Exosense Server Operations Manual

Copyright 2012, 2013 Feuerlabs, Inc. All rights reserved.

2013-09-20

Contents

0.1 Exosense Server Operations Guide

The Exosense Server runs as a single Linux process (the Erlang VM).

0.1.1 Installing an Exosense Server

In the following, we will refer to the top directory where the Exosense server is installed as $EXODM_DIR (as in EXOsense Device Management, the name 'exodm' will appear in several places in the file structure, and is also the default node name for the Erlang VM instance).

Declare the environment variable `EXODM_DIR` in a suitable startup file, e.g. `.bashrc` or `.login`.

Start by unpacking the tar file containing the Exosense server system.

```
$ cd $EXODM_DIR
$ tar xf exosense_server-X.Y.Z.tgz
```

[exodm_file_structure_clean][]

[exodm_file_structure_clean]: exodm_file_structure_clean.png "File structure of a clean installation" width=412px height=461px

In the current example, the Exosense server version is 0.2.24. To make this the current version, we issue a control command:

```
$ rel/lib/exodm_0.2.24/ctