arguments**

None.

**Returns**

An array of node objects.

**Get metadata about node families**

Retrieve information about all the managed families of nodes in WombatOAM.

**Definition**

```
GET /api/topo/family
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/topo/family"
```
GET /api/topo/node-family

Example request

curl -X GET "http://127.0.0.1:8080/api/topo/node-family"

Example response

```
[
  {
    "id": "823f293e-ba72-43b0-a4cf-e8be08d68f22",
    "name": "OTP APN 181 01 R16003.1",
    "description": [],
    "node_selection": "random",
    "bootstrap_strategy": [{"wo_bootstrap_strategy": "none"}, {}],
    "bootstrap_strategy_opts": [],
    "neighbors": [],
    "nodes": [
      "95b12f74-0c45-4b38-827f-59db5e88e56c",
      "f3e198ae-aace-4755-a8c1-68dc9610a904",
      "bf1641d1-b50c-4bb0-8c1e-387713b9323f",
      "19c61bb9-9f32-4c29-9e87-cbbac74ba489"
    ],
    "plugins_opts": [],
    "deleted": false,
  },
  ...
]
```

Arguments

None.

Returns

An array of node family objects.

Get more information about a node

Definition

GET /api/topo/node/NODE_ID

Example request

curl -X GET "http://10.211.55.14:8080/api/topo/node/298548c9-f5f8-464d-9a2c-07d494c3ecd4"

Example response

```
{
  "id": "298548c9-f5f8-464d-9a2c-07d494c3ecd4",
  "name": "wombat@127.0.0.1",
  "node_family_id": "01a85d4a-7af3-43bd-b96e-21f0ffbc2e3b",
  "host": "undefined",
  "cookie": "wombat",
  "domain_id": "undefined",
  "state": "UP",
  "tref": "undefined",
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node to be retrieved.</td>
</tr>
</tbody>
</table>

Returns

A node object if a valid identifier was provided. When requesting the identifier of a node that has been deleted, the "deleted" property will be true.

Get more information about a family

Retrieve details of a specified node family.

Definition

GET /api/topo/node-family/NODE_FAMILY_ID

Example request

curl -X GET "http://10.211.55.14:8080/api/topo/node-family/01a85d4a-7af3-43bd-b96e-21f0ffbc2e3b"

Example response

```
{
    "id": "01a85d4a-7af3-43bd-b96e-21f0ffbc2e3b",
    "name": "wombat pre-0.9.0",
    "description": [],
    "node_selection": "random",
    "bootstrap_strategy": [
        "no_bootstrap_strategy",
        "none",
        []
    ],
    "bootstrap_strategy_opts": [],
    "neighbors": [],
    "nodes": [
        "298548c9-f5f8-464d-9a2c-07d494c3ecd4"
    ],
    "plugins_opts": [],
    "deleted": false
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family ID</td>
<td>Required. The identifier of the node family to be retrieved.</td>
</tr>
</tbody>
</table>

Returns

A node family object. When requesting the identifier of a node family that has been deleted, the "deleted" property will be true.

Add new node
Add a managed node.

**Definition**

`POST /api/topo/action/add-node`

**Data**

```
{
  "name": "NEW_NODE_NAME",
  "cookie": "NEW_NODE_COOKIE"
}
```

**Example request**

```
curl -X POST "http://127.0.0.1:8080/api/topo/action/add-node" \
   -H "Content-Type: application/json" \
   -d "{"name":"wo_test_1@127.0.0.1", "cookie":"cookietest"}"
```

**Example response**

```
{
  "id": "c2d750f5-b4eb-40d6-b15c-9226577c0c4d",
  "name": "wo_test_1@127.0.0.1",
  "node_family_id": "823f293e-ba72-43b0-a4cf-e8be08d68f22",
  "host": "undefined",
  "cookie": "cookietest",
  "domain_id": "undefined",
  "state": "STOPPED",
  "tref": "undefined",
  "deleted": false,
  "plugins_opts": []
}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><strong>Required.</strong> The name of the node to be added.</td>
</tr>
<tr>
<td>cookie</td>
<td><strong>Required.</strong> The cookie for the node to be added.</td>
</tr>
</tbody>
</table>

**Returns**

The node object. Note that, right after adding the node, the "state" may be STOPPED or DOWN.

**Add new node and discover connected nodes**

Add a new managed node, and discover and automatically add other nodes connected to it.

**Definition**

`POST /api/topo/action/add-node-and-disc`

**Data**

```
{
  "name": "NEW_NODE_NAME",
  "cookie": "NEW_NODE_COOKIE"
}
```
Example request

curl -X POST "http://127.0.0.1:8080/api/topo/action/add-node-and-disc" \
-H "Content-Type: application/json" \
-d "{"name": "wo_test_1@127.0.0.1", "cookie": "cookietest"}"

Example response

```
[
  {
    "id": "a2df05ae-75ba-49aa-a993-b3cfd63a120",
    "name": "wo_test_3@127.0.0.1",
    "node_family_id": "93be4f01-dae6-49b0-9608-4552f841a720",
    "host": "undefined",
    "cookie": "cookietest",
    "domain_id": "undefined",
    "state": "STOPPED",
    "tref": "undefined",
    "deleted": false,
    "plugins_opts": []
  },
  {}
]
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><em>Required</em>. The name of the node to be added.</td>
</tr>
<tr>
<td>cookie</td>
<td><em>Required</em>. The cookie for the node to be added.</td>
</tr>
</tbody>
</table>

Returns

The node objects for the discovered nodes. Note that, right after adding the nodes, the "state" may be STOPPED or DOWN.

Add new family

Add a node family.

Definition

POST /api/topo/node-family

Data

```
{
  "name": "NEW_FAMILY_NAME",
  "bootstrap_node_selection": "SELECTION",
  "bootstrap_strategy": "STRATEGY",
  "bootstrap_strategy_opts": []
}
```

Example request

curl -X POST "http://127.0.0.1:8080/api/topo/node-family" \
-H "Content-Type: application/json" \
-d "{"name": "NEW_FAMILY_NAME",
    "bootstrap_node_selection": "SELECTION",
    "bootstrap_strategy": "STRATEGY",
    "bootstrap_strategy_opts": []}"

Example response

```json
{
  "id": "0db17632-5264-48e0-9282-7cb2db3983d6",
  "name": "new_family_name",
  "description": [],
  "node_selection": "selection",
  "bootstrap_strategy": "strategy",
  "bootstrap_strategy_opts": [],
  "neighbors": [],
  "nodes": [],
  "plugins_opts": [],
  "deleted": false
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><strong>Required.</strong> A custom string with which to identify the node family.</td>
</tr>
<tr>
<td>bootstrap_node_selection</td>
<td><strong>Required.</strong> Defines how another node is selected when a new node joins the cluster. Currently the only option is random.</td>
</tr>
<tr>
<td>bootstrap_strategy</td>
<td><strong>Required.</strong> Determines what WombatOAM should do when it deploys a new node to the node family. The options are Distributed Erlang (WombatOAM only makes sure that the nodes are connected with each other via Erlang Distribution), and Custom (you can specify a script to be executed in bootstrap_strategy_opts).</td>
</tr>
<tr>
<td>bootstrap_strategy_opts</td>
<td><strong>Required.</strong> A list containing details of what should happen when a new node is deployed into the node family, for example the execution of a specified script. The list may be empty.</td>
</tr>
</tbody>
</table>

Returns

A node family object.

Remove node

Remove a specified node from being managed in WombatOAM.

Definition

DELETE /api/topo/node/NODE_ID

Example request

```bash
curl -X DELETE "http://127.0.0.1:8080/api/topo/node/a2df05ae-75ba-49aa-a993-b3cfc63a12"
```
Example response
204 No Content

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family ID</td>
<td>Required. The identifier of the node to be removed.</td>
</tr>
</tbody>
</table>

Returns
Only the HTTP response code.

Remove family
Remove a specified node family from being managed in WombatOAM.

Definition
DELETE /api/topo/node-family/NODE_FAMILY_ID/force

Example request

curl -X DELETE "http://127.0.0.1:8080/api/topo/node_families/823f293e-ba72-43b0-a4cf-e8be08d68f22/force"

Example response
204 No Content

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family ID</td>
<td>Required. The identifier of the node family to be removed.</td>
</tr>
</tbody>
</table>

Returns
Only the HTTP response code.

Move a node to another family
Move a specified node to a specified node family.

Definition
POST /api/topo/action/move-node

Example request 1

curl -X POST "http://127.0.0.1:8080/api/topo/action/move-node" \
   -H "Content-Type: application/json" \ 
   -d "{"node":"d1a4c84f-754a-4d44-9ba2-4932faf65004", \ 
"family":"33c14e1f-5e56-415f-9b2c-32ff9ed653ff"}"

Example response 1
Example request 2

curl -X POST "http://127.0.0.1:8080/api/topo/action/move-node" \
   -H "Content-Type: application/json" \
   -d "{"node":"d35345f-754a-4d44-9ba2-4932fgvi73y", "family":"334533f-5e56-415f-9b2c-32ff9vjhebff"}"

Example response 2

```
{
    "success": false,
    "reason": "node_or_family_not_found"
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node.</td>
</tr>
<tr>
<td>node family ID</td>
<td>Required. The identifier of the node family to move the node to.</td>
</tr>
</tbody>
</table>

Returns

returns an object showing the success or failure.

Analyse reachability of a node

Analyse the possibility of reaching the node via Distributed Erlang.

Definition

POST /api/topo/action/analyse-reachability

Data

```
{
    "name": "NODE_NAME"
}
```

Example request 1

curl -X POST "http://127.0.0.1:8080/api/topo/action/analyse-reachability" \
   -H "Content-Type: application/json" \
   -d "{"name":"wo_test_1@127.0.0.1"}"

Example response 1

```
{
    "reachability": "should be reachable",
    "details": [
        "checksPerformed": [
            "check_node_format",
            "check_name_type",
            "check_host_resolution",
```
Example request 2

curl -X POST "http://127.0.0.1:8080/api/topo/action/analyse-reachability" \
-H "Content-Type: application/json" \
-d 

Example response 2

{  "reachability": "is not reachable",  "details": [  {    "reasonCode": "bad_node_format"  },  {    "reasonDetails": "[\{node,name@\}]"  }  ],  "description": "The node name specified name@ does not comply with the name@host format.",  "hints": [    "Specify a node name compliant to the name@host format."  ]}

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Required. The name of the node to be analysed.</td>
</tr>
</tbody>
</table>

Returns

An object describing the reachability of the node, including the checks performed, the reason why the node might not be reachable, (if relevant), and steps the user could take (under “hints”) concerning the management of the node. The fields `number_of_crashlogs_this_hour` and `number_of_new_alarms_raised_this_hour` show the number updates on those counters between the beginning of the current hour and the current time.

Get node statistics

Return statistical information about the entire cluster.

Definition

GET /api/topo/node-stat
Example request

curl -X GET "http://10.211.55.14:8080/api/topo/node-stat"

Example response

```json
{
  "number_of_nodes": 2,
  "number_of_up_nodes": 2,
  "number_of_families": 1,
  "number_of_crashlogs_this_hour": 22,
  "number_of_new_alarms_raised_this_hour": 57,
  "number_of_active_alarms": 0,
  "family_stats": [
    {
      "id": "c0dce081-89ae-458d-8465-0533a29c4037",
      "number_of_nodes": 2,
      "number_of_up_nodes": 2
    }
  ]
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>

Returns

Returns information about the managed nodes, containing information about the number of nodes, running nodes, managed node families, number of alarms and number of crash logs in the last one hour.
Node information REST interface

- Get system information about a node
- Get application information about a node
- Get module information about a node

Get system information about a node

Retrieve information about the software and hardware environment of the Erlang node including operating system and Erlang versions, processor bit size, etc.

Definition

GET /api/node-infos/node/NODE_ID/node-info

Example request

curl -X GET "http://127.0.0.1:8080/api/node-infos/node/956549c6-89bb-4ee0-a34b-c5a772ef78fa/node-info"

Example response

```
{
    "os_type": "unix",
    "os_descr": "Debian GNU/Linux 8.7 (jessie)",
    "kernel_version": "3.16.0",
    "n.cores": "2",
    "otp_version": "17",
    "system_version": "Erlang/OTP 17 [erts-6.2] [source] [64-bit] [smp:2:2] [async-threads:30] [kernel-poll:true]\\n",
    "ets_limit": "2053",
    "process_limit": "1048576",
    "port_limit": "65536",
    "driver_version": "3.1",
    "machine": "BEAM",
    "smp_support": "true",
    "wordsize": "64",
    "compat_rel": "17",
    "hostname": "debian",
    "ip": "127.0.1.1",
    "architecture": "x86_64-pc-linux-gnu",
    "release_name": "Erlang/OTP",
    "release_version": "17"
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node to be retrieved.</td>
</tr>
</tbody>
</table>

Returns

A nodeInfo object if a valid identifier was provided. The values of the attributes are strings.

Get application information about a node

Retrieve information about the applications loaded on the Erlang node.

Definition

GET /api/node-infos/node/NODE_ID/node-app-info
Example request

curl -X GET "http://127.0.0.1:8080/api/node-infos/node/956549c6-89bb-4ee0-a34b-c5a772ef78fa/node-app-info"

Example response

```
{
    "name": "kernel",
    "description": "ERTS  CXC 138 10",
    "version": "3.0.3",
    "running": "<0.9.0>",
    "type": "permanent",
    "config": {
        "name": "inet_dist_listen_min",
        "value": "25672"
    },
    "modules": "application, application_controller, application_master, application_starter, auth, code, code_server, disk_log, disk_log_1, disk_log_server, disk_log_sup, dist_ac, dist_util, erl_boot_server, erl_dll, erl_distribution, erl_epmd, erl_reply, error_handler, error_logger, erts_debug, file, file_io_server, file_server, gen_scct, gen_tcp, gen_udp, global, global_group, global_search, group, heartbeat, hipe_unified_loader, inet, inet6_scct, inet6_tcp, inet6_tcp_dist, inet6_udp, inet_config, inet_db, inet_dns, inet_gethost_native, inet_hosts, inet_parse, inet_res, inet_scct, inet_tcp, inet_tcp_dist, inet_udp, kernel, kernel_config, net, net_admin, net_kernel, os, pg2, ram_file, rpc, seq_trace, standard_error, user, user_drv, user_sup, wrap_log_reader"
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node to be retrieved.</td>
</tr>
</tbody>
</table>

Returns

An array of nodeAppInfo objects if a valid identifier was provided. There's one nodeAppInfo object for each application. The possible values of the type attribute are

- permanent: see application:start/2 in Erlang/OTP documentation
- transient: see application:start/2 in Erlang/OTP documentation
- temporary: see application:start/2 in Erlang/OTP documentation
- unknown: the application is loaded, but not started
- Error: application running, but not started!

The possible values for the running attribute are

- the process identifier (PID) of the application master process in the application
- library: if the application is started, but has no master process (e.g. stdlib doesn't have one)
- Loaded, but not started: the application is loaded, but not started
- crashed: the application is started, but is not running
- unknown: the application is not started but is running and there's no application master process for the application
The `modules` attribute contains the list of Erlang modules in this application. The `config` attribute contains the application environment variables. The name, description and version attributes are strings.

**Get module information about a node**

Retrieve information about the modules loaded on the Erlang node.

**Definition**

GET `/api/node-infos/node/NODE_ID/module-info`

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/node-infos/node/956549c6-89bb-4ee0-a34b-c5a772ef78fa/module-info"
```

**Example response**

```
[
  {
    "module": "lists",
    "md5": "5efb7bde5ffcf963142304f7d0b648d",
    "vsn": "[126253210273804497200583865567816672397]",
    "cmp_version": "\"5.0\"",
    "time": 1428156234000,
    "source": "/tmp/buildd/erlang-17.3-dfs/lib/stdlib/src/lists.erl",
    "native": "false",
    "beam_location": "/usr/lib/erlang/lib/stdlib-2.2/ebin/lists.beam",
    "sticky": "true",
    "application": "stdlib",
    "binary_size": 2980,
    "exported_funcs": [
      {
        "name": "module_info",
        "arity": 1
      }
    ],
    "compile_info": {
      "outdir": "/Users/user/.kerl/builds/17.5/otp_src_17.5/lib/kernel/ebin",
      "include_paths": [
        "/Users/user/.kerl/builds/17.5/otp_src_17.5/lib/stdlib/include",
      ],
      "options": [
        "warnings_as_errors",
        "debug_info",
        
        "[parse_transform,lager_transform]
      ],
      "defines": [
        {
          "key": "otp_17",
          "value": "true"
        }
      ]
    }
  }
]
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node to be retrieved.</td>
</tr>
</tbody>
</table>

**Returns**

An array of module objects if a valid identifier was provided. There's one
module object for each Erlang module.

- The `time` attribute contains the time when the module was compiled, in milliseconds since 1970-01-01 00:00:00.
- `cmp_version` is the version of the Erlang compiler application used to compile this module.
- `vsn` is the version of the Erlang module (see `beam_lib:version/1` in Erlang/OTP documentation).
- `md5` is the MD5 checksum of the module code (see `beam_lib:md5/1` in Erlang/OTP documentation).
- `source` is the full path of the source file used to compile this module.
- `beam_location` is the full path of the compiled object file.
- `native` is "true" if the module has native compiled code, "false" otherwise.
- `sticky` is "true" if module that has been loaded from a sticky directory, "false" otherwise.
- `application` is the name of the application that contains this module or undefined if the module doesn't belong to any application.
- `binary_size` the size of the compiled binary (.beam) file in bytes
- `exported_funcs` the module's exported function in an array containing objects whith keys `name` and `arity`
- `compile_info` is an object containing compile time information
  - `outdir`: the compiled object's directory
  - `include_paths`: array of paths which were used as include paths during compilation
  - `options`: array of strings containing compile time options
  - `defines`: an array of objects having `key` and `value` properties containing the macros defined at compile time

Get the metric limits for a node

Retrieve all the available metric limits of the Erlang node.

**Definition**

```
GET /api/node-infos/node/NODE_ID/metric-limits
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/node-infos/node/956549c6-89bb-4ee0-a34b-c5a772ef70fa/metric-limits"
```

**Example response**

```
{
    "atom_count": "1048576",
    "cpu_avg_1": "8",
    "ets_count": "2053",
    "open_ports": "65536",
    "process_count": "262144"
}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>node ID</strong></td>
<td><strong>Required.</strong> The identifier of the node to be retrieved.</td>
</tr>
</tbody>
</table>
Returns
An array of tuples with every metric associated to its limit value.
Alarms REST interface

- Get all alarms
- Get all alarms from a specified ID
- Get alarms for a node
- Get alarms for a node from a specified ID
- Get alarms for a node family
- Get alarms for a node family from a specified ID
- Get all alarm sources
- Get alarm sources for a node
- Get alarm sources for a node family
- Send acknowledge for an alarm
- Send unacknowledge for an alarm
- Send clear for an alarm
- Send comment for an alarm

Get all alarms

Retrieve the 20 most recent alarms.

Definition

GET /api/alarm?source=SOURCE&limit=LIMIT&tags=LISTOFTAGS

Example request

curl -X GET "http://127.0.0.1:8080/api/alarm?tags=dev,op"

Example response

```json
{
  "alarms": [
    {
      "alarmId": "system_memory_high_watermark",
      "alarmType": null,
      "src": {
        "nodeId": "3fa56f92-3195-401b-93a8-7bc43e54d66f",
        "nodeName": "wo_test_1@127.0.0.1",
        "custom": "node_level",
        "alarmServer": "wombat"
      },
      "eventTime": 1406873289000,
      "eventId": "{1406,873289,745642}",
      "severity": "indeterminate",
      "probableCause": "",
      "proposedRepairAction": "",
      "description": "",
      "additionalInformation": ""Hi there! all O",
      "correlatedEvents": "[]",
      "comments": "",
      "trend": null,
      "threshold": null,
      "state": "new",
      "ackInfo": null,
      "rawSrc": "wo_alarm_source,3fa56f92-3195-401b-93a8-7bc43e54d66f",
      "wo_test_1@127.0.0.1"
    },
    ...
  ]
}
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>Optional. The src.custom field used for filtering alarms.</td>
</tr>
<tr>
<td>limit</td>
<td>Optional. Specifies the maximum number of alarms in the response. By default, it is 20.</td>
</tr>
<tr>
<td>tags</td>
<td>Optional. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
</tbody>
</table>

Returns

An array of alarm objects that are tagged with any of the active tags. The value of eventId can be used to retrieve details of specific alarms.

Get all alarms from a specified ID

Retrieve the 20 most recent alarms for a specified event.

Definition

GET /api/alarm/previous/all?eid=EVENT_ID&source=SOURCE&limit=LIMIT&tags=LISTOFTAGS

Example request

curl -X GET "http://127.0.0.1:8080/api/alarm/previous/all?eid=%7B1406,873289,745642%7D"

Example response

See "Get all alarms", above.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eid</td>
<td>Required. The event's identifier (including curly brackets).</td>
</tr>
<tr>
<td>source</td>
<td>Optional. The src.custom field used for filtering alarms.</td>
</tr>
<tr>
<td>limit</td>
<td>Optional. Specifies the maximum number of alarms in the response. By default, it is 20.</td>
</tr>
<tr>
<td>tags</td>
<td>Optional. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
</tbody>
</table>

Returns

See "Get all alarms", above.

Get alarms for a node

Retrieve the 20 most recent alarms for a specified node.

Definition

GET /api/alarm/node/NODE_ID?source=SOURCE&limit=LIMIT&tags=LISTOFTAGS

Example request

```
curl -X GET "http://127.0.0.1:8080/api/alarm/node/3fa56f92-3195-401b-93a8-7bc43e54d66f"
```

Example response

See "Get all alarms", above.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node id</td>
<td>Required. The node's identifier</td>
</tr>
<tr>
<td>source</td>
<td>Optional. The src.custom field used for filtering alarms.</td>
</tr>
<tr>
<td>limit</td>
<td>Optional. Specifies the maximum number of alarms in the response. By default, it is 20.</td>
</tr>
<tr>
<td>tags</td>
<td>Optional. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
</tbody>
</table>

Returns

See "Get all alarms", above.

Get alarms for a node from a specified ID

Retrieve the 20 most recent alarms for a specified event on a node.

Definition

```
GET /api/alarm/previous/node/NODE_ID?eid=EVENT_ID&source=SOURCE&limit=LIMIT&tags=LISTOFTAGS
```

Example request

```
curl -X GET "http://127.0.0.1:8080/api/alarm/previous/node/5b16ec55-56a3-4430-b174-c996897ccd14?eid=%7B1406,873289,745642%7D"
```

Example response

See "Get all alarms", above.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node id</td>
<td>Required. The node's identifier</td>
</tr>
<tr>
<td>eid</td>
<td>Required. The event's identifier (including curly brackets).</td>
</tr>
<tr>
<td>source</td>
<td>Optional. The src.custom field used for filtering alarms.</td>
</tr>
<tr>
<td>limit</td>
<td>Optional. Specifies the maximum number of alarms in the response. By default, it is 20.</td>
</tr>
<tr>
<td>tags</td>
<td>Optional. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
</tbody>
</table>

Returns
Get alarms for a node family

Retrieve the 20 most recent alarms for all the nodes in a specified node family.

**Definition**

```
GET /api/alarm/node-family/NODE_FAMILY_ID?source=SOURCE&limit=LIMIT&tags=LISTOFTAGS
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/alarm/node-family/93be4f01-dae6-49b0-9608-4552f841a720"
```

**Example response**

See "Get all alarms", above.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family id</td>
<td>Required. The node family's identifier</td>
</tr>
<tr>
<td>source</td>
<td>Optional. The src.custom field used for filtering alarms.</td>
</tr>
<tr>
<td>limit</td>
<td>Optional. Specifies the maximum number of alarms in the response. By default, it is 20.</td>
</tr>
<tr>
<td>tags</td>
<td>Optional. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
</tbody>
</table>

**Returns**

See "Get all alarms", above.

Get alarms for a node family from a specified ID

Retrieve the 20 most recent alarms for a specified event on the nodes in a node family.

**Definition**

```
GET /api/alarm/previous/node-family/NODE_FAMILY_ID?eid=EVENT_ID &source=SOURCE&limit=LIMIT&tags=LISTOFTAGS
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/alarm/previous/node-family/93be4f01-dae6-49b0-9608-4552f841a720?eid=%7B1406,873289,745642%7D"
```

**Example response**

See "Get all alarms", above.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event id</td>
<td>Optional. The event id that the alarms are for.</td>
</tr>
<tr>
<td>source</td>
<td>Optional. The src.custom field used for filtering alarms.</td>
</tr>
<tr>
<td>limit</td>
<td>Optional. Specifies the maximum number of alarms in the response. By default, it is 20.</td>
</tr>
<tr>
<td>tags</td>
<td>Optional. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
</tbody>
</table>
**Returns**

See "Get all alarms", above.

### Get all alarm sources

Retrieve all alarm sources that can be used for filtering alarms.

**Definition**

```
GET /api/alarm/source
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/alarm/source"
```

**Example response**

```
{
"sources": [
  "node_level"
]
}
```

**Arguments**

None.

**Returns**

An array of alarm sources. These are the same values that may turn up in the `src.custom` field of alarm objects.

**Limitations**

Except for the default "node_level" source in the current version of WombatOAM alarm sources are not returned if they don't have any active alarms.

### Get alarm sources for a node

Retrieve alarm sources for a node that can be used for filtering alarms.

**Definition**

```
GET /api/alarm/source/node/NODE_ID
```

**Example request**
Example response
See "Get all alarm sources" above.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node id</td>
<td>Required. The node's identifier</td>
</tr>
</tbody>
</table>

Returns
See "Get all alarm sources" above.

Limitations
See "Get all alarm sources" above.

Get alarm sources for a node family
Retrieve alarm sources for a node family that can be used for filtering alarms.

Definition

GET /api/alarm/source/node-family/NODE_FAMILY_ID

Example request

```
curl -X GET "http://127.0.0.1:8080/api/alarm/source/node-family/93be4f01-dae6-49b0-9608-4552f841a720"
```

Example response
See "Get all alarm sources" above.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family id</td>
<td>Required. The node family's identifier</td>
</tr>
</tbody>
</table>

Returns
See "Get all alarm sources" above.

Limitations
See "Get all alarm sources" above.

Send acknowledge for an alarm
Provide acknowledgement of an alarm.

Definition

POST /api/alarm/acknowledge

Data
Example request

```bash
curl -X POST "http://127.0.0.1:8080/api/alarm/acknowledge" \
-H "Content-Type: application/json" \
-d "{"userId":"administrator", "alarmId":"system_memory_high_watermark", "alarmSrc":{"wo_alarm_source,\"5b16ec55-56a3-4430-b174-c996897ccd14\",'wo_test_2@127.0.0.1',\nnode_level,wombat}"}
```

Example response

```json
{}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td><strong>Required.</strong> The identifier of the user sending the acknowledgement.</td>
</tr>
<tr>
<td>alarmId</td>
<td><strong>Required.</strong> The value of alarmId in the alarm object.</td>
</tr>
<tr>
<td>alarmSrc</td>
<td><strong>Required.</strong> The value of rawSrc in the alarm object.</td>
</tr>
</tbody>
</table>

Returns

An empty object. If successful, the HTTP response is 200 OK. In the alarm object, state changes to "acknowledged", and ackInfo is updated with user and time.

Send unacknowledge for an alarm

Remove the acknowledgement for an alarm, so that it is shown as new.

Definition

POST /api/alarm/unacknowledge

Data

```json
{
    "userId": "USER_ID",
    "alarmId": "ALARM_ID",
    "alarmSrc": "RAW_SRC"
}
```

Example request

```bash
curl -X POST "http://127.0.0.1:8080/api/alarm/unacknowledge" \
-H "Content-Type: application/json" \
-d "{"userId":"administrator1", "alarmId":"system_memory_high_watermark", "alarmSrc":{"wo_alarm_source,\"5b16ec55-56a3-4430-b174-c996897ccd14\",'wo_test_2@127.0.0.1',\nnode_level,wombat}"}
```

Example response

```json
{}
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>Required. The identifier of the user sending the acknowledgement</td>
</tr>
<tr>
<td>alarmId</td>
<td>Required. The value of alarmId in the alarm object.</td>
</tr>
<tr>
<td>alarmSrc</td>
<td>Required. The value of rawSrc in the alarm object.</td>
</tr>
</tbody>
</table>

Returns

An empty object. If successful, the HTTP response is 200 OK. In the alarm object, state changes to "new".

Send clear for an alarm

Clear an alarm, removing it from the list of alarms.

Definition

POST /api/alarm/clear

Data

```
{
    "userId": "USER_ID",
    "alarmId": "ALARM_ID",
    "alarmSrc": "RAW_SRC"
}
```

Example request

curl -X POST "http://127.0.0.1:8080/api/alarm/clear" \
-H "Content-Type: application/json" \
-d "{"userId":"administrator1","alarmId":"system_memory_high_watermark","alarmSrc":\"{wo_alarm_source,\"5b16c55-56a3-4430-b174-c996897ccd14\",\nnode_level,wombat\"\"}""

Example response

```
{}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>Required. The identifier of the user sending the acknowledgement</td>
</tr>
<tr>
<td>alarmId</td>
<td>Required. The value of alarmId in the alarm object.</td>
</tr>
<tr>
<td>alarmSrc</td>
<td>Required. The value of rawSrc in the alarm object.</td>
</tr>
</tbody>
</table>

Returns

An empty object. If successful, the HTTP response is 200 OK.
Send comment for an alarm

Add a comment to the details of a specified alarm.

**Definition**

POST /api/alarm/comment

**Data**

```json
{
    "userId": "USER_ID",
    "alarmId": "ALARM_ID",
    "alarmSrc": "RAW_SRC",
    "comment": "MESSAGE"
}
```

**Example request**

```
curl -X POST "http://127.0.0.1:8080/api/alarm/comment" \
    -H "Content-Type: application/json" \
    -d "{"userId": "administrator", "alarmId": "system_memory_high_watermark", "alarmSrc": "{wo_alarm_source,\"298548c9-f5f8-464d-9a2c-07d494c3ecd4\",\n    node_level,wombat},\"rawSrc\":\"298548c9-f5f8-464d-9a2c-07d494c3ecd4\",\n    \"comment\":\"Support has been notified.\"}"
```

**Example response**

```json
{}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userId</td>
<td>Required. The identifier of the user sending the acknowledgement.</td>
</tr>
<tr>
<td>alarmId</td>
<td>Required. The value of alarmId in the alarm object.</td>
</tr>
<tr>
<td>alarmSrc</td>
<td>Required. The value of rawSrc in the alarm object.</td>
</tr>
<tr>
<td>comment</td>
<td>Required. The text that will be included as a comment.</td>
</tr>
</tbody>
</table>

**Returns**

An empty object. In the alarm object, an item is added to comments, with the properties *user*, *time*, and *text*. 
Logs/notifications REST interface

- Get all logs
- Get 20 logs from a cursor position
- Get logs for a node
- Get 20 logs from a cursor position for a node
- Get logs for a node family
- Get 20 logs from a cursor position for a node family
- Search in logs
- Get 20 logs from a cursor position for the search query
- Get all log originators
- Get log originators for a node
- Get log originators for a node family
- Get log levels for a node
- Set log level for current node
- Get part of a log message

Get all logs

Retrieve the latest log entries for all managed nodes.

**Definition**

GET /api/logs?originator=ORIGINATOR&after_date=AFTER_DATE_AND_TIME&tags=LISTOFTAGS

**Example requests**

curl -X GET "http://127.0.0.1:8080/api/logs"
curl -X GET "http://127.0.0.1:8080/api/logs?originator=alarm"
curl -X GET "http://127.0.0.1:8080/api/logs?tags=dev,critical"

**Example response**

```
{
   "logs": [  
      {  
        "date": "1406817800759",
        "severity": "warning",
        "originator": "System monitor",
        "nodeId": "298548c9-f5f8-464d-9a2c-07d494c3ecd4",
        "nodeName": "wombat@127.0.0.1",
        "message": "Long schedule time <0.19954.0>\[{timeout, 307},\n        in,\{folsom_ets,notify,1\},\n        out,\{wo_metrics_graphite_ch,send_to_carbon,1\}\}],
        "type": "logEvent"
      },
      {  
        "date": "1406817800452",
        "severity": "warning",
        "originator": "System monitor",
        "nodeId": "298548c9-f5f8-464d-9a2c-07d494c3ecd4",
        "nodeName": "wombat@127.0.0.1",
        "message": "Long schedule time <0.19954.0>\[{timeout, 158},\n        in,\{folsom_metrics,notify,1\},\n        out,\{folsom_ets,notify,1\}\}],
        "type": "logEvent"
      },
   ]
}
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>originator</td>
<td>Optional. Filter the logs by this field's value.</td>
</tr>
<tr>
<td>originator[]</td>
<td>Optional. Multiple originators can be supplied through this parameter.</td>
</tr>
<tr>
<td>not_originator[]</td>
<td>Optional. The supplied originators will not be included in the returned logs.</td>
</tr>
<tr>
<td>after_date</td>
<td>Optional. Logs created before this date will be returned.</td>
</tr>
<tr>
<td>tags</td>
<td>Optional. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
<tr>
<td>limit</td>
<td>Optional. A positive integer specifying the maximum number of logs returned. The maximum is 100.</td>
</tr>
</tbody>
</table>

Returns

An array of the 20 or the limit amount of most recent log entries for all managed nodes, starting with the newest log entry. Each entry has the following properties: `date`, `severity`, `originator`, `nodeId`, `nodeName`, `message`, `props` and `type`. A cursor is provided that can be used to enumerate further log entries (unless it's null which indicates there are no more results). Possible properties when the full log message is bigger than 1000 bytes: - `hasNext`: there's enough data in the log message to show on next page - `hasPrev`: there's enough data in the log message to show on previous page - `start`: the index of the first byte currently shown. Use the Get part of a log message call to get the rest of the message. - `cursor`: contains the cursor of this message.

Get 20 logs from a cursor position

Retrieve older log entries for all managed nodes.

Definition

GET /api/logs/previous/all?c=CURSOR&originator=ORIGINATOR&after_date=AFTER_DATE_AND_TIME&tags=dev,critical

Example response

See "Get all logs", above.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor</td>
<td>Required. The cursor returned in the previous call.</td>
</tr>
<tr>
<td>originator</td>
<td>Optional. Filter the logs by this field's value.</td>
</tr>
<tr>
<td>originator[]</td>
<td>Optional. Multiple originators can be supplied</td>
</tr>
</tbody>
</table>
through this parameter.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>not_originator[]</td>
<td><strong>Optional.</strong> The supplied originators will not be included in the returned logs.</td>
</tr>
<tr>
<td>after_date</td>
<td><strong>Optional.</strong> Logs created before this date will be returned.</td>
</tr>
<tr>
<td>tags</td>
<td><strong>Optional.</strong> A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
<tr>
<td>limit</td>
<td><strong>Optional.</strong> A positive integer specifying the maximum number of logs returned. The maximum is 100.</td>
</tr>
</tbody>
</table>

**Returns**

An array of 20 or the limit amount of log entries, starting from the specified cursor position, for all managed nodes. The most recent entries appear first. See “Get all logs”, above, for more details.

**Get logs for a node**

Retrieve the latest log entries for a specified node.

**Definition**

```
GET /api/logs/node/NODE_ID?originator=ORIGINATOR&after_date=AFTER_DATE_AND_TIME&tags=dev,critical
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/logs/node/3fa56f92-3195-401b-93a8-7bc43e54d66f"
```

**Example response**

See “Get all logs”, above.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td><strong>Required.</strong> The node's identifier.</td>
</tr>
<tr>
<td>originator</td>
<td><strong>Optional.</strong> Filter the logs by this field's value.</td>
</tr>
<tr>
<td>originator[]</td>
<td><strong>Optional.</strong> Multiple originators can be supplied through this parameter.</td>
</tr>
<tr>
<td>not_originator[]</td>
<td><strong>Optional.</strong> The supplied originators will not be included in the returned logs.</td>
</tr>
<tr>
<td>after_date</td>
<td><strong>Optional.</strong> Logs created before this date will be returned.</td>
</tr>
<tr>
<td>tags</td>
<td><strong>Optional.</strong> A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
<tr>
<td>limit</td>
<td><strong>Optional.</strong> A positive integer specifying the maximum number of logs returned. The maximum is 100.</td>
</tr>
</tbody>
</table>

**Returns**

Log entries for the specified node and a cursor. See “Get all logs”, above,
for more details.

**Get 20 logs from a cursor position for a node**

Retrieve older log entries for a specified node.

**Definition**

```markdown
GET /api/logs/previous/node/NODE_ID?c=CURSOR&originator=ORIGINATOR&after_date=AFTER_DATE_AND_TIME&tags=dev,critical
```

**Example response**

See "Get all logs", above.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td><strong>Required.</strong> The node's identifier.</td>
</tr>
<tr>
<td>cursor</td>
<td><strong>Required.</strong> The cursor returned in the previous call.</td>
</tr>
<tr>
<td>originator</td>
<td><strong>Optional.</strong> Filter the logs by this field's value.</td>
</tr>
<tr>
<td>originator[]</td>
<td><strong>Optional.</strong> Multiple originators can be supplied through this parameter.</td>
</tr>
<tr>
<td>not_originator[]</td>
<td><strong>Optional.</strong> The supplied originators will not be included in the returned logs.</td>
</tr>
<tr>
<td>after_date</td>
<td><strong>Optional.</strong> Logs created before this date will be returned.</td>
</tr>
<tr>
<td>tags</td>
<td><strong>Optional.</strong> A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
<tr>
<td>limit</td>
<td><strong>Optional.</strong> A positive integer specifying the maximum number of logs returned. The maximum is 100.</td>
</tr>
</tbody>
</table>

**Returns**

An array of 20 or the limit amount of log entries, starting from the specified cursor position, for the specified node. The most recent entries appear first. See "Get all logs", above, for more details.

**Get logs for a node family**

Retrieve the latest log entries for all the managed nodes in a specified node family.

**Definition**

```markdown
GET /api/logs/node-family/NODE_FAMILY_ID?originator=ORIGINATOR&after_date=AFTER_DATE_AND_TIME&tags=dev,critical
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/logs/node-family/93be4f01-da6-49b0-9608-4552f841a720"
```

**Example response**
See "Get all logs", above.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family ID</td>
<td>Required. The node family's identifier.</td>
</tr>
<tr>
<td>originator</td>
<td>Optional. Filter the logs by this field's value.</td>
</tr>
<tr>
<td>originator[]</td>
<td>Optional. Multiple originators can be supplied through this parameter.</td>
</tr>
<tr>
<td>not_originator[]</td>
<td>Optional. The supplied originators will not be included in the returned logs.</td>
</tr>
<tr>
<td>after_date</td>
<td>Optional. Logs created before this date will be returned.</td>
</tr>
<tr>
<td>tags</td>
<td>Optional. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
<tr>
<td>limit</td>
<td>Optional. A positive integer specifying the maximum number of logs returned. The maximum is 100.</td>
</tr>
</tbody>
</table>

Returns

An array of 20 or the limit amount of log entries, starting from the specified cursor position, for the nodes in the specified node family. The most recent entries appear first. See "Get all logs", above, for more details.

Get 20 logs from a cursor position for a node family

Retrieve older log entries for all the managed nodes in a specified node family.

Definition

GET /api/logs/previous/node-family/NODE_FAMILY_ID?c=CURSOR&originator=ORIGINATOR&after_date=AFTER_DATE_AND_TIME&tags=dev,critical

Example response

See "Get all logs", above.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family ID</td>
<td>Required. The node family's identifier.</td>
</tr>
<tr>
<td>cursor</td>
<td>Required. The cursor returned in the previous call.</td>
</tr>
<tr>
<td>originator</td>
<td>Optional. Filter the logs by this field's value.</td>
</tr>
<tr>
<td>originator[]</td>
<td>Optional. Multiple originators can be supplied through this parameter.</td>
</tr>
<tr>
<td>not_originator[]</td>
<td>Optional. The supplied originators will not be included in the returned logs.</td>
</tr>
<tr>
<td>after_date</td>
<td>Optional. Logs created before this date will be returned.</td>
</tr>
</tbody>
</table>
### Returns

An array of 20 or the limit amount of log entries, starting from the specified cursor position, for the nodes in the specified node family. The most recent entries appear first. See "Get all logs", above, for more details.

### Search in logs

Retrieve log entries for which all the given criteria hold. The criteria can be defined as a logical conjunction using some filters defined as follows.

- **q=TEXT**: The log message contains the TEXT string.
- **originator=ORIGINATOR**: The originator of the log is ORIGINATOR.
- **node=NODEID**: The log was collected from node NODEID.
- **node-family=NODEFAMILYID**: The log was collected from a node belonging to node family NODEFAMILYID.
- **negate**: The log message does not contain the 'TEXT' string specified via the 'q' parameter.
- **after_date=AFTER_DATE_AND_TIME**: The logs that were created after AFTER_DATE_AND_TIME will be returned.
- **tags=LISTOFTAGS**: The log entry is tagged with at least one of the given tags.

### Definition

```
GET /api/logs/search?q=TEXT&originator=ORIGINATOR&node=NODEID&node-family=NODEFAMILYID&negate&after_date=AFTER_DATE_AND_TIME&tags=dev,critical
```

### Example requests

- Retrieves all the log entries that contain the `error` string in their log message.
  ```bash
curl -X GET "http://127.0.0.1:8080/api/logs/search?q=error"
  ```

- Retrieves all the log entries that satisfy all the following properties:
  - The log message contains the `terminated` string.
  - The originator of the log is `lager`.
  - The log was collected from node `cfeae528-d45c-4a9e-8a41-aefa39d0cdef`.
  ```bash
curl -X GET "http://127.0.0.1:8080/api/logs/search?q=terminated&originator=lager&node=cfeae528-d45c-4a9e-8a41-aefa39d0cdef"
  ```

- Retrieves all the log entries that satisfy all the following properties:
  - The log message contains the `loaded` string.
  - The log was collected from any of the nodes belonging to node family `c86645ba-684c-4ca8-a722-cdd51e3ba273`.
  ```bash
curl -X GET "http://127.0.0.1:8080/api/logs/search?q=loaded&node-family=c86645ba-684c-4ca8-a722-cdd51e3ba273"
  ```
curl -X GET "http://127.0.0.1:8080/api/logs/search?q=loaded&node-family=c86645ba-684c-4ca8-a722-cdd51e3ba273"

- Retrieves all the log entries that does not contain the 'error' string in their log message.

curl -X GET "http://127.0.0.1:8080/api/logs/search?q=error&negate"

**Example response**

See "Get all logs", above.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td><strong>Required.</strong> The string to search for.</td>
</tr>
<tr>
<td>cs</td>
<td><strong>Optional.</strong> Do a case sensitive search.</td>
</tr>
<tr>
<td>originator</td>
<td><strong>Optional.</strong> Filter the logs by this field's value.</td>
</tr>
<tr>
<td>originator[]</td>
<td><strong>Optional.</strong> Multiple originators can be supplied through this parameter.</td>
</tr>
<tr>
<td>not_originator[]</td>
<td><strong>Optional.</strong> The supplied originators will not be included in the returned logs.</td>
</tr>
<tr>
<td>node</td>
<td><strong>Optional.</strong> The identifier of a node. Only those logs are considered that were collected from this node.</td>
</tr>
<tr>
<td>node-family</td>
<td><strong>Optional.</strong> The identifier of a node family. Only those logs are considered that were collected from this node family.</td>
</tr>
<tr>
<td>negate</td>
<td><strong>Optional.</strong> The log message does not contain the TEXT string specified via the q parameter.</td>
</tr>
<tr>
<td>after_date</td>
<td><strong>Optional.</strong> Logs created before this date will be returned.</td>
</tr>
<tr>
<td>tags</td>
<td><strong>Optional.</strong> A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
<tr>
<td>limit</td>
<td><strong>Optional.</strong> A positive integer specifying the maximum number of logs returned. The maximum is 100.</td>
</tr>
</tbody>
</table>

**Returns**

An array of the 20 or the limit amount of most recent log entries for which all the given criteria hold. See “Get all logs”, above, for more details.

**Get 20 logs from a cursor position for the search query**

Retrieve older log entries for which all the given criteria hold. The criteria can be defined as a logical conjunction using some filters defined as follows.

- q=TEXT: The log message contains the TEXT string.
- originator=ORIGINATOR: The originator of the log is ORIGINATOR.
- node=NODEID: The log was collected from node NODEID.
- `node-family=NODEFAMILYID`: the log was collected from a node belonging to node family NODEFAMILYID.
- `negate`: The log message does not contain the TEXT string specified via the q parameter.
- `after_date=AFTER_DATE_AND_TIME`: The logs that were created after AFTER_DATE_AND_TIME will be returned.
- `tags=LISTOFTAGS`: The log entry is tagged with at least one of the given tags.

**Definition**

```
GET /api/logs/previous/search?q=TEXT&c=CURSOR&cs&originator=ORIGINATOR&node=NODEID&node-family=NODEFAMILYID&negate&after_date=AFTER_DATE_AND_TIME&tags=dev,critical
```

**Example response**

See "Get all logs", above.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cursor</td>
<td><strong>Required.</strong>  The cursor returned in the previous call.</td>
</tr>
<tr>
<td><code>q</code></td>
<td><strong>Required.</strong>  The string to search for.</td>
</tr>
<tr>
<td><code>cs</code></td>
<td><strong>Optional.</strong>  Do a case sensitive search.</td>
</tr>
<tr>
<td><code>originator</code></td>
<td><strong>Optional.</strong>  Filter the logs by this field's value.</td>
</tr>
<tr>
<td><code>originator[]</code></td>
<td><strong>Optional.</strong>  Multiple originators can be supplied through this parameter.</td>
</tr>
<tr>
<td><code>not_originator[]</code></td>
<td><strong>Optional.</strong>  The supplied originators will not be included in the returned logs.</td>
</tr>
<tr>
<td><code>node</code></td>
<td><strong>Optional.</strong>  The identifier of a node. Only those logs are considered that were collected from this node.</td>
</tr>
<tr>
<td><code>node-family</code></td>
<td><strong>Optional.</strong>  The identifier of a node family. Only those logs are considered that were collected from this node family.</td>
</tr>
<tr>
<td><code>negate</code></td>
<td><strong>Optional.</strong>  The log message does not contain the TEXT string specified via the q parameter.</td>
</tr>
<tr>
<td><code>after_date</code></td>
<td><strong>Optional.</strong>  Logs created before this date will be returned.</td>
</tr>
<tr>
<td><code>tags</code></td>
<td><strong>Optional.</strong>  A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
<tr>
<td><code>limit</code></td>
<td><strong>Optional.</strong>  A positive integer specifying the maximum number of logs returned. The maximum is 100.</td>
</tr>
</tbody>
</table>

**Returns**

An array of 20 or the specified amount of log entries, starting from the specified cursor position, for which all the given criteria hold. See "Get all logs", above, for more details.

**Get all log originators**
Retrieve the list of log originators for all nodes.

**Definition**

GET /api/logs/originator

**Example request**

curl -X GET "http://127.0.0.1:8080/api/logs/originator"

**Example response**

```
{
    "originators" : [
        "Module checker",
        "Shell",
        "System monitor",
        "lager"
    ]
}
```

**Arguments**

None.

**Returns**

An array that contains each originator that can be used for filtering logs.

**Get log originators for a node**

Retrieve the list of log originators for a given node.

**Definition**

GET /api/logs/originator/node/NODE_ID

**Example request**

curl -X GET "http://127.0.0.1:8080/api/logs/originator/node/3fa56f92-3195-401b-93a8-7bc43e54d66f"

**Example response**

See "Get all log originators", above.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td><strong>Required.</strong> The node's identifier.</td>
</tr>
</tbody>
</table>

**Returns**

See "Get all log originators", above.

**Get log originators for a node family**

Retrieve the list of log originators for a given node family.

**Definition**

GET /api/logs/originator/node-family/NODE_FAMILY_ID
Example request

curl -X GET "http://127.0.0.1:8080/api/logs/originator/node-family/93be4f01-dae6-49b0-9608-4552f841a720"

Example response

See “Get all log originators”, above.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family ID</td>
<td>Required. The node family's identifier</td>
</tr>
</tbody>
</table>

Returns

See “Get all log originators”, above.

Get log levels for a node

Retrieve the current log levels for a specified node.

Definition

GET /api/logs/level/NODE_ID

Example request

curl -X GET "http://127.0.0.1:8080/api/logs/level/3fa56f92-3195-401b-93a8-7bc43e54d66f"

Example response

```
{
  "logLevels": [
    {
      "handler": "wo_logs_lager_agent",
      "logLevel": "warning"
    },
    {
      "handler": "wo_logs_elogger_agent",
      "logLevel": "warning"
    }
  ]
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The node's identifier</td>
</tr>
</tbody>
</table>

Returns

An array that contains each handler and its current log level for the specified node.

Set log level for current node

Update the log level for the specified handler on a node.
**Definition**

POST /api/logs/level/NODE_ID

**Data**

```
{
  "handler": "HANDLER",
  "level": "SEVERITY"
}
```

**Example request**

```
curl -X POST "http://127.0.0.1:8080/api/logs/level/3fa56f92-3195-401b-93a8-7bc43e54d66f" \
   -H "Content-Type: application/json" \
   -d "{\"handler\":\"wo_logs_lager_agent\", \"level\":\"info\"}"
```

**Example response**

```
{}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The node's identifier.</td>
</tr>
<tr>
<td>handler</td>
<td>Required. The handler for which to update the log level.</td>
</tr>
<tr>
<td>level</td>
<td>Optional. The new log level (info, warning, or error).</td>
</tr>
</tbody>
</table>

**Returns**

An empty object. If successful, the HTTP response is 200 OK.

**Get part of a log message**

If a log message is too big (more than 1000000 bytes), WombatOAM might run out of memory crash when encoding it to JSON. The web browsers have trouble displaying this much data too. To avoid these problems, the server only returns big log messages page by page, when a page contains around 1000 bytes data (might be 1-3 bytes more to avoid splitting multibyte UTF-8 encoded characters). This function is used to get these pages. To go through the pages in forward direction, set the FROM value to the start property and set DIRECTION to next. Set DIRECTION to prev and you can go backwards within the pages. If DIRECTION is set to first or last, then the first or last page is returned, regardless of the FROM value. The hasNext and the hasPrev properties in the result show if there are any more pages in either direction.

Get first/last/next/previous page (1000 bytes) of a single log message.

**Definition**

GET /api/logs/previous/part?c=CURSOR&from=FROM&direction=DIRECTION

**Example requests**

```
curl -X GET "http://127.0.0.1:8080/api/logs/previous/part?c=r3j"
```
Example response

```
{
  "logs": [
    {
      "date": "1494850906467",
      "severity": "error",
      "originator": "error_logger",
      "nodeId": "3fcaf6a5-a222-40e0-a708-ef19c4eeb0b",
      "nodeName": "test_app@127.0.0.1",
      "message": "187, 94188, 94189, 94190, 94191, 94192, 94193, \n5 94194, 94195, 94196, 94197, 94198, 94199, 94200",
      "props": {
        "cursor": "r3jpgnb8hSoMyL9RNkPhTRrsEeaDbAAAAAFqaAJoA2IAAAXXYgADIHZiAA0YHWEU\n6 f rom=0&direction=next",
        "hasNext": true,
        "start": 1001,
        "hasPrev": true
      },
      "type": "logEvent",
      "displayName": "test_app@127.0.0.1"
    },
    {
      "cursor": null
    }
  ]
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cursor</strong></td>
<td>Required. The cursor returned from the previous call. This specifies the log message.</td>
</tr>
<tr>
<td><strong>from</strong></td>
<td>Required. The index of the byte from where the previous log message part is returned.</td>
</tr>
<tr>
<td><strong>direction</strong></td>
<td>Required. The direction of the next page. Can be first, prev, next or last.</td>
</tr>
</tbody>
</table>
Metrics REST interface

- Get metrics metadata for nodes
- Get metric metadata for node families
- Get metric historical data for a node
- Get metric historical data for a node family
- Get metric histogram for a node
- Get metric duration for a node
- Get metric meter for a node
- Get spiral metric for a node
- Get gauge info for a node

Get metrics metadata for nodes

Get metadata for all known metrics in the managed nodes. The metrics are grouped hierarchically according to three main groups (wombat_metrics, folsom_metrics, and exometer_metrics) and the categories under which they are displayed in the web dashboard.

**Definition**

GET /api/metric/metric-info/node/NODE_ID?tags=LISTOFTAGS

**Example request**

curl -X GET "http://127.0.0.1:8080/api/metric/metric-info/node/3fa56f92-3195-401b-93a8-7bc43e54d66f?tags=dev,op"

**Example response**

```json
{
  "groups": [ 
  {
    "name": "wombat metrics",
    "display": "WombatOAM metrics",
    "categories": [ 
      {
        "display": "Memory",
        "metrics": [ 
          {
            "name": "total_memory",
            "type": "gauge",
            "display": "Total memory",
            "unit": "byte"
          },
          {
            "name": "atom_memory",
            "type": "gauge",
            "display": "Atom memory",
            "unit": "byte"
          }
        ]
      },
      {
        "display": "Runtime",
        "metrics": [ 
          {
            "name": "cpu_avg1",
            "type": "gauge",
            "display": "CPU load for 1 avg",
            "unit": "numeric"
          }
        ]
      }
    ]
  }
}
```
```
{
  
  },
  {
    {
      
    },
    {
      
    },
    {
      
    }
  
  },
  {
    "name": "folsom_metrics",
    "display": "Folsom metrics",
    "categories": [
      {
        "display": "Folsom metrics",
        "metrics": [
          {
            "name": "cpu",
            "type": "gauge",
            "display": "cpu",
            "unit": "numeric"
          }
        
        },
        {
          
        },
        {
          
        }
      
      },
      {
        "name": "exometer_metrics",
        "display": "Exometer metrics",
        "categories": [
          {
            "display": "Exometer metrics",
            "metrics": [
              {
                "name": "[cpu,counter]",
                "type": "counter",
                "display": "cpu_counter",
                "unit": "numeric"
              }
            
          },
          {
            
          },
          {
            
          }
        
        },
        {
          
        },
        {
          
        }
      
      }
    
  }

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node for which to retrieve metrics metadata.</td>
</tr>
<tr>
<td>tags</td>
<td>Optional. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
</tbody>
</table>

Returns

An array of metric groups, categories, and information about individual metrics that are tagged with any of the active tags. The "display" property is the text that is displayed in the web dashboard.

For each individual metric:
### Get metric metadata for node families

Get metadata for all known metrics in node families.

**Definition**

```
GET /api/metric/metric-info/node-family/NODE_FAMILY_ID?tags=LIS\nTOFTAGS
```

**Example request**

curl -X GET "http://127.0.0.1:8080/api/metric/metric-info/node-family/93be4f01-dae6-49b0-9608-4552f841a720"

**Example response**

See "Get metrics metadata for nodes", above.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family ID</td>
<td><strong>Required</strong>. Description</td>
</tr>
<tr>
<td>tags</td>
<td><strong>Optional</strong>. A comma separated list of tags that overrides the tags set for the user.</td>
</tr>
</tbody>
</table>

**Returns**

An array of metric groups, categories, and information about individual metrics that are tagged with *any* of the active tags. The "display" property is the text that is displayed in the web dashboard.

For each individual metric:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The internal name of the metric. This name should be used in calls to retrieve data for specific individual metrics.</td>
</tr>
<tr>
<td>type</td>
<td>The type of metric, determining the call that should be used when retrieving data for that metric.</td>
</tr>
<tr>
<td>display</td>
<td>The text displayed in the web dashboard.</td>
</tr>
<tr>
<td>unit</td>
<td>The metric's unit of measurement.</td>
</tr>
</tbody>
</table>

### Get metric historical data for a node

Retrieve the historical data for a specified node and a specified metric.
The request needs to specify the series from which to query (see "Arguments", below), as well as the time interval.

**Definition**

GET /api/metric/history/node/NODE_ID/METRIC/SERIES?from=FROM&to=TO

**Example request**

curl -X GET "http://127.0.0.1:8080/api/metric/history/node/3fa56f92-3195-401b-93a8-7bc43e54d66f/total_memory/3?from=2014-07-31T00:00:00.000Z&to=2014-07-31T23:59:59.999Z"

**Example response**

```
{
  "data": [
    {
      "time": "2014-07-31 12:40:40.000",
      "jstime": "1406810440000",
      "value": 50258288.0
    },
    {
      "time": "2014-07-31 12:55:40.000",
      "jstime": "1406811340000",
      "value": 43555390.4
    }
  ]
}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td><strong>Required.</strong> The identifier of the node for which to retrieve metrics data.</td>
</tr>
<tr>
<td>metric</td>
<td><strong>Required.</strong> The name of the metric (see &quot;Get metrics metadata...&quot;, above).</td>
</tr>
<tr>
<td>series</td>
<td><strong>Required.</strong> The numeric identifier of the series. 1 means the base sample data, 2 is the first consolidated series, and 3 is the second consolidated series.</td>
</tr>
<tr>
<td>from</td>
<td><strong>Required.</strong> The start date and time of the samples. Combined date and time in ISO 8601 format. For example: 2014-07-31T00:00:00.000Z</td>
</tr>
<tr>
<td>to</td>
<td><strong>Required.</strong> The end date and time of the samples. Combined date and time in ISO 8601 format. For example: 2014-07-31T23:59:59.999Z</td>
</tr>
</tbody>
</table>

**Returns**

A data property that contains an array of metric samples, each with the following details:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>The date and time of the sample.</td>
</tr>
<tr>
<td>jstime</td>
<td>The millisecond value of the date and time.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the metric sample.</td>
</tr>
</tbody>
</table>
Get metric historical data for a node family

Retrieve the historical data for a specified node family and a specified metric. The request needs to specify the series from which to query (see "Arguments", below), as well as the time interval.

**Definition**

```
GET /api/metric/history/node-family/NODE_FAMILY_ID/METRIC/SERIES?from=FROM&to=TO
```

**Example request**

curl -X GET "http://127.0.0.1:8080/api/metric/history/node-family/93be4f01-da6-49b0-9608-4552f8413720/total_memory/3?from=2014-07-31T00:00:00.000Z&to=2014-07-31T23:59:59.999Z"

**Example response**

```
[  
  {  
    "nodeId": "5b16ec55-56a3-4430-b174-c996897cc14",  
    "data": [  
      {  
        "time": "2014-07-31 12:40:40.000",  
        "jstime": "1406810440000",  
        "value": 49912632.0     
      },  
      {  
        "time": "2014-07-31 12:55:41.000",  
        "jstime": "1406811341000",  
        "value": 43825455     
      },  
      {  
        "time": "2014-07-31 13:10:42.000",  
        "jstime": "1406812842000",  
        "value": 43525455     
      }  
    ]  
  },  
  {  
    "nodeId": "913ca6f92-3195-401b-93a8-7bc43e54d66f",  
    "data": [  
      {  
        "time": "2014-07-31 12:40:40.000",  
        "jstime": "1406810440000",  
        "value": 50258288.0     
      },  
      {  
        "time": "2014-07-31 12:55:41.000",  
        "jstime": "1406811341000",  
        "value": 43555233     
      },  
      {  
        "time": "2014-07-31 13:10:42.000",  
        "jstime": "1406812842000",  
        "value": 43555233     
      }  
    ]  
  }  
]
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node family ID</td>
<td><strong>Required.</strong> The identifier of the node family for which to retrieve metrics data.</td>
</tr>
<tr>
<td>metric</td>
<td><strong>Required.</strong> The name of the metric (see &quot;Get metrics REST interface&quot;)</td>
</tr>
</tbody>
</table>
### series
**Required.** The numeric identifier of the series. 1 means the base sample data, 2 is the first consolidated series, and 3 is the second consolidated series.

### from
**Required.** The start date and time of the samples. Combined date and time in ISO 8601 format. For example: `2014-07-31T00:00:00.000Z`

### to
**Required.** The end date and time of the samples. Combined date and time in ISO 8601 format. For example: `2014-07-31T23:59:59.999Z`

## Returns
For each node, a data property that contains an array of metric samples, each with the following details:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>The date and time of the sample.</td>
</tr>
<tr>
<td>jstime</td>
<td>The millisecond value of the date and time.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the metric sample.</td>
</tr>
</tbody>
</table>

### Get metric histogram for a node
Use this call to retrieve metric data for metrics of the type `histogram` (see "Get metrics metadata for a node").

**Definition**

```
GET /api/metric/histogram/node/NODE_ID/METRIC?from=FROM&to=TO
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/metric/histogram/node/3f56f92-3195-401b-93a8-7bc43e54d66f/%7Bcpu,histogram%7D?from=2014-07-31T00:00:00.000Z&to=2014-07-31T23:59:59.999Z"
```

**Example response**

```
{
  "data": [
    {
      "jstime": "1406810830000",
      "value": {
        "min": 0,
        "max": 100,
        "arithmetic_mean": 51.00486381322957,
        "geometric_mean": 35.66788826103998,
        "harmonic_mean": 23.94843838939527,
        "median": 52,
        "variance": 1002.5579120554382,
        "standard_deviation": 31.663194912317966,
        "skewness": -0.053591991674156764,
        "kurtosis": -1.3019963450719712,
        "n": 1028,
        "percentiles": [
          {
            "percentile": 50, "value": 52},
          {
            "percentile": 75, "value": 79},
          {
            "percentile": 90, "value": 93},
          {
            "percentile": 95, "value": 98}
        ]
    }
  ]
}
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>node ID</strong></td>
<td><em>Required.</em> The identifier of the node for which to retrieve metrics data.</td>
</tr>
<tr>
<td><strong>metric</strong></td>
<td><em>Required.</em> The name of the metric (see &quot;Get metrics metadata…&quot;, above).</td>
</tr>
<tr>
<td><strong>from</strong></td>
<td><em>Required.</em> The start date and time of the samples. Combined date and time in ISO 8601 format. For example: 2014-07-31T00:00:00.000Z</td>
</tr>
<tr>
<td><strong>to</strong></td>
<td><em>Required.</em> The end date and time of the samples. Combined date and time in ISO 8601 format. For example: 2014-07-31T23:59:59.999Z</td>
</tr>
</tbody>
</table>

Returns

A data property that contains an array of metric samples within the specified time interval, each with the following details:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>time</strong></td>
<td>The date and time of the sample.</td>
</tr>
<tr>
<td><strong>jstime</strong></td>
<td>The millisecond value of the date and time.</td>
</tr>
<tr>
<td><strong>value</strong></td>
<td>A data structure which provides the details of the data point.</td>
</tr>
</tbody>
</table>

Get metric duration for a node

Use this call to retrieve metric data for metrics of the type **duration** (see "Get metrics metadata for a node").

Definition

```
GET /api/metric/duration/node/NODE_ID/METRIC?from=FROM&to=TO
```  

Example request

```
curl -X GET "http://127.0.0.1:8080/api/metric/duration/node/3fa"
```
Example response

```json
{
  "data": [
    {
      "jstime": "1406810830000",
      "count": 2567,
      "last": 34,
      "min": 0,
      "max": 100,
      "arithmetic_mean": 51.00486381322957,
      "geometric_mean": 35.66788826103998,
      "harmonic_mean": 23.94843838939527,
      "median": 52,
      "variance": 1002.5579120554382,
      "standard_deviation": 31.663194912317966,
      "skewness": -0.053591991674156764,
      "kurtosis": -1.3019963450719712,
      "n": 1028,
      "percentiles": [
        {"percentile": 50, "value": 52},
        {"percentile": 75, "value": 79},
        {"percentile": 90, "value": 93},
        {"percentile": 95, "value": 98},
        {"percentile": 99, "value": 100},
        {"percentile": 999, "value": 100}
      ],
      "histogram": [
        {"x": 11, "num": 146},
        {"x": 22, "num": 106},
        {"x": 40, "num": 176},
        {"x": 50, "num": 74},
        {"x": 60, "num": 81},
        {"x": 70, "num": 103},
        {"x": 80, "num": 91},
        {"x": 90, "num": 107},
        {"x": 100, "num": 144},
        {"x": 110, "num": 0}
      ]
    }
  ],
  "error": null
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td><strong>Required.</strong> The identifier of the node for which to retrieve metrics data.</td>
</tr>
<tr>
<td>metric</td>
<td><strong>Required.</strong> The name of the metric (see &quot;Get metrics metadata...&quot;, above).</td>
</tr>
<tr>
<td>from</td>
<td><strong>Required.</strong> The start date and time of the samples.</td>
</tr>
<tr>
<td>to</td>
<td><strong>Required.</strong> The end date and time of the samples.</td>
</tr>
</tbody>
</table>

The **from** and **to** fields are combined date and time in ISO 8601 format. For example: 2014-07-31T23:59:59.999Z
Returns
A data property that contains an array of metric samples within the specified time interval, each with the following details:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>The date and time of the sample.</td>
</tr>
<tr>
<td>jstime</td>
<td>The millisecond value of the date and time.</td>
</tr>
<tr>
<td>value</td>
<td>A data structure which provides the details of the data point.</td>
</tr>
</tbody>
</table>

Get metric meter for a node

Use this call of retrieving metric data for metrics of the type **meter** (see "Get metrics metadata for a node").

**Definition**

GET /api/metric/meter/node/NODE_ID/METRIC?from=FROM&to=TO

**Example request**

curl -X GET "http://127.0.0.1:8080/api/metric/meter/node/3fa56f92-3195-401b-93a8-7bc43e54d66f/%7Bcpu,meter%7D?from=2014-07-31T00:00:00.000Z&to=2014-07-31T23:59:59.999Z"

**Example response**

```json
{
  "data": [
    {
      "jstime": "1406810830000",
      "value": {
        "one": "2.068",
        "five": "3.085",
        "fifteen": "2.993",
        "day": "2.226"
      }
    },
    {
      "time": "2014-07-31 12:47:59.213",
      "jstime": "1406810879000",
      "value": {
        "one": "4693.918",
        "five": "1164.305",
        "fifteen": "403.433",
        "day": "6.443"
      }
    }
  ]
}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td><strong>Required.</strong> The identifier of the node for which to retrieve metrics data.</td>
</tr>
<tr>
<td>metric</td>
<td>Required. The name of the metric (see &quot;Get metrics metadata...&quot;, above).</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>from</td>
<td>Required. The start date and time of the samples. Combined date and time in ISO 8601 format. For example: 2014-07-31T00:00:00.000Z</td>
</tr>
<tr>
<td>to</td>
<td>Required. The end date and time of the samples. Combined date and time in ISO 8601 format. For example: 2014-07-31T23:59:59.999Z</td>
</tr>
</tbody>
</table>

**Returns**

A data property that contains an array of metric samples within the specified time interval, each with the following details:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>The date and time of the sample.</td>
</tr>
<tr>
<td>jstime</td>
<td>The millisecond value of the date and time.</td>
</tr>
<tr>
<td>value</td>
<td>The data points at the given time.</td>
</tr>
</tbody>
</table>

**Get spiral metric for a node**

Use this call of retrieving metric data for metrics of the type **spiral** (see "Get metrics metadata for a node").

**Definition**

```
GET /api/metric/spiral/node/NODE_ID/METRIC/SERIES?from=FROM&to=TO
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/metric/spiral/node/3fa56f92-3195-401b-93a8-7bc43e54d66f/%5Bcpu,duration%5D/2?from=2014-07-31T00:00:00.000Z&to=2014-07-31T23:59:59.999Z"
```

**Example response**

```
{
    "data": [
        {
            "jstime": "1406810833000",
            "value": {
                "count": 305076,
                "one": 38554
            }
        },
        {
            "time": "2014-07-31 12:48:01.240",
            "jstime": "1406810881000",
            "value": {
                "count": 737557,
                "one": 461738
            }
        }
    ]
}
```
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node for which to retrieve metrics data.</td>
</tr>
<tr>
<td>metric</td>
<td>Required. The name of the metric (see &quot;Get metrics metadata..., above).</td>
</tr>
<tr>
<td>Series</td>
<td>Optional. The numeric identifier of the series. 1 means the base sample data, 2 is the first consolidated series, and 3 is the second consolidated series. The default value is 1.</td>
</tr>
<tr>
<td>from</td>
<td>Required. The start date and time of the samples. Combined date and time in ISO 8601 format. For example: 2014-07-31T00:00:00.000Z</td>
</tr>
<tr>
<td>to</td>
<td>Required. The end date and time of the samples. Combined date and time in ISO 8601 format. For example: 2014-07-31T23:59:59.999Z</td>
</tr>
</tbody>
</table>

### Returns

A data property that contains an array of metric samples within the specified time interval, each with the following details:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>The date and time of the sample.</td>
</tr>
<tr>
<td>jstime</td>
<td>The millisecond value of the date and time.</td>
</tr>
<tr>
<td>value</td>
<td>The data points at the given time.</td>
</tr>
</tbody>
</table>

### Get gauge info for a node

Retrieve the current configuration of gauges shown on the web dashboard.

#### Definition

GET /api/metric/gauge-info/node/NODE_ID

#### Example request

```
curl -X GET "http://127.0.0.1:8080/api/metric/gauge-info/node/3fa56f92-3195-401b-93a8-7bc43e54d66f"
```

#### Example response

```
{
  "gauges": [
    {
      "name": "total_memory",
      "min_value": 10000000,
      "max_value": 30000000
    }
  ]
}
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node for which to retrieve gauge details.</td>
</tr>
</tbody>
</table>

**Returns**

An array containing each gauge's name, minimum value, and maximum value.
Services REST interface

GET requests for getting information about services:
- Get all configurator services available for a node
- Get all configurator services available for a node family
- Get all explorer services available for a node
- Get all explorer services available for a node family
- Get all executor services available for a node
- Get all executor services available for a node family

POST requests to initiate or stop a request:
- Initiate a new service request
- Initiate a feature request
- Stop a running request
- Initiate new process monitors

GET requests for getting information about requests:
- Get information about a request
- Get all running request per node
- Get all running request per node family
- Get all finished requests per node
- Get all finished requests per node family

Get all configurator services available for a node

Definition

GET /api/service/service-info/configurator/node/NODE_ID

Example request

curl -X GET "http://127.0.0.1:8080/api/service/service-info/configurator/node/5b9625f2-2fd1-4311-92aa-3477b33d11d7"

Example response

```json
{
  "service_infos": [{
    "serviceId": "g2gFZAAEbm9kZW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjItMmZkMzlsOmx5GzI1dW0AAAAkNWI5NjI1ZjI=",
    "objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7",
    "isInternal": false,
    "label": "Display an application's configuration parameters",
    "description": "Retrieve all configuration parameters of the given application."
  },
  "options": []
}
```
"crypto",
"elarm",
"exometer",
"folsom",
"goldrush",
"gproc",
"kernel",
"lager",
"os_mon",
"sasl",
"stdlib",
"syntax_tools",
"wo_test",
"wombat_plugin"
],
"listitemType": []
}
}
{
"serviceId": "g2gfZAAEbm9kZW0AAAkNWI5Nj1lzItMmZkM500MzExLTkyY2EzMTQyZWN0b3JpZmljYXRlZmFsc2V9",
"objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7",
"isInternal": false,
"label": "Set a configuration parameter",
"description": "Set a configuration parameter of the given application.",
"options": [
{
"name": "application",
"label": "Application",
"type": "enum",
"default": "",
"enabled": true,
"mandatory": true,
"values": [
"bear",
"compiler",
"crypto",
"elarm",
"exometer",
"folsom",
"goldrush",
"gproc",
"kernel",
"lager",
"os_mon",
"sasl",
"stdlib",
"syntax_tools",
"wo_test",
"wombat_plugin"
],
"listitemType": []
},
{
"name": "changes",
"label": "List of configuration parameters to change",
"type": "list",
"default": "",
"enabled": true,
"mandatory": true,
"values": [],
"listitemType": []
}
]
"enabled": true,
"mandatory": true,
"values": [],
"listItemType": []
},

{
   "name": "configParamValue",
   "label": "The value to be set for the configuration parameter",
   "type": "string",
   "default": "",
   "enabled": true,
   "mandatory": true,
   "values": [],
   "listItemType": []
},

{
   "name": "persist",
   "label": "Shall the change be persistent?",
   "type": "enum",
   "default": "No",
   "enabled": false,
   "mandatory": true,
   "values": [
      "Yes",
      "No"
   ],
   "listItemType": []
}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
}]
]
]}
]
]}
]
}]
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
]}
]
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_FAMILY_ID</td>
<td>Required. The identifier of the node family.</td>
</tr>
</tbody>
</table>

Returns

A service_infos object containing the list of available services for the given node family.
Get all explorer services available for a node

**Definition**

```
GET /api/service/service-info/explorer/node/NODE_ID
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/service/service-info/explorer/node/5b9625f2-2fd1-4311-92aa-3477b33d11d7"
```

**Example response**

```
{
  "service_infos": [
    {
      "serviceId": "g2gFZAAEbm9kZW0AAAAkNWI5NjI1ZjItMmZkM S00MzExLTkyWtMzQ3N2IzM2QxMWQ3ZAAIZXhwbG9yZXRJkABjwcm9jZX
      NzXzRyY3Rrb25hcnaKABB3b21iYXRAMTI3LjAuMC4xAgAANM4AAAAAAA=",
      "objectType": "node",
      "objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7",
      "isInternal": false,
      "label": "Process dictionary",
      "description": "Show process dictionary",
      "options": [
        {
          "name": "Name",
          "label": "Registered name",
          "type": "string",
          "default": "",
          "enabled": true,
          "mandatory": true,
          "values": [],
          "listitemType": []
        }
      ]
    },
    {
      "serviceId": "g2gFZAAEbm9kZW0AAAAkNWI5NjI1ZjItMmZkM S00MzExLTkyWtMzQ3N2IzM2QxMWQ3ZAAIZXhwbG9yZXRJkABjwcm9jZX
      NzXzRyY3Rrb25hcnaKABB3b21iYXRAMTI3LjAuMC4xAgAANM4AAAAAAA=",
      "objectType": "node",
      "objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7",
      "isInternal": false,
      "label": "Process info",
      "description": "Show process info",
      "options": [
        {
          "name": "Name",
          "label": "Registered name",
          "type": "string",
          "default": "",
          "enabled": true,
          "mandatory": true,
          "values": [],
          "listitemType": []
        }
      ]
    },
    {
      "serviceId": "g2gFZAAEbm9kZW0AAAAkNWI5NjI1ZjItMmZkM S00MzExLTkyWtMzQ3N2IzM2QxMWQ3ZAAIZXhwbG9yZXRJkABjwcm9jZX
      NzXzRyY3Rrb25hcnaKABB3b21iYXRAMTI3LjAuMC4xAgAANM4AAAAAAA=",
      "objectType": "node",
      "objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7",
      "isInternal": false,
      "label": "Process messages",
      "description": "Show process messages",
      "options": [
        {
          "name": "Name",
          "label": "Registered name",
          "type": "string",
          "default": "",
          "enabled": true,
          "mandatory": true,
          "values": [],
          "listitemType": []
        }
      ]
    }
  ]
}
```
"description": "Show process messages",
"options": [
  {
    "name": "Name",
    "label": "Registered name",
    "type": "string",
    "default": ",",
    "enabled": true,
    "mandatory": true,
    "values": [],
    "listitemType": []
  },
  {
    "name": "Name",
    "label": "Registered name",
    "type": "string",
    "default": ",",
    "enabled": true,
    "mandatory": true,
    "values": [],
    "listitemType": []
  },
  {
    "name": "Name",
    "label": "Registered name",
    "type": "string",
    "default": ",",
    "enabled": true,
    "mandatory": true,
    "values": [],
    "listitemType": []
  }
],
"serviceId": "g2gFZAAEbm9kZW0AAAAnKNwI5NjI1ZjItMmZkM0Z0MzExLTkyYy9tNzIuY29tYmF0QDExNjEwLjAwMjE4LjUzNl0AAAAA=",
"objectType": "node",
"objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7",
"isInternal": false,
"label": "Process stack trace",
"description": "Show process stack trace",
"options": [
  {
    "name": "Name",
    "label": "Registered name",
    "type": "string",
    "default": ",",
    "enabled": true,
    "mandatory": true,
    "values": [],
    "listitemType": []
  },
  {
    "name": "Name",
    "label": "Registered name",
    "type": "string",
    "default": ",",
    "enabled": true,
    "mandatory": true,
    "values": [],
    "listitemType": []
  },
  {
    "name": "Name",
    "label": "Registered name",
    "type": "string",
    "default": ",",
    "enabled": true,
    "mandatory": true,
    "values": [],
    "listitemType": []
  }
],
"serviceId": "g2gFZAAEbm9kZW0AAAAnKNwI5NjI1ZjItMmZkM0Z0MzExLTkyYy9tNzIuY29tYmF0QDExNjEwLjAwMjE4LjUzNl0AAAAA=",
"objectType": "node",
"objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7",
"isInternal": false,
"label": "Process state",
"description": "Show process state",
"options": [
  {
    "name": "Name",
    "label": "Registered name",
    "type": "string",
    "default": ",",
    "enabled": true,
    "mandatory": true,
    "values": [],
    "listitemType": []
  },
  {
    "name": "Name",
    "label": "Registered name",
    "type": "string",
    "default": ",",
    "enabled": true,
    "mandatory": true,
    "values": [],
    "listitemType": []
  },
  {
    "name": "Name",
    "label": "Registered name",
    "type": "string",
    "default": ",",
    "enabled": true,
    "mandatory": true,
    "values": [],
    "listitemType": []
  }
]
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ID</td>
<td>Required. The identifier of the node.</td>
</tr>
</tbody>
</table>

Returns

A service_infos object containing the list of available services for the given node.

Get all explorer services available for a node family

Definition
GET /api/service/service-info/explorer/node-family/NODE_FAMILY_ID

Example request

curl -X GET "http://127.0.0.1:8080/api/service/service-info/explorer/node-family/12a74b0b-2d50-4270-a90c-2b5e50c2e0c8"

Example response

```
1 2 3  
{ "service_infos": [] }
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_FAMILY_ID</td>
<td>Required. The identifier of the node family.</td>
</tr>
</tbody>
</table>

Returns

A service_infos object containing the list of available services for the given node family.

Get all executor services available for a node

Definition

GET /api/service/service-info/executor/node/NODE_ID

Example request

curl -X GET "http://127.0.0.1:8080/api/service/service-info/executor/node/5b9625f2-2fd1-4311-92aa-3477b33d11d7"

Example response

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
{ "service_infos": [
  {
    "serviceId": "g1AAAAB3eJwtxkE0gjAQRuE/qIlLFx6BbY0zAzbsvAlpmSFuSpMC0WtwY0z0fZv301mOUI2LA0oZdq9uR3Y8KrlG1FzHIThpvi8i5qRecbaPDeuSi+L6Wv+vX2Uo/5JTCpAUHxeWdwzLk9jf7l9UAbLh3w5ZB63",
    "objectType": "node",
    "objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7",
    "isInternal": false,
    "label": "Execute user command",
    "description": "Execute user command on the managed node.",
    "options": [
      {
        "name": "Expr",
        "label": "Expression to be evaluated",
        "type": "string",
        "default": "",
        "enabled": true,
        "mandatory": true,
        "values": [],
        "listitemType": []
      }
    ],
  }
] }
```
$\{ "serviceId": "g2gFZAAEbm9kZW0AAAAkNWISNJ1ZjItMmZkMS00MzExLTk5YWEtMzQ3N2IzMTQ0xMWQ3ZAAIZXhlY3V0b3JkAAhmb3JjZV9nY3IQA2QAEhvbWJhdEAxMjcuMC4wLjECAAA02wAAAAAAAAAA", 
"objectType": "node", 
"objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7", 
isInternal": false, 
"label": "Garbage Collect Processes", 
"description": "Force garbage collection on all processes. 
,options": [ 
{ 
"name": "GroupSize", 
"label": "Number of processes garbage collected in a bunch", 
"type": "number", 
"default": "1000", 
"enabled": true, 
"mandatory": true, 
"values": [], 
"listitemType": [] 
}, 
{ 
"name": "GroupPause", 
"label": "Pause between process GCs in millisecs", 
"type": "number", 
"default": "100", 
"enabled": true, 
"mandatory": true, 
"values": [], 
"listitemType": [] 
} 
] 
\}$

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ID</td>
<td><strong>Required.</strong> The identifier of the node.</td>
</tr>
</tbody>
</table>

**Returns**

A `service_infos` object containing the list of available services for the given node.

**Get all executor services available for a node family**

**Definition**

GET /api/service/service-info/executor/node-family/NODE_FAMILY_ID

**Example request**

```
curl -X GET "http://127.0.0.1:8000/api/service/service-info/executor/node-family/12a74b0b-2d50-4278-a90c-2b5e50c2e8c8"
```

**Example response**

```
$\{ "service_infos": [] \}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_FAMILY_ID</td>
<td>Required. The identifier of the node family.</td>
</tr>
</tbody>
</table>

Returns

A service_infos object containing the list of available services for the given node family.

Initiate a new service request

Definition

POST /api/request/service_request

Example request

curl -X POST "http://127.0.0.1:8080/api/request/service_request" \
   -H "Content-Type: application/json" \
   -d '{"id": "g2gFZAEBm9kZwOAAAkKWI5NjIIZjItMmZkM00MzExLTkyYWVtMz0N2IzMz0xMwQ2ZAMY29uZmlndXJhdG9yZAAALc2V0X2NvbmZpZ3NyAANkABB3b21iYXRAMTI3LjAuMC4xAgAANOYAAAAAAAAAAA==", "arguments": "application": "lager", "changes": [{"configParamName": "crash_log_count", "configParamValue": "10", "persist": "No"}]}'

Example response

```
{
  "requestId": "108f6333-d518-4017-8b0a-da245d1ef2a8",
  "label": "Set a configuration parameter"
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Required. The identifier of service (serviceId) as returned by the GET requests above.</td>
</tr>
<tr>
<td>arguments</td>
<td>Required. The arguments to be passed to the service. Each value should be either a string or a list containing string→string objects.</td>
</tr>
</tbody>
</table>

Returns

Information about the request that was initiated.

Initiate a feature request

When a service responds by sending data to WombatOAM (which is displayed to the user), this data may contain further actions that can be executed by the user.

Let's assume that we have a "Processes" request, whose response to the Get information about a request HTTP request contains a table of the top processes on the managed node, with some additional information.

The cells in the first column in this table are links because they provide a
local menu, which shows all the actions that can be performed in this
cell. Each such action defines a new potential request. This "initiate a
feature request" HTTP request is about initiating such a feature request.

The JSON response behind this Dashboard table is the following:

```
"data": [
  {
    "data": "<0.3.0>",
    "actions": [
      {
        "label": "Terminate process",
        "featureRequest": {
          "featureName": "kill_process",
          "objectType": "node",
          "objectId": null,
          "node": "test1@127.0.0.1",
          "arguments": "/g2wAAAAAcAjtAAAAA1BpZG0AAAHpDauMy4
wPmgCbQAAAAASZ ZwFzb25tAAAAABGtpbGxq"
        }
      },
      ... // further actions in this cell
    ],
    ... // other columns
  },
  ... // other rows
]
```

When the user selects an action, that action shall be sent to
WombatOAM using the "initiate a feature request" HTTP request without
being changed. The "arguments" parameter contains the arguments
needed by the new feature request (in this case the pid of the selected
process) in an encoded format.

**Definition**

POST /api/request/feature_request

**Example request**

```
curl -X POST "http://127.0.0.1:8080/api/request/feature_request" \
  -H "Content-Type: application/json" \
  -d '{"featureName": "set_as_global", "objectType": "node-family", "objectId": null, "node": "test1@127.0.0.1", "arguments": "g2wAAAAAcAjtAAAAAC2FwcGxpY2F0aW9uQAAAAAAAA9jb25nWddcEhjbUB5WVtAAAAAD2NyXNoX2xvZ19jb3VudGo="}'
```

**Example response**

```
{"requestId": "6dee9dab-2266-45e2-8fc8-f8fe500ae8a3",
 "label": "Set application parameter as global"
}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureName</td>
<td><strong>Required.</strong> The name of the feature to be initiated.</td>
</tr>
<tr>
<td>objectType</td>
<td><strong>Required.</strong> Either &quot;node&quot; or &quot;node-family&quot;. It specified whether the action should be performed</td>
</tr>
</tbody>
</table>
on a node or a node family.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td><strong>Required.</strong> (string or null) This field can be used to specify the id of the node or node family on which the action should be performed.</td>
</tr>
<tr>
<td>node</td>
<td><strong>Optional.</strong> (string or null) This field can be used to specify the name of the node on which the action should be performed if objectId is null. Either objectId or node has to be not null.</td>
</tr>
<tr>
<td>arguments</td>
<td><strong>Required.</strong> The arguments to be given to the feature. This field is received in an encoded format from /api/request/request-info and shall be supplied here unchanged.</td>
</tr>
</tbody>
</table>

**Returns**

Information about the request that was initiated.

**Stop a running request**

Stop a request. Note that only interruptible requests can be stopped, such as periodically listing information about the processes.

**Definition**

POST /api/request/stop/REQUEST_ID

**Example request**

curl -X POST "http://127.0.0.1:8080/api/request/stop/108f6333-d518-4017-8b0a-da245d1ef2a8" \
   -H "Content-Type: application/json"

**Example response (success)**

In case of success, the response body is empty.

**Example response (error)**

```
{
  "error": "no such child,<<"108f6333-d518-4017-8b0a-da245d1ef2a8">>"
}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_ID</td>
<td><strong>Required.</strong> The identifier of the request to be stopped.</td>
</tr>
</tbody>
</table>

**Returns**

An empty response.

**Initiate new process monitors**

Start monitoring given processes on all or given nodes. Without a list of nodes the processes will be monitored on all nodes.

**Definition**
Example requests

Version 1:
curl -X POST "http://localhost:8080/api/config/monitor"
  -H "Content-Type: application/json"
  -d '{"processes": ["process_to_monitor"]}'

Version 2:
curl -X POST "http://localhost:8080/api/config/monitor"
  -H "Content-Type: application/json"
  -d '{"processes": ["process_to_monitor"], "nodes": ["wombat@127.0.0.1"]}'

Example response (success)

In case of success, the response body contains a true value.

Example response (error)

```
{
  "error": "{missing_key, <<"Missing processes: key from body">>}"
}
```

```
{
  "error": {
    "success": false,
    "error": <<"Only admin users are allowed to do this request">>,
    "reason": not_authorized
  }
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESSES</td>
<td>Required. The list of processes must be given in order to start monitoring. List of nodes can be left out.</td>
</tr>
</tbody>
</table>

Returns

An empty response.

Get information about a request

Definition

GET /api/request/request-info/REQUEST_ID

Example request

```
curl -X GET "http://127.0.0.1:8080/api/request/request-info/0cde1cb-7d3e-4e98-9c20-86d14d9dd8e4"
```

Example response

```
{

```
"requestId": "0cb6dec7-7d3e-4e98-9c20-86d14d9dd8e4",
"label": "Processes",
"objectType": "node",
"objectId": "5b9625f2-2fd1-4311-92aa-3477b33d11d7",
"startDate": 1459016287788,
"endDate": null,
"arguments": [
  {"label": "Refresh interval in seconds", "value": "10"},
  {"label": "Number of listed processes", "value": "2"},
  {"label": "Order by column", "value": "Reds"},
  {"label": "Order direction", "value": "Descending"}],
"infoMessages": [],
"displayInfo": {
  "dataStructure": "table",
  "label": "Processes",
  "tableHeaders": [
    "Pid",
    "Name or Initial Func",
    "Reds",
    "Memory",
    "MsgQ",
    "Current Function"
  ],
  "isInteractive": true
},
"streamInfo": {
  "isPeriodic": true,
  "isInterruptible": true
},
"data": [
  {
    "data": "<0.3.0>"
  },
  {
    "label": "Terminate process",
    "featureRequest": {
      "featureName": "kill_process",
      "objectType": "node",
      "objectId": null,
      "node": "test1@127.0.0.1",
      "arguments": "g2wAAAACaJtAAAAA1BpZG0AAAAHPDAuMy4wPmgCbQAAAASwFzv25tAAAABGtpbGxq"
    }
  },
  {
    "label": "Process info",
    "featureRequest": {
      "featureName": "process_info",
      "objectType": "node",
      "objectId": null,
      "node": "test1@127.0.0.1",
      "arguments": "g2wAAAAbAjtAAAAA1BpZG0AAAAHlDAuMy4wPmo="
    }
  },
  {
    "label": "Process messages",
    "featureRequest": {
      "featureName": "process_messages",
      "arguments": "g2wAAAAbAjtAAAAA1BpZG0AAAAHlDAuMy4wPmo="
    }
  }
]
"objectType": "node",
"objectId": null,
"node": "test1@127.0.0.1",
"arguments": "g2wAAAAABaAjtAAAAA1BpZG0AAAAAHP DAuMy4wPmo="
},
{
  "label": "Process dictionary",
  "featureRequest": {
    "featureName": "process_dictionary",
    "objectType": "node",
    "objectId": null,
    "node": "test1@127.0.0.1",
    "arguments": "g2wAAAAABaAjtAAAAA1BpZG0AAAAAHP DAuMy4wPmo="
  },
  "data": 
  "services": 
  "data": 
  "actions": []
},
{
  "data": 
  "services": 
  "data": 
  "actions": []
},
{
  "data": 
  "services": 
  "data": 
  "actions": []
},
{
  "data": 
  "services": 
  "data": 
  "actions": []
}]
"data": 
"services": 
"data": 
"actions": []
},
{
  "data": 
  "services": 
  "data": 
  "actions": []
},
{
  "data": 
  "services": 
  "data": 
  "actions": []
}]
"data": 
"services": 
"data": 
"actions": []
},
{
  "data": 
  "services": 
  "data": 
  "actions": []
}]
"data": 
"services": 
"data": 
"actions": []
}]}
"node": "test1@127.0.0.1",
arguments": "g2wAAAACaAjtAAAAA1BpZG0AAAMADnMTYyNzkuMTI+aAjtAAAAA1BpZG0AAAMADnMTYyNzkuMTI+ag==",

D AuMTYyNzkuMTI+ag==

{
"label": "Process info",
"featureRequest": {
  "featureName": "process_info",
  "objectType": "node",
  "objectId": null,
  "node": "test1@127.0.0.1",
  arguments": "g2wAAAAAaAjtAAAAA1BpZG0AAAMADnMTYyNzkuMTI+ag==",
}

D AuMTYyNzkuMTI+ag==

{
  "label": "Process messages",
  "featureRequest": {
    "featureName": "process_messages",
    "objectType": "node",
    "objectId": null,
    "node": "test1@127.0.0.1",
    arguments": "g2wAAAAAaAjtAAAAA1BpZG0AAAMADnMTYyNzkuMTI+ag==",
  }
}

D AuMTYyNzkuMTI+ag==

{
  "label": "Process dictionary",
  "featureRequest": {
    "featureName": "process_dictionary",
    "objectType": "node",
    "objectId": null,
    "node": "test1@127.0.0.1",
    arguments": "g2wAAAAAaAjtAAAAA1BpZG0AAAMADnMTYyNzkuMTI+ag==",
  }
}

D AuMTYyNzkuMTI+ag==

{
  "label": "Process state",
  "featureRequest": {
    "featureName": "process_state",
    "objectType": "node",
    "objectId": null,
    "node": "test1@127.0.0.1",
    arguments": "g2wAAAAAaAjtAAAAA1BpZG0AAAMADnMTYyNzkuMTI+ag==",
  }
}

D AuMTYyNzkuMTI+ag==

{
  "label": "Process stack trace",
  "featureRequest": {
    "featureName": "process_stack_trace",
    "objectType": "node",
    "objectId": null,
    "node": "test1@127.0.0.1",
    arguments": "g2wAAAAAaAjtAAAAA1BpZG0AAAMADnMTYyNzkuMTI+ag==",
  }
}

D AuMTYyNzkuMTI+ag==

{
  "data": "{wombat_plugin,init,1}"
}

D AuMTYyNzkuMTI+ag==

{wombat_plugin,init,1}
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_ID</td>
<td>Required. The identifier of the request.</td>
</tr>
</tbody>
</table>

Returns

Information about the request, including its last stream data.

Get all running request per node

Definition

GET /api/request/node/NODE_ID

Example request

curl -X GET "http://127.0.0.1:8080/api/request/node/5b9625f2-2fd1-4311-92aa-3477b33d11d7"

Example response

```
{ "requests": [ 
    { 
      "requestId": "eb93644c-5dd8-4b53-ade2-f4024ea7b397", 
      "label": "Processes" 
    } 
  ] }
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ID</td>
<td>Required. The identifier of the node.</td>
</tr>
</tbody>
</table>

Returns

Basic information about the requests running on the given node.

Get all running request per node family

Definition

GET /api/request/node-family/NODE_FAMILY_ID

Example request

---

curl -X GET "http://127.0.0.1:8080/api/request/node-family/12a74b0b-2d50-4270-a90c-2b5e50c2e0c8"

Example response

```
{
  "requests": [
    {
      "requestId": "84d829d1-453c-4b45-aa51-c0ab670b9b98",
      "label": "List global configs"
    }
  ]
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE FAMILY ID</td>
<td>Required. The identifier of the node family.</td>
</tr>
</tbody>
</table>

Returns

Basic information about the requests running on the given node family.

Get all finished requests per node

Definition

GET /api/request/history/node/NODE_ID

Example request

curl -X GET "http://127.0.0.1:8080/api/request/history/node/5b9625f2-2fd1-4311-92aa-3477b33d11d7"

Example response

```
{
  "requests": [
    {
      "requestId": "b2558d43-f118-4f92-92ea-e9fd29de9bb7",
      "label": "Processes"
    },
    {
      "requestId": "17fa01c5-1fa2-4a41-b288-f7cf5131bc08",
      "label": "Display an application's configuration parameters"
    }
  ]
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_ID</td>
<td>Required. The identifier of the node.</td>
</tr>
</tbody>
</table>

Returns

Basic information about the requests finished on the given node family.

Get all finished requests per node family
Definition

GET /api/request/history/node-family/NODE_FAMILY_ID

Example request

curl -X GET "http://127.0.0.1:8080/api/request/history/node-family/12a74b0b-2d50-4270-a90c-2b5e50c2e0c8"

Example response

```
{
  "requests": [
    {
      "requestId": "b2558d43-f118-4f92-92ea-e9fd29de9bb7",
      "label": "Processes"
    },
    {
      "requestId": "17fa01c5-1fa2-4a41-b288-f7cf5131bc08",
      "label": "Display an application's configuration parameters"
    }
  ]
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE_FAMILY_ID</td>
<td>Required. The identifier of the node family.</td>
</tr>
</tbody>
</table>

Returns

Basic information about the requests finished on the given node.
Orchestration REST interface

Please also read the Orchestration page, especially with regards to understanding the difference between Orchestration v1 and v2.

GET requests:
- Get supported providers
- Get available providers
- Get information about a provider
- Get available releases
- Get information about a release
- Get available node families (Orchestration v1)
- Get available node families (Orchestration v2)
- Get information about a node family (Orchestration v2)
- Get available domains of a node family
- Get information about a domain
- Get available nodes (Orchestration v1)
- Get available nodes (Orchestration v2)
- Get information about a node (Orchestration v2)

POST requests:
- Register provider
- Upload release
- Create node family (Orchestration v1)
- Create node family (Orchestration v2)
- Attach release to a node family
- Set firewall configuration for a node family
- Set node hooks for a node family
- Add deployment domains to a node family
- Deploy a node
- Upload file to a node (deprecated - only Orchestration v1)

DELETE requests:
- Unregister provider
- Delete release
- Delete family (Orchestration v1)
- Delete family (Orchestration v2)
- Terminate family (Orchestration v2)
- Terminate a node

Get supported providers

Retrieve a list of all the providers supported by this WombatOAM version.

Definition
GET /api/orch/supported-provider

Example request
curl -X GET "http://127.0.0.1:8080/api/orch/supported-provider"

Example response
1
[

Arguments

None.

Returns

An array of provider objects.

Get available providers

Retrieve a list of all the available providers (i.e. providers already registered by the user).

Definition

GET /api/orch/provider

Example request

curl -X GET "http://127.0.0.1:8080/api/orch/provider"

Example response

```
[
  {
    "uuid": "d010c157-dd36-4990-b894-eecce873c09",
    "name": "EC2",
    "type": "ec2",
    "driver": "wo_orch_elibcloud_driver",
    "opts": {
      "name": "EC2",
      "username": "AKIAIWCGOGME26P7VB3A",
      "password": "GAQZqfvsIgEaIjR5900AdxtdNAsOQ3UrKpM9bqg",
      "type": "virtual",
      "vprovider": "ec2"
    }
  }
]
```

Arguments
None.

**Returns**
An array of provider objects.

**Get information about a provider**
Retrieve details of a specified provider.

**Definition**
GET /api/orch/provider/PROVIDER_UUID

**Example request**
curl -X GET "http://127.0.0.1:8080/api/orch/provider/0336fee8-d8c4-48ba-adfb-50f2e6421107"

**Example response**
```
{
  "uuid": "0336fee8-d8c4-48ba-adfb-50f2e6421107",
  "name": "Rackspace",
  "type": "rackspace",
  "driver": "wo_orch_elibcloud_driver",
  "opts": {
    "name": "Rackspace",
    "username": "myuser",
    "password": "mypassword",
    "type": "virtual",
    "vprovider": "rackspace",
    "service_region": "lon"
  }
}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>release UUID</td>
<td><strong>Required.</strong> The identifier of the release.</td>
</tr>
</tbody>
</table>

**Returns**
The release object.

**Get available releases**
Retrieve a list of all available releases.

**Definition**
GET /api/orch/release

**Example request**
curl -X GET "http://127.0.0.1:8080/api/orch/release"

**Example response**
```
[{
  "uuid": "952d97dd-c895-4f4a-8261-1f320d8d9ba1",
}
```
Arguments
None.

Returns
An array of release objects.

Get information about a release
Retrieve details of a specified release.

Definition
GET /api/orch/release/RELEASE_UUID

Example request
```
curl -X GET "http://127.0.0.1:8080/api/orch/release/952d97dd-c895-4f4a-8261-1f320d9ba1"
```

Response definition
```
{
  "uuid": "ReleaseUUID",
  "name": "ReleaseName",
  "description": "ReleaseDescription",
  "state": "UPLOADING" or "UPLOADED",
  "path": "PathToTargz",
  "templates": [ "TemplateFile"
  ],
  "cookie": "Cookie",
  "start_cmd": "StartCmd",
  "stop_cmd": "StopCmd"
}
```

Example response
```
{
  "uuid": "952d97dd-c895-4f4a-8261-1f320d9ba1",
  "name": "Riak",
  "state": "UPLOADED",
  "path": "//Users/demo/wombat/rel/wombat/wombat/data/wombat@127.0.0.1/orch/releases/952d97dd-c895-4f4a-8261-1f320d9ba1.tar.gz",
  "templates": [ "/etc/vm.args"
  ],
  "cookie": "riak",
  "start_cmd": "bin/riak start",
  "stop_cmd": "bin/riak stop"
}
```
"stop_cmd": "bin/riak stop"

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>release UUID</td>
<td>Required. The identifier of the release.</td>
</tr>
</tbody>
</table>

Returns

The release object.

Get available node families (Orchestration v1)

Retrieve a list of node families. Note that this request belongs to the Topology service. The following also shows the Orchestration-specific parts of the response.

Definition

GET /api/topo/node-family

Example request

```
curl -X GET "http://127.0.0.1:8080/api/topo/node-family"
```

Response definition

```
[
  {
    "id": "familyId",
    "name": "familyName",
    "description": [ ],
    "node_selection": "random",
    "bootstrap_strategy": [ 
      "wo_bootstrap_strategy",
      "none",
    [ 
      
    ),
    "bootstrap_strategy_opts": [ ],
    "neighbors": [ ],
    "nodes": [ ],
    "plugins_opts": { 
      "wo_orch": {
        "release": "Release ID",
        "firewall": [ 
          
        ],
        "domains": [ 
          { 
            "UUID",
            
          },
          { 
            "opts": { 
              "hwprof": "Hardware profile",
              "vdisk": "Virtual disk ID"
            }
          }
        ],
        "deleted": Boolean
```
Example response

```
[  
  {"id": "fcb0d915-0400-4515-b6f2-fadac756b550",  
    "name": "Node family name",  
    "description": [],  
    "node_selection": "random",  
    "bootstrap_strategy": "custom",  
    "bootstrap_strategy_opts": {  
      "cmd": "bin/bootstrap"
    },  
    "neighbors": [],  
    "nodes": [  
      "78286035-997d-4074-b505-7a5806d75db1",  
      "24f31cae-6e72-41d0-8dc9-1dbdcd836b",  
      "4153af69-7812-4f69-7a58-02b4f0197b220"
    ],  
    "plugins_opts": {  
      "wo_orch": {  
        "release": "393d6e6bc-99d7-43c3-a8bd-ac75ba4d70c6",  
        "firewall": {  
          "protocol": "tcp",  
          "port_from": "4369",  
          "port_to": "4369",  
          "sources": "undefined"
        },  
        "domains": [  
          {  
            "uuid": "863228a5-771a-4d65-aa78-bc17b8ec942f",  
            "provider": "e8a8f99d-dabe-4be0-4ba492b42d7d",  
            "family": "fcb0d915-0400-4515-b6f2-fadac756b550",  
            "ssh_user": "ubuntu",  
            "props": [],  
            "driver": "wo_orch_ssh_driver",  
            "capabilities": [  
              {  
                "id": "ram_size_mb",  
                "type": "value",  
                "value": 512
              },  
              {  
                "id": "disk_size_gb",  
                "type": "one_in_range",  
                "value": [1, 10]
              }
            ],  
            "opts": {  
              "hwprof": "t1.micro",  
              "vdisk": "ami-0cdf4965"
            }
          }
        ]
      }
    },  
    "deleted": false
  }
]
```

Arguments
None.

**Returns**

An array of node family objects.

**Get available node families (Orchestration v2)**

**Definition**

```
GET /api/orch/node-family
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/orch/node-family"
```

**Example response**

```
[
  {
    "id": "4512b026-e37b-401f-924e-76a2ad00e23b",
    "name": "Riak node family",
    "errors": [],
    "clusterFormed": true,
    "bootstrapNodeId": "968d375b-392a-48cd-9da6-aedf6efe3245",
    "releaseId": "9e522e8e-43bb-43de-80ec-4715ccf71f52",
    "domains": [
      {
        "id": "ac110d14-28bf-496d-8e64-62df69047a1f",
        "providerId": "0336f8ee-48c4-48ba-adfb-58f2e6421107",
        "sshUser": "myuser",
        "capabilities": [],
        "opts": {
          "hwprof": "2",
          "vdisk": "598a4282-f14b-4e50-af4c-b3e52749d9f9",
        }
      }
    ],
    "firewallRules": [
      {
        "protocol": "tcp",
        "port_from": "4369",
        "port_to": "4369",
        "sources": null
      },
      {
        "protocol": "tcp",
        "port_from": "8907",
        "port_to": "8909",
        "sources": null
      }
    ],
    "hooks": {
      "pre_node_start": [
        {
          "action": "run",
          "executable": "bin/pre-start.sh",
          "location": "node",
          "args": ["start", {"var": "node_name"}]
        }
      ]
    }
  }
]
```
Arguments
None.

Returns
An array of orchestration node family objects.

Get information about a node family
(Orchestration v2)

Definition
GET /api/orch/node-family/FAMILY_UUID

Example request
curl -X GET "http://127.0.0.1:8080/api/orch/node-family/4512b026-e37b-401f-924e-76a2ad00e23b"

Example response
```json
{
    "id": "4512b026-e37b-401f-924e-76a2ad00e23b",
    "name": "Riak node family",
    "errors": [],
    "clusterFormed": true,
    "bootstrapNodeId": "968d375b-392a-48cd-9da6-aedf6efe3245",
    "releaseId": "9e522e8e-43bb-43de-80ec-4715ccf71f52",
    "domains": [
        {
            "id": "ac110d14-28bf-496d-8e64-62df69047a1f",
            "providerId": "0336fee8-d8c4-48ba-adfb-50f2e6421107"
        },
        {
            "sshUser": "myuser",
            "capabilities": [
                {
                    "id": "cpu_count",
                    "type": "value",
                    "value": 1
                },
                {
                    "id": "packages",
                    "type": "all_in_list",
                    "value": ["erlang", "python"]
                },
                {
                    "id": "disk_gb",
                    "type": "one_in_range",
                    "value": [10, "no_limit"]
                }
            ],
            "opts": {
                "hwprof": "2",
                "vdisk": "598a4282-f14b-4e50-af4c-b3e52749d9f9"
            },
            "state": "configured"
        }
    ],
    "firewallRules": [
        {
            "protocol": "tcp",
            "port_from": "4369",
            "port_to": "4369"
        }
    ]
}
```
Arguments

None.

Returns

An orchestration node family object.

Get available domains of a node family

Definition

GET /api/orch/node-family/FAMILY_UUID/domain

Example request

curl -X GET "http://127.0.0.1:8080/api/orch/node-family/4512b026-e37b-401f-924e-76a2ad00e23b/domain"

Example response

```
[
  {
    "state": "configured",
    "uuid": "ac110d14-28bf-496d-8e64-62df69047a1f",
    "provider": "0336fee8-d8c4-48ba-adfb-50f2e6421107",
    "family": "4512b026-e37b-401f-924e-76a2ad00e23b",
    "ssh_user": "myuser",
    "props": [],
    "driver": "undefined",
    "capabilities": [],
    "opts": {
      "vdisk": "598a4282-f14b-4e50-af4c-b3e52749d9f9",
      "hwprof": "2"
    }
  }
]
```

Arguments

None.

Returns
An array of domain objects.

**Get information about a domain**

**Definition**

GET /api/orch/node-family/FAMILY_UUID/domain/DOMAIN_ID

**Example request**

curl -X GET "http://127.0.0.1:8080/api/orch/node-family/4512b026-e37b-401f-924e-76a2ad00e23b/domain/ac110d14-28bf-496d-8e64-62df69047a1f"

**Example response**

```javascript
{
  "uuid": "ac110d14-28bf-496d-8e64-62df69047a1f",
  "provider": "0336fee8-d8c4-48ba-adfb-50f2e6421107",
  "family": "4512b026-e37b-401f-924e-76a2ad00e23b",
  "ssh_user": "myuser",
  "props": [],
  "driver": "undefined",
  "capabilities": [],
  "opts": {
    "hwprof": "2",
    "vdisk": "598a4282-f14b-4e50-af4c-b3e52749d9f9"
  },
  "state": "configured"
}
```

**Arguments**

None.

**Returns**

A domain object.

**Get available nodes (Orchestration v1)**

Retrieve a list of deployed nodes.

**Definition**

GET /api/orch/node

**Example request**

curl -X GET "http://127.0.0.1:8080/api/orch/node"

**Response definition**

```javascript
[
  {
    "id": "nodeId",
    "name": "nodeName",
    "state": "UP",
    "node_family_id": "familyId",
    "host": "Host",
    "cookie": "Cookie",
    "domain_id": "Domain ID",
    "tref": "Tref",
    "deleted": Boolean
  }
]```
"plugin_opts": {
  "wo_orch": {
    "start_cmd": "Start command",
    "stop_cmd": "Stop command",
    "ssh_user": "SSH username",
    "domain": "Domain ID",
    "driver": "Driver"
  }
},
...

Arguments
None.

Returns
An array of node objects.

Get available nodes (Orchestration v2)
Retrieve a list of deployed nodes.

Definition
GET /api/orch/node

Example request
curl -X GET "http://127.0.0.1:8080/api/orch/node"

Example response
[{
  "id": "4a308dd1-3e69-41d4-9479-e9c7af891c0",
  "name": "4a308dd1-3e69-41d4-9479-e9c7af891c0@127.0.0.1",
  "nodeFamilyId": "dc5f2396-6b93-4601-a9a2-fb79ee553526",
  "cookie": "riak",
  "state": "STOPPED",
  "orchState": "stopped",
  "errors": [],
  "inCluster": false,
  "host": "127.0.0.1",
  "sshUser": "myuser",
  "startCmd": "bin/riak start",
  "stopCmd": "bin/riak stop",
  "domainId": "55fab803-c94d-48c0-9179-76bd9e02d5fc",
  "domainOpts": {},
  "providerId": "0bae6d7-0b7d-49e7-8aa3-fac6ef5e0683",
  "familyHooks": {},
  "hooks": {},
  "driver": "wo_orch_ssh_driver"
},
...
]
Arguments
None.

Returns
An array of node objects.

Get information about a node (Orchestration v2)
Retrieve a list of deployed nodes.

Definition
GET /api/orch/node/NODE_UUID

Example request
curl -X GET "http://127.0.0.1:8080/api/orch/node/4a308dd1-3e69-41d4-9479-e9c7a7df891c0"

Example response
```
{
    "id": "4a308dd1-3e69-41d4-9479-e9c7a7df891c0",
    "name": "4a308dd1-3e69-41d4-9479-e9c7a7df891c0@127.0.0.1",
    "nodeFamilyId": "dc5f2396-6b93-4601-a9a2-fb79ee55326",
    "cookie": "riak",
    "state": "STOPPED",
    "orchState": "stopped",
    "errors": [],
    "inCluster": false,
    "host": "127.0.0.1",
    "sshUser": "myuser",
    "startCmd": "bin/riak start",
    "stopCmd": "bin/riak stop",
    "domainId": "55fab803-c94d-48c0-9179-76bd9e02d5fc",
    "domainOpts": {},
    "providerId": "8bae64d7-0b7d-49e7-8aa3-fac6e5e0683",
    "familyHooks": {},
    "hooks": {},
    "driver": "wo_orch_ssh_driver"
}
```

Arguments
None.

Returns
A node object.
Register provider

Register a cloud provider in WombatOAM.

Definition

POST /api/orch/provider

Data

Request body for a physical provider

```
{  
  "name": "Provider name",
  "servers": ["127.0.0.1", "127.0.0.2"],
  "type": "physical"
}
```

Request body for Amazon EC2

```
{  
  "name": "Provider name",
  "username": "Amazon EC2 API key (generated on the Amazon web dashboard)",
  "password": "Amazon EC2 API code (same)",
  "type": "virtual",
  "vprovider": "ec2"
}
```

Request body for the HP Cloud

```
{  
  "name": "Provider name",
  "username": "HP Cloud username",
  "password": "HP Cloud password",
  "type": "virtual",
  "vprovider": "hp",
  "auth_url": "https://region-a.geo-1.identity.hpcloudsvc.com:35357/v2.0/tokens",
  "tenant_name": "Project name (shown on the HP Cloud web dashboard)",
  "service_region": "region-b.geo-1"
}
```

Request body for Rackspace

```
{  
  "name": "Provider name",
  "username": "Username",
  "password": "Password",
  "type": "virtual",
  "vprovider": "rackspace",
  "service_region": "lon"
}
```

Example request

The following example adds an Amazon EC2 provider.

curl -X POST "http://127.0.0.1:8080/api/orch/provider" \
-H "Content-Type: application/json" \
-d "{"name": "EC2", "username": "AKIAIVCY0GNE25F7VB3A"}"

Example response

```
{
  "uuid": "d010c157-dd36-4990-b894-eeccce873c09",
  "name": "EC2",
  "type": "ec2",
  "driver": "wo_orch_elibcloud_driver",
  "opts": {
    "name": "EC2",
    "username": "AKIAIVCYOGNE25F7VB3A",
    "password": "GAQXqfvs1gEaIjR470OAdztbNAsOQ2UrKbMBbqg7"
  },
  "type": "virtual",
  "vprovider": "ec2"
}
```

Arguments

Details are provided in the request bodies for the various providers, above.

Returns

The provider object.

Upload release

Upload a release to WombatOAM.

Definition

```
POST /api/orch/release
```

Data

```
{
  "name": "Name",
  "description": "Description",
  "start_cmd": "bin/riak start",
  "stop_cmd": "bin/riak stop",
  "cookie": "riak",
  "node_name_templates": ["/etc/vm.args"],
  "release": "release.tar.gz encoded with base64"
}
```

Example request

```
curl -X POST "http://127.0.0.1:8080/api/orch/release" \
  -H "Content-Type: application/json" \
  -d "{"name":"Riak", "description":"Riak release", "start_cmd":"bin/riak start", "stop_cmd":"bin/riak stop", "cookie":"riak", "node_name_templates":["/etc/vm.args"], "release":"H4sIAE3f7FMAA+3POxKCQBAPFQI6yR9gPs0chWMgvjEDvLyooapkF1J6m5uza1vatLxuSjHgse/DD+s4FDLm7/wTuie1pKgwXENMOY1DF+KlVx2e22Na91Nu9/PeXpvnk/3xxz8BAAAAAAAAADq18yDLTBACgAA==\"}"```

Example response

```
{
  "uuid": "952d97dd-c895-4f4a-8261-1f320d8d9b1a",
  "type": "virtual",
  "vprovider": "ec2"
}
```
"name": "Riak",
"state": "UPLOADING",
"path": "/Users/hj/wombat/rel/wombat/wombat/data/wombat@
127.0.0.1/orch/releases/952d97dd-c895-4f4a-8261-1f320d0d9b
al.tar.gz",
"templates": [
    "/etc/vm.args"
],
"cookie": "riak",
"start_cmd": "bin/riak start",
"stop_cmd": "bin/riak stop"
}

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><strong>Required.</strong> A custom string with which to identify the release.</td>
</tr>
<tr>
<td>description</td>
<td><strong>Optional.</strong> A custom string that can contain a more detailed string about the release.</td>
</tr>
<tr>
<td>start_cmd</td>
<td><strong>Required.</strong> WombatOAM will use this command to start the release once deployed.</td>
</tr>
<tr>
<td>stop_cmd</td>
<td><strong>Required.</strong> WombatOAM will use this command to stop the release.</td>
</tr>
<tr>
<td>cookie</td>
<td><strong>Required.</strong> The Erlang cookie that WombatOAM will use to connect to the deployed node.</td>
</tr>
<tr>
<td>node_name_templates</td>
<td><strong>Required.</strong> The path to the <code>vm.args</code> file within the release archive.</td>
</tr>
<tr>
<td>release</td>
<td><strong>Required.</strong> The release archive encoded in Base64.</td>
</tr>
</tbody>
</table>

### Returns

A release object. Note that the response is returned before the release is ready to be used (after copying, WombatOAM will perform some preparations on it). Before using a release (e.g. specifying it when creating a node family), use GET `/api/orch/release` to check that its status is `UPLOADED` (as opposed to `UPLOADING`).

### Create node family (Orchestration v1)

Create a node family, which can deploy a certain release to certain providers. In Orchestration v1, the node family can be created using the Topology API.

#### Definition

**POST /api/topo/node-family**

#### Data

```json
{
    "name": "Node family name",
    "bootstrap_node_selection": "random",
    "bootstrap_strategy": "custom",
    "bootstrap_strategy_opts": [{"cmd": "bin/bootstrap"}]
```
Create node family (Orchestration v2)

Example request

```
curl -X POST "http://127.0.0.1:8080/api/topo/node-family" \
-H "Content-Type: application/json" \
-d "{"name":"Riak Cluster","bootstrap_node_selection":"random","bootstrap_strategy":"custom","bootstrap_strategy_opts":{"cmd": "bin/bootstrap"}}"
```

Example response

```
{
  "id": "281a6ed1-e878-4ec0-8250-9355d9d17ec5",
  "name": "Riak Cluster",
  "description": [],
  "node_selection": "random",
  "bootstrap_strategy": "custom",
  "bootstrap_strategy_opts": {
    "cmd": "bin/bootstrap"
  },
  "neighbors": [],
  "nodes": [],
  "plugins_opts": [],
  "deleted": false
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><strong>Required.</strong> A custom string with which to identify the node family.</td>
</tr>
<tr>
<td>bootstrap_node_selection</td>
<td><strong>Required.</strong> When a new node joins a cluster, its bootstrap script gets another node as a parameter. This field defines how to select that other node. Currently the only option is random, which means that a random node is selected.</td>
</tr>
<tr>
<td>bootstrap_strategy</td>
<td><strong>Required.</strong> Determines what WombatOAM should do when it deploys a new node to a node family. The available options are distributed erlang, sd erlang and custom. With custom, you can specify a script to be executed in bootstrap_strategy_opts.</td>
</tr>
<tr>
<td>bootstrap_strategy_opts</td>
<td><strong>Optional.</strong> Specify what should happen when a new node is deployed into a node family. For example, to execute the script bin/bootstrap (included in the release archive), specify &quot;cmd&quot;: &quot;bin/bootstrap&quot;.</td>
</tr>
</tbody>
</table>

Returns

A node family object.
Create a node family, which can deploy a certain release to certain providers. In Orchestration v2, there is no "bootstrap strategy", so creating a node family is simpler than in v1.

**Definition**

POST /api/orch/node-family

**Data**

```
{
  "name": "Node family name"
}
```

**Example request**

```
curl -X POST "http://127.0.0.1:8080/api/orch/node-family" 
-H "Content-Type: application/json" 
-d "{"name":"Riak Cluster"}"
```

**Example response**

```
{
  "id": "281a6ed1-e878-4ec0-8250-9355d9d17ec5",
  "name": "Riak Cluster"
}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td><strong>Required.</strong> A custom string with which to identify the node family.</td>
</tr>
</tbody>
</table>

**Returns**

A node family object.

**Attach release to a node family**

Connect a node family with a release.

**Definition**

POST /api/orch/node-family/FAMILY_UUID/release

**Data**

```
{
  "release": "Release UUID"
}
```

**Example request**

```
curl -X POST "http://127.0.0.1:8080/api/orch/node-family/281a6ed1-e878-4ec0-8250-9355d9d17ec5/release" 
-H "Content-Type: application/json" 
-d "{"release":"952d97dd-c895-4f4a-8261-1f320d0d9ba1"}"
```

**Example response**

```
{
  "id": "952d97dd-c895-4f4a-8261-1f320d0d9ba1",
  "name": "Riak Cluster"
}
```
200 OK

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>family ID</td>
<td><strong>Required.</strong> The identifier of the node family.</td>
</tr>
<tr>
<td>release</td>
<td><strong>Required.</strong> The identifier of the release to be deployed on the nodes in the node family.</td>
</tr>
</tbody>
</table>

**Returns**

The HTTP status code only.

**Set firewall configuration for a node family**

Set the firewall rules that define network ports to be open on the virtual machine instances in the node family.

**Definition**

POST /api/orch/node-family/FAMILY_UUID/firewall

**Data**

The request body is an array that contains one or more rule objects.

```json
[
  {
    "protocol": "tcp",
    "port_from": "4369",
    "port_to": "4369"
  },
  {...
}
]
```

**Example request**

```
curl -X POST "http://127.0.0.1:8080/api/orch/node-family/fcb0d915-8400-4515-b6f2-fad756b550/firewall" \\  \
-H "Content-Type: application/json" \\
-d "{["protocol":"tcp","port_from":"4369","port_to":"4369"]}"
```

**Example response**

200 OK

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>family ID</td>
<td><strong>Required.</strong> The identifier of the node family.</td>
</tr>
<tr>
<td>protocol</td>
<td><strong>Required.</strong> The protocol whose network traffic to filter with this firewall rule. The options are &quot;tcp&quot; and &quot;udp&quot;.</td>
</tr>
<tr>
<td>port_from</td>
<td><strong>Required.</strong> The first in the range of port numbers to which to apply this rule.</td>
</tr>
<tr>
<td>port_to</td>
<td><strong>Required.</strong> The first in the range of port numbers to which to apply this rule.</td>
</tr>
</tbody>
</table>
Response
The HTTP status code only.

Set node hooks for a node family
Replace the set of hooks defined for a node family.

Definition
POST /api/orch/node-family/FAMILY_UUID/hook

Data

```
{
    "node_hooks": NodeHooks
}
```

Example request
```
curl -X POST "http://127.0.0.1:8080/api/orch/node-family/20718a06-3643-4d93-b291-48c08b761ce5/hook" 
   -H "Content-Type: application/json" 
   -d '{"node_hooks": {"pre node_start": [{"action": "run", "executable": "ls", "location": "node"}]}'}
```

Example response
200 OK

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_hooks</td>
<td><strong>Required</strong> An object describing which hooks shall be called on which occasions. See the detailed explanation and specification in the &quot;Node hooks&quot; section.</td>
</tr>
</tbody>
</table>

Returns
The HTTP status code only.

Add deployment domains to a node family
Add a deployment domain to a node family. A deployment domain is a part of the node family which uses the same provider with the same virtual image and hardware profile. When a node is deployed to a node family, it is actually deployed into a domain.

If the domain is a cloud domain (i.e. it will deploy nodes on hosts requested from a cloud provider like Amazon EC2, HP Cloud or Rackspace), then an SSH key and possibly a security group are created on the cloud provider during the execution of this request.

In Orchestration v2, the domain object has a "configured" field. The domain is ready to be used when that field is set to "true". The node family can still receive commands about node deployments, and the deployments will automatically start once the domain to be used has been configured.

Definition
POST /api/orch/node-family/FAMILY_UUID/domain
Example request body

```json
{
  "provider": "e8a8f99d-dabe-4be0-4ba492b42d7d",
  "ssh_user": "ubuntu",
  "vdisk": "ami-0cdf4965",
  "hw_profile": "t1.micro",
  "type": "virtual",
  "capabilities": [  
    {
      "id": "ram_size_mb",
      "type": "value",
      "value": 512,
    },
    {
      "id": "disk_size_gb",
      "type": "one_in_range",
      "value": [10000, 100000]
    }
  ]
}
```

**Note:** The vdisk and hw_profile fields are not used for physical providers.

Example request

```bash
curl -X POST "http://127.0.0.1:8080/api/orch/node-family/fcb0d915-0400-4515-6b2f-fdbac75b550/domain" 
  -H "Content-Type: application/json" 
  -d "{"provider": "e8a8f99d-dabe-4be0-4ba492b42d7d", 
    "ssh_user": "ubuntu", 
    "vdisk": "ami-0cdf4965", 
    "hw_profile": "t1.micro", 
    "type": "virtual",
    "capabilities": [{
      "id": "ram_size_mb",
      "type": "value",
      "value": 512
    },
    {
      "id": "disk_size_gb",
      "type": "one_in_range",
      "value": [10000, 100000]}
  ]"
```

Example response

```json
{
  "uuid": "d8f3baf8-d37f-4981-9c32-89761988b89",
  "provider": "90ec86ae-bfe5-4010-b37b-8d0c26b569fe",
  "family": "dc3f8e09-d9b0-4d79-94c6-0a3e2bf8cfe2",
  "ssh_user": "ubuntu",
  "props": [],
  "driver": "wo_orch_ssh_driver",
  "capabilities": ["wo_orch_capability","ram_size_mb","value",512],
  ["wo_orch_capability","disk_size_gb","one_in_range","uuid0001u0002"]
}
```
```json
{
    "uuid": "e5ac62f4-5ed3-48f1-ae4a-6a281dc20f57",
    "provider": "e8a8f99d-dabe-4be0-a1b0-4ba492b42d7d",
    "family": "9542cf8a-192b-4e36-a84c-9959ad8396b4",
    "ssh_user": "ubuntu",
    "props": [],
    "driver": "wo_orch_ssh_driver",
    "capabilities": [
        {
            "id": "ram size mb",
            "type": "value",
            "value": 512
        }
    ],
    "opts": {
        "hwprof": "t1.micro",
        "vdisk": "ami-0cdf4965"
    }
}
```

**Note:** Ranges are converted into JSON in a weird way. This will be fixed in the future.

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>family ID</td>
<td><strong>Required.</strong> The identifier of the node family.</td>
</tr>
<tr>
<td>provider</td>
<td><strong>Required.</strong> The identifier of the provider.</td>
</tr>
<tr>
<td>provider</td>
<td><strong>Required.</strong> The infrastructure provider used by the domain. Each domain uses one provider.</td>
</tr>
<tr>
<td>ssh_user</td>
<td><strong>Required.</strong> The user with which WombatOAM can SSH into virtual machine instances created by this provider.</td>
</tr>
<tr>
<td>vdisk</td>
<td><strong>Required.</strong> Virtual disk. The system image that should be placed in the virtual machine instance.</td>
</tr>
<tr>
<td>hw_profile</td>
<td><strong>Required.</strong> The hardware profile of the virtual machine instances. Different providers use different terminology: Amazon calls them instance types; OpenStack, HP Cloud and Rackspace calls them flavors; Libcloud calls them sizes.</td>
</tr>
<tr>
<td>type</td>
<td><strong>Required.</strong> &quot;virtual&quot; or &quot;physical&quot;.</td>
</tr>
<tr>
<td>capabilities</td>
<td><strong>Optional.</strong> An array of objects with the following properties: id, type (&quot;value&quot;, &quot;all_in_list&quot;, &quot;one_in_list&quot;, &quot;all_in_range&quot; or &quot;one_in_range&quot;), and value. The possible type of value field depends on the value of the type field. If type is &quot;value&quot;, then value can be any JSON value. If type is &quot;all_in_list&quot; or &quot;one_in_list&quot;, then value should be a list of JSON values. If type is &quot;all_in_range&quot; or &quot;one_in_range&quot;, then value should be a list with two elements, a minimum and a maximum. Both can be either a number or the string &quot;no_limit&quot;, which stands for minus infinity when used as the minimum, and stands for plus infinity when used as the maximum.</td>
</tr>
</tbody>
</table>
The behaviour of capabilities is described in detail in a document called Capability-matching Heterogeneous deployment of a Load Testing Tool.

Returns
A deployment domain object.

Deploy a node
Deploy nodes to the configured providers.

Definition

```
POST /api/orch/node-family/FAMILY_UUID/node
```

Data

```
{
  // Mandatory fields
  "amount": NumberOfNodesToBeDeployed,

  // Optional fields
  "autostart": Boolean,
  "autojoin": Boolean, // Only for Orchestration
  "demands": [ { "id": "CapabilityIdentifier", "type": "value" OR "all_in_list" OR "one_in_list" OR "all_in_range" OR "one_in_range", "value": "CapabilityValue", "nature": "mandatory" OR "optional" } ],
  "node_name_base_template": TemplateString,
  "node_hooks": NodeHooks
}
```

Request body example

```
{
  "amount": 2,
  "autostart": true,
  "demands": [ { "id": "ram_size_mb", "type": "value", "value": 512, "nature": "mandatory" }, { "id": "provider", "type": "value", "value": "dc3f8e09-d9b0-4d79-94c6-fec3e2bf8cf2", "nature": "optional" } ],
  "node_name_base_template": "mynode-{{index}}",
  "node_hooks": { 
    // When the node is started, run the node-started script with the node
    "post_node_start": [ 
      { "action": "run", "executable": "/home/user/node-started.sh", "location": "wombat", "args": [ { "var": "node_name" } ] } ]
  }
}
```

Example request

```
curl -X POST "http://127.0.0.1:8080/api/orch/node-family/fcb0d9"
```

Example response (Orchestration v1)

```json
[{
    "id": "d1821de5-f300-4148-beff-a87f85743019",
    "name": "undefined",
    "node_family_id": "dc3f8e09-d9b0-4d79-94c6-fec3e2bf8cf",
    "host": "undefined",
    "cookie": "undefined",
    "domain_id": "undefined",
    "state": "UNINIT",
    "tref": "undefined",
    "deleted": false,
    "plugins_opts": []
},
{
    "id": "1dd0ffd6-137b-4437-9031-e9fda8780610",
    "name": "undefined",
    "node_family_id": "dc3f8e09-d9b0-4d79-94c6-fec3e2bf8cf",
    "host": "undefined",
    "cookie": "undefined",
    "domain_id": "undefined",
    "state": "UNINIT",
    "tref": "undefined",
    "deleted": false,
    "plugins_opts": []
}]
```

Example response (Orchestration v2)

In case of success, the ids of the new nodes is returned (with status code 200):

```json
[ "72dbc9ef-429d-4a58-b324-5e36da8b5940",
  "a98b9a55-1459-4c01-8021-ad99c1c9c657"
]
```

In case of error, the error is returned (with status code 400):

```json
[{"error": "no_satisfying_domain_found"}]
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>family UUID</td>
<td>Required. The identifier of the node family to deploy the node into.</td>
</tr>
<tr>
<td>amount</td>
<td>Required. The number of nodes to deploy.</td>
</tr>
<tr>
<td>autostart</td>
<td><strong>Optional.</strong> Boolean. If true, WombatOAM will not only copy the release archive file to the instances, but it will also start the nodes.</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>autojoin</td>
<td><strong>Optional.</strong> Boolean. If true, WombatOAM will not only start the nodes but make them join into a cluster.</td>
</tr>
<tr>
<td>demands</td>
<td><strong>Optional.</strong> An array of objects describing capabilities, each with the following properties: <strong>id</strong>, <strong>type</strong>, <strong>value</strong>, and <strong>nature</strong>. The objects might also have the following priority: <strong>maximize_method</strong> (default: match), <strong>priority</strong> (default: 0) and <strong>multiplier</strong> (default: 1).</td>
</tr>
<tr>
<td>node_name_base_template</td>
<td><strong>Optional.</strong> A string from which the base name of the nodes is calculated. See the detailed explanation below.</td>
</tr>
<tr>
<td>node_hooks</td>
<td><strong>Optional</strong> An object describing which hooks shall be called on which occasions. See the detailed explanation and specification below in the &quot;Node hooks&quot; section.</td>
</tr>
</tbody>
</table>

The behaviour of demands is described in detail in a document called Capability-matching Heterogeneous deployment of a Load Testing Tool.

**Returns**

In case of Orchestration v1: The list of nodes for which the deployment process was started. This may be less than the number of nodes requested. If no nodes were deployed because the deployment demands weren't satisfied by any domain, the response is an empty list.

In case of Orchestration v2: The ids of the nodes for which the deployment process was started. There are always the same amount of nodes that the user asked for -- i.e. the deployment is started for either all nodes or none of them. In the latter case (if the deployment demands weren't satisfied by any domain, or the licence doesn't allow the number of nodes the user wanted to start), an error JSON is reported.

**Node name base templates**

When deploying a set of nodes with WombatOAM, the user may specify a node name base template. The actual node names are generated from this template. E.g. if the template is `node-{{index}}`, and 3 nodes are deployed on hostA, hostB and hostC, then the names of the nodes will be `node-1@hostA`, `node-2@hostB` and `node-3@hostC`.

The node name base template might contain some patterns which are expanded. All other characters will go into the node names unmodified. The following patterns are accepted:

- `{{node_id}}`: Will be replaced with the identifier of the node.
- `{{index}}`: When the user deploys N nodes, this pattern is replaced with the index of the node (which goes from 1 to N by default). So if e.g. 3 nodes are deployed with template `node-{{index}}`, the generated node name bases will be `node-1`, `node-2` and `node-3`.

Node hooks

When deploying a set of nodes with WombatOAM, the user may specify a set of actions that need to be executed before/after certain events happen. Each such occasion where WombatOAM is able to execute a list of custom actions is called a hook.

This section starts with explaining hooks with examples. The "Specifying the hooks and actions to WombatOAM" subsection contains the exact specification of how hooks can be defined to WombatOAM.

Hooks

There are six hooks: pre_node_start, post_node_start, pre_node_stop, post_node_stop, join and leave. Their names describe when they fire, e.g. the actions assigned to the pre_node_start hook will be executed when the user issues WombatOAM to start a node, but before it is actually started. Both families and nodes may have actions assigned to hooks.

Note that these are all "node hooks", even those that are assigned to the family, so they fire when any node of the family is started or stopped, and not when the family itself is created or deleted.

Starting and stopping nodes

When a node is started, the actions are executed in the following order:

- pre_node_start hook actions that are assigned to the family of the node.
- pre_node_start hook actions that are assigned to the node.
- The node is started by executing the start command.
- post_node_start hook actions that are assigned to the family of the node.
- post_node_start hook actions that are assigned to the node.

When a node is stopped, the actions are executed in the following order:

- pre_node_stop hook actions that are assigned to the node.
- pre_node_stop hook actions that are assigned to the family of the node.
- The node is stopped by executing the stop command.
- post_node_stop hook actions that are assigned to the node.
- post_node_stop hook actions that are assigned to the family of the node.

When multiple nodes are deployed, this is performed in parallel, so the order of hooks running on different nodes is not deterministic (they may happen to run actually in parallel).

Nodes joining and leaving the cluster (in Orchestration v2)
When a node **joins** a cluster, the actions are executed in the following order:

- join hook actions that are assigned to the family of the node.
- join hook actions that are assigned to the node.

When a node **leaves** the cluster, the actions are executed in the following order:

- leave hook actions that are assigned to the node.
- leave hook actions that are assigned to the family of the node.

Note that there is no "join command" or "leave command", so these hooks are themselves responsible for asking the nodes to join/leave the cluster, and return only when that operation has finished.

The user may specify the "autojoin" option when deploying the nodes. In that case, when the first node of the node family is started, it will be appointed as the "bootstrap" node, and it will be declared a part of the cluster without executing anything. For each successive node that is started, the "join" is performed towards the bootstrap node (i.e. the join hooks are executed with the `bootstrap_node_id` and `bootstrap_node_name` variables set accordingly). The "join" operations are performed sequentially (i.e. join for a node is started only if the join for the previous node has been finished).

Note that in the future, it should be possible to initiate a node joining/leaving the cluster by sending a REST request, but these are not yet implemented. So the only way to get a node join a cluster is to use "autojoin", and there is no way to make a node leave a cluster.

**Actions**

There are three kinds of actions: **run**, **call** and **send_file**.

In each case, the output of the actions is logged in WombatOAM's debug log.

If the action shall be executed on a node, and the node is not (yet) available, then the action is retried (10 times by default, with waiting 30 seconds between retries).

**"Run" action**

This action runs a program (either a script or an executable binary). The following options are available:

- **executable** (mandatory): The path to the file that should be executed. If the script is executed on the node, this path can be either absolute or relative to the release directory (`~/<node_id>/`). If it is executed on WombatOAM, it shall be absolute.
- **location** (mandatory): Whether the script should be executed on the WombatOAM host or the managed node's host.
- **args** (optional): The arguments to be passed to the executable. Each argument can be either a string (which is passed unmodified), the variable `node_id` or `node_name` (whose value is passed) or a concatenation of strings and variables. In case of the "join" and "leave" hooks, the variables `bootstrap_node_id` and `bootstrap_node_name` can also be used.

Running a script on a remote node is performed using SSH.

As an example, if the following action is executed on node with id `d010c157-dd36-4990-b894-eeccee873c09` and name `mynode@127.0.0.1`
then the following will be executed:

```
script.sh "node started" d010c157-dd36-4990-b894-eeccce873c09
--log-dir d010c157-dd36-4990-b894-eeccce873c09/log/my
node@127.0.0.1
```

The return code of the script is checked and if it is not zero, the execution is considered to be failed and further actions on the node are cancelled.

"Call" action

This action calls an Erlang function. The following options are available:

- **mod_name, fun_name** (mandatory): The module and name of the function to be called.
- **location** (mandatory): Whether the function should be called on the WombatOAM node or the managed node.
- **args** (optional): The arguments to be passed to the function. Each argument can be either a term (which is passed unmodified after being parsed from the JSON string) or the variable node_id or node_name (whose value is passed). In case of the "join" and "leave" hooks, the variables bootstrap_node_id and bootstrap_node_name can also be used.

Calling a function on a remote node is performed using Erlang RPC.

As an example, if the following action is executed on node with id d010c157-dd36-4990-b894-eeccce873c09 and name mynode@127.0.0.1

```
{"action": "call",
"mod_name": "mymod",
"fun_name": "myfun",
"location": "node",
"args": ["node_start",
{"var": "node_name"},
{"var": "node_id"}]
}
```

then the following function call will be executed:

```
mymod:myfun(node_start,
[1, [2, 3]],
'mynode@127.0.0.1',
"d010c157-dd36-4990-b894-eeccce873c09").
```

In order to execute an Erlang function on WombatOAM, WombatOAM needs to find the containing module. To do that, the directory that contains the compiled version of the module shall be added to the "wo_orch/hook_dirs" option. For example:

```
{wo_orch, [hook_dirs, ["/home/user/my_wombat_hook_dir"]]
}
```
In order to execute an Erlang function on the deployed node, the creator of the release package needs to make sure that the node will find the module.

Since Erlang functions on the node can be executed only when the node is running, "call" actions on nodes should not be specified in \texttt{pre\_node\_start} and \texttt{post\_node\_stop} hook.

\textbf{"Send file" action}

This action copies a file from WombatOAM to the node's host machine. The following options are available:

- \texttt{source} (mandatory): The path to the file to be copied on the WombatOAM host. It is recommended to use an absolute path.
- \texttt{target} (mandatory): The path that the file shall have on the target machine, or the directory into which it should be copied. It can be either relative or absolute. In the latter case, it is relative to the node's main directory (which is \texttt{~/<node\_id>}).

Copying a file to the node is performed using SCP.

\textbf{Specifying the hooks and actions to WombatOAM}

The exact structure of the \texttt{NodeHooks} JSON object is the following:

```
NodeHooks =
{
    // All fields are optional
    "pre\_node\_start": [HookAction],
    "post\_node\_start": [HookAction],
    "pre\_node\_stop": [HookAction],
    "post\_node\_stop": [HookAction],
    "join": [HookAction],
    "leave": [HookAction]
}

HookAction =
    RunHookAction OR // Hook that runs a script
    CallHookAction OR // Hook that calls an Erlang functi
    on
    SendFileHookAction // Hook that copies a file to the n
    ode

RunHookAction =
{
    // Mandatory fields
    "action": "run",
    "executable": "PathToScript",
    "location": "wombat" OR "node",
    // Optional fields
    "args": [RunArg]
}

RunArg =
    "ArgString" OR
    {"var": "node\_id"} OR
    {"var": "node\_name"} OR
    {"concat": [RunArg]}

CallHookAction =
{
    // Mandatory fields
    "action": "call",
    "mod\_name": "ModuleName",
    "fun\_name": "FunctionName",
    "location": "wombat" OR "node",
```
Upload file to a node (deprecated - only Orchestration v1)

Upload a file to a specified location on a node.

This feature is deprecated and will be removed. If you wish to upload files to the nodes, ask the hostnames from WombatOAM and use `scp`.

**Definition**

POST /api/orch/node/NODE_UUID/upload-file

**Data**

{  
  "path": "TARGET_FILE_PATH",  
  "content": CONTENT
}

**Example response**

200 OK

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>Required. The location on the target node to which the file will be copied.</td>
</tr>
<tr>
<td>content</td>
<td>Required. The base64-encoded version of the content of the file to be uploaded.</td>
</tr>
</tbody>
</table>

**Returns**

The HTTP status code only.

**Unregister provider**

Remove an infrastructure provider from the list of available providers.

**Definition**

DELETE /api/orch/provider/PROVIDER_UUID
Example request

curl -X DELETE "http://127.0.0.1:8080/api/orch/provider/a9ccf27a-056a-4f5a-9112-abaab9c35602"

Example response

200 OK

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>provider UUID</td>
<td>Required. The identifier of the infrastructure provider.</td>
</tr>
</tbody>
</table>

Returns

The HTTP status code only.

Delete release

Delete a release.

Definition

DELETE /api/orch/release/RELEASE_UUID

Example request

curl -X DELETE "http://127.0.0.1:8080/api/orch/release/8827fe5a-e392-4c1d-8c55-2594c88685ce"

Example response

200 OK

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>release UUID</td>
<td>Required. The identifier of the release.</td>
</tr>
</tbody>
</table>

Returns

The HTTP status code only.

Delete family (Orchestration v1)

Delete a node family (using the Topology API).

Definition

DELETE /api/topo/node-family/FAMILY_UUID

Example request

curl -X DELETE "http://127.0.0.1:8080/api/topo/node-family/131b0f3-72ab-4bfb-a00d-4a6f171607ab"

Example response

200 OK
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>family UUID</td>
<td>Required. The identifier of the node family.</td>
</tr>
</tbody>
</table>

Returns

The HTTP status code only.

Delete family (Orchestration v2)

Delete a node family (using the Orchestration API). This includes releasing all resources associated with the deployment domains in the cloud (SSH keys and security groups).

If the family has nodes, an error is returned.

Definition

DELETE /api/orch/node-family/FAMILY_UUID

Example request

curl -X DELETE "http://127.0.0.1:8080/api/orch/node-family/131b0f3-72ab-4bfb-a00d-4a6f171607ab"

Example response

200 OK

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>family UUID</td>
<td>Required. The identifier of the node family.</td>
</tr>
</tbody>
</table>

Returns

The HTTP status code only.

Terminate family (Orchestration v2)

Delete a node family (using the Orchestration API). If the family has nodes, terminate them.

Definition

DELETE /api/orch/node-family/FAMILY_UUID/brutal

Example request

curl -X DELETE "http://127.0.0.1:8080/api/orch/node-family/131b0f3-72ab-4bfb-a00d-4a6f171607ab/brutal"

Example response

200 OK
family UUID  Required. The identifier of the node family.

Returns
The HTTP status code only.

Terminate a node
Terminate a deployed node. If the node is running in the cloud, this includes releasing the host machine.

Definition
DELETE /api/orch/node/NODE_UUID

Example request
curl -X DELETE "http://127.0.0.1:8080/api/orch/node/b15cc69d-bd38-4a96-affc-e6884df18d2b"

Example response
200 OK

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>family UUID</td>
<td>Required. The identifier of the deployed node.</td>
</tr>
</tbody>
</table>

Returns
The HTTP status code only.
Plugins REST interface

- Get node plugins state
- Set node all plugins state
- Set node plugin state

Get node plugins state

Retrieve the status of all plugins of a managed node in WombatOAM.

Definition

GET /api/plugins/node/NODE_ID/plugin

Example request

curl -v -X GET "http://10.211.55.14:8080/api/plugins/node/298548c9-f5f8-464d-9a2c-07d494c3ecd4/plugin"

Example response

```
{
  "plugins": [
    {
      "plugin_name": "wombat_plugin_kernel",
      "plugin_apps": "kernel",
      "plugin_state": "on"
    },
    {
      "plugin_name": "wombat_plugin_code_tracer",
      "plugin_apps": "kernel",
      "plugin_state": "off"
    }
  ]
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node from which to retrieve plugin state.</td>
</tr>
</tbody>
</table>

Returns

An array of plugin objects. The plugin objects contain the plugin_name, the plugin_apps, and the plugin_state, which indicates whether a plugin is switched on or off.

Set node all plugins state

Set the state of all plugins on a node to either on or off.

Definition

PUT /api/plugins/node/NODE_ID/plugin

Example request

```json
{
    "plugins":{
        "wombat_plugin_kernel",
        "wombat_plugin_code_tracer"
    }
}
```

## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node on which to set the plugin state.</td>
</tr>
<tr>
<td>plugin state</td>
<td>Required. The intended plugin state (on or off).</td>
</tr>
</tbody>
</table>

## Returns

An array of plugin objects. The plugin objects contain the `plugin_name`, the `plugin_apps`, and the `plugin_state`, which indicates whether a plugin is switched on or off.

## Set node plugin state

Set a specific node plugin state to either on or off.

### Definition

**PUT** /api/plugins/node/NODE_ID/plugin/PLUGIN_ID

### Example request

```bash
```

### Example response

```json
[
    {
        "plugin_name": "wombat_plugin_code_tracer",
        "plugin_apps": "kernel",
        "plugin_state": "off"
    }
]
```

## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>node ID</td>
<td>Required. The identifier of the node on which to set the plugin state.</td>
</tr>
<tr>
<td>plugin ID</td>
<td>Required. The identifier of the plugin for which to set the state.</td>
</tr>
<tr>
<td>plugin state</td>
<td>Required. The intended plugin state (on or off).</td>
</tr>
</tbody>
</table>
Returns

Updated details of the plugin, showing the new plugin state.
**Bookmark REST interface**

- Create bookmarks
- Read all bookmarks
- Update bookmarks
- Delete bookmarks

**Create bookmarks**

Create bookmark entries in WombatOAM, by sending a JSON array in the "bookmark" JSON object, all of which is within the HTTP request body. This array can contain one or more bookmark entries to be created.

**Definition**

<table>
<thead>
<tr>
<th></th>
<th>PUT /api/bookmark</th>
</tr>
</thead>
</table>

**Example request**

```
curl -u admin:admin --insecure -v -H "Content-Type:application/json" -X PUT "https://127.0.0.1:8443/api/bookmark" -d "{"bookmarks": [{"title": "title","url": "url1"},{"title": "title2","url": "url2"}]}"
```

**Example request body**

```json
{
    "bookmarks": [
        {
            "title": "My-Bookmark",
            "url": "https://localhost:8443/#/metrics?f=0&n=0&mtype=default&metrics=&mcat=1",
            ...
        },
        ...
    ]
}
```

**Example response**

```json
{
    "bookmarks": [
        {
            "title": "My-Bookmark",
            "url": "https://localhost:8443/#/metrics?f=0&n=0&mtype=default&metrics=&mcat=1",
            ...
        },
        ...
    ]
}
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bookmarks</td>
<td>Required</td>
</tr>
</tbody>
</table>

**Returns**

All the bookmarks currently created in WombatOAM. The JSON response is a "bookmarks" object with an array of bookmark arrays.

**Read all bookmarks**
Retrieve all the existing bookmarks created in WombatOAM.

**Definition**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GET /api/bookmark</td>
</tr>
</tbody>
</table>

**Example request**

```
curl -u admin:admin --insecure -v /api/bookmarks -X GET "https://127.0.0.1:8443/api/bookmark"
```

**Example response**

```
{
   "bookmarks": [
      {
         "title": "My-Bookmark",
         "url": "https://localhost:8443/#/metrics?f=0&n=0&mtype=default&metrics=&mcat=1",
         {"...": "...",
          ...
      },
      ...
   ]
}
```

**Returns**

All the bookmarks currently created in WombatOAM. The JSON response is a "bookmarks" object with an array of bookmark arrays.

**Update bookmarks**

Update/create one or more bookmark entries in WombatOAM.

**Definition**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>POST /api/bookmark</td>
</tr>
</tbody>
</table>

**Example request**

```
curl -u admin:admin --insecure -v -H "Content-Type:application/json" -X POST "https://127.0.0.1:8443/api/bookmark" -d "{"bookmarks": [{"title": "Newtitle","url": "Newurl 1"},{"title": "title2","url": "url2"}]"}
```

**Example request body**

```
{
   "bookmarks": [
      {
         "title": "My-Bookmark",
         "url": "https://localhost:8443/#/metrics?f=0&n=0&mtype=default&metrics=&mcat=1",
         {"...": "...",
          ...
      },
      ...
   ]
}
```

**Example response**

```
{
   "bookmarks": [
      {
         "title": "My-Bookmark",
         "url": "https://localhost:8443/#/metrics?f=0&n=0&..."}
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bookmarks</td>
<td>Required.</td>
</tr>
</tbody>
</table>

Returns

All the bookmarks currently created in WombatOAM. The JSON response is a "bookmarks" object with an array of bookmark arrays.

Delete bookmarks

Delete one or more bookmark entries in WombatOAM.

Definition

```
DELETE /api/bookmark
```

Example request

```
               {"..."}],
               ...
            }
            "}"
```

Example request body

```
{
   "bookmarks":[
      {
         "title":"My-Bookmark",
         "url": "https://localhost:8443/#/metrics?f=0&n=0&mtype=default&metrics=&mcat=1"},
      {
         ...
      }
   ]
}
```

Example response

```
{
   "bookmarks":[
      {
         "title":"My-Bookmark",
         "url": "https://localhost:8443/#/metrics?f=0&n=0&mtype=default&metrics=&mcat=1"},
      {
         ...
      }
   ]
}
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bookmarks</td>
<td>Required.</td>
</tr>
</tbody>
</table>
Returns

All the bookmarks currently created in WombatOAM. The JSON response is a "bookmarks" object with an array of bookmark arrays.
Front page REST interface

- Get metadata about front pages
- Get metadata about front pages owned by a user
- Get front page by name
- Update existing front page
- Create new front page
- Delete existing front page
- Front page JSON

Get metadata about front pages

Retrieve information about all the available front pages.

**Definition**

```
GET /api/front-page
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/front-page"
```

**Example response**

```
[{
  "user": "$default_user",
  "front_page_name": "Default front page"
},
{}]
```

**Arguments**

None.

**Returns**

An array of front page objects.

Get metadata about front pages owned by a user

Retrieve information about all the available front pages owned by a specified user.

**Definition**

```
GET /api/front-page?user=USER_NAME
```

**Example request**

```
curl -X GET "http://127.0.0.1:8080/api/front-page?user=%24default_user"
```

**Example response**

```
[{
  "user": "$default_user",
}
]
"front_page_name": "Default front page"

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>Required. The string value of the username.</td>
</tr>
</tbody>
</table>

Returns

An array of front page objects owned by the specified user.

Get front page by name

Retrieve a front page.

Definition

GET /api/front-page/FRONTPAGE_NAME

Example request

curl -X GET "http://127.0.0.1:8080/api/front-page/Default%20front%20page"

Example response

```
[
  {
    "items": [...],
    "name": "Memory usage",
    "type": "metric",
    "subType": "simple-numeric",
    "size": 4
  },
  {...},
  {...}
]
```

Arguments

None.

Returns

A front page object if a valid name was provided.

Create front page

Create a new front page entry in WombatOAM. For more details of the front_page value read the Front page JSON section.

Definition

PUT /api/front-page

Data

```
Example request

```bash
curl -X PUT "http://127.0.0.1:8080/api/front-page" \
-H "Content-Type: application/json" \
-d "{"front_page_name":"My new front page name", "front_page":[]}"
```

Example response

200 OK

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>front_page_name</td>
<td>Required. The name of the front page to be created.</td>
</tr>
<tr>
<td>front_page</td>
<td>Required. The front page to be created.</td>
</tr>
</tbody>
</table>

Returns

In case of success, the HTTP response is 200 OK, and the new front page is stored.

Otherwise, HTTP 400 is sent with an error object:

```json
{"error": "ERROR_REASON_STR"}
```

Update front page

Update an existing front page entry in WombatOAM. For more details of the front_page value read the Front page JSON section.

Definition

**POST /api/front-page**

Data

```json
{
  "old_page_name": "Original page",
  "new_page_name": "My new front page name",
  "front_page": [..]
}
```

Example request

```bash
curl -X POST "http://127.0.0.1:8080/api/front-page" \
-H "Content-Type: application/json" \
-d "{"old_page_name":"Original page", "new_page_name":"My new front page name", "front_page":[]}"
```

Example response

200 OK

Arguments
### Front page REST interface

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oldtypename</td>
<td><strong>Required.</strong> The old name of the front page to be updated.</td>
</tr>
<tr>
<td>typename</td>
<td><strong>Optional.</strong> The new name of the front page to be updated.</td>
</tr>
<tr>
<td>typename</td>
<td><strong>Required.</strong> The front page to be updated.</td>
</tr>
</tbody>
</table>

### Returns

In case of success, the HTTP response is 200 OK and the new front page is stored.

Otherwise, HTTP 400 is sent with an error object:

```json
{ "error": "ERROR_REASON_STR" }
```

## Delete front page

Delete an existing front page entry in WombatOAM.

### Definition

DELETE /api/front-page/FRONTPAGE_NAME

### Example request

```
curl -X DELETE "http://127.0.0.1:8080/api/front-page/my%20page"
```

### Example response

```
[
  {
    "user": "$default_user",
    "front_page_name": "Default front page"
  },
  {
  },
  {
  }
]
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typename</td>
<td><strong>Required.</strong> The name of the front page to be deleted.</td>
</tr>
</tbody>
</table>

### Returns

An array of front page objects.

## Front page JSON

This section specifies a JSON structure called "front page JSON object". This structure is returned in the response of a get front page request. Also, when adding a new page or updating an existing one, a front page JSON object needs to be specified to describe how the page should look like.

The front page is a list of panels. An empty list represents a front page
Each panel is a JSON object with the following mandatory fields:

- **items**: Describes the contents of the panel.
- **name**: The name of the panel.
- **type**: The type of the panel can be list or metric.
- **subType**: If type is list, then subType can be alarms or nodes. If type is metric, subType can be simple-numeric, live-numeric or histogram.
- **size**: The width of the panel.

If the panel's subType is simple-numeric or histogram, then there should be an extra field: **series**. It defines the time interval of the metric data. The valid values are the following: 1, 2, 3, 5, 7.

Each item has 3 mandatory fields:

- **id**: The identifier of the item.
- **type**: The type of the item. Valid values are: node, nodeFamily, all.
- **name**: The name of the item.

If an item is a metric, then it have to have a 4th field. The key is metric and the value is an object that describes the metric. It has 6 fields:

- **name**: The name of the metric.
- **type**: Valid types are counter and gauge.
- **display**: This text will appear on the dashboard.
- **unit**: Valid units are byte, numeric and percentage.
- **origin**: The metric group's name.
- **active**: Boolean value.
System configuration REST interface

Get default values from sys.config

Retrieve the current default values in the system configuration file.

Definition

GET /api/system

Example request

curl -X GET "http://127.0.0.1:8080/api/system"

Example response

```json
{
  "enableOrch": true,
  "orchVersion": "v1",
  "enableTopoGraph": true,
  "enablePopupMessages": true,
  "enableForceSSL": false,
  "forceWebsocketSSL": false
}
```

Example responses if licence expired

```json
{
  "enableOrch": true,
  "orchVersion": "v1",
  "enableTopoGraph": true,
  "enablePopupMessages": true,
  "enableForceSSL": false,
  "forceWebsocketSSL": false,
  "expiryInfo": "allowed with warning",
  "expiryText": "The license expired on 2015-12-31. Wombat OAM will stop working after 2016-01-31. Please renew your licence by contacting your Erlang Solutions sales representative or wombat@erlang-solutions.com."
}
```

Arguments

None.

Returns

An object with the current configuration settings.
Help/about REST interface

Get license text

Retrieve the WombatOAM licence text

Definition

GET /api/help/about

Example request

curl -X GET "http://127.0.0.1:8080/api/help/about"

Example response

```
{"text": "This is WombatOAM, an operations and maintenance tool for Erlang systems ..."}
```

Arguments

None.

Returns

The full WombatOAM licence text. Only the first line is shown in the example response.
WombatOAM debug script

This script is to be used to collect some information about a running WombatOAM instance, as a tarred file of the name debug_<current_date_and_time>.tar.gz into a directory called debug. Following are the information collected:

- **logs**: All the files under the directory rel/wombat/wombat/log. The source directory structure is preserved.

- **data**: The disk usage information of all the files under the directory rel/wombat/wombat/data will be stored in a file called data_file_sizes.

- **config**: The entire directory rel/wombat/files. The source directory structure is preserved. The following vm.args files are copied as well
  - rel/common/vm.args is copied as vm.args.template.
  - rel/wombat/wombat/releases/pre-2.0.1/vm.args is copied as vm.args.

- **system commands**: The output of some system commands will be stored under the directory system_commands. Each command has its own ASCII text file that has the command that was executed, the output of the command and the result of executing the command.

- **root system commands**: Similar to system commands above, but these need root privileges.

So, for example, if the command `uname -a` is executed, then a file called `uname_a` will be created with the following content:

```
command:uname -a Linux localhost 3.13.0-24-generic #46-Ubuntu SMP
Thu Apr 10 19:08:14 UTC 2014 i686 i686 i686 GNU/Linux
return_value:0
```

Following are the system commands that are executed.

- `uname -a`
- `ifconfig`
- `hostname`
- `w`
- `date`
- `df`
- `df -i`
- `dmesg`
- `iostat -mx`
- `last`
- `mount`
- `netstat -an`
- `netstat -i`
- `netstat -rn`
- `ps aux`
- `sysctl -a`
- `lsb_release -a` (only on Linux)

Following are the root system commands that are executed.

- `iptables -L -t nat` (only on Linux)
- `iptables -L` (only on Linux)
WombatOAM reference

Logs

The logs are in rel/wombat/wombat/log. debug.log contains all log messages; wombat.log contains only error, warning and info messages. These log files can be backed up using the scripts/wombat-debug.sh script (see WombatOAM debug script for details).

WombatOAM data

All data (the list of nodes, node families, collected metrics, etc.) is stored in the rel/wombat/wombat/data/ directory. If you want a fresh start, delete this directory. The disk usage information of these data files can be collected using the scripts/wombat-debug.sh script (see WombatOAM debug script for details).

Managing older Erlang nodes

The WombatOAM agents module are compiled with an older Erlang/OTP version (R14B04) as WombatOAM itself. This way WombatOAM can manage nodes that run R14B04 or newer Erlang/OTP.

Multiple WombatOAM instances

You can run multiple WombatOAM instances on the same machine, but you need to modify the wombat.config files to make sure that different instances use different ports and different directories for storing data.

Gauges

If you would like to see graphical gauges showing the current value of a few numeric metrics, the metrics need to be specified in the wombat.config file. You can find a sample configuration in sys.config.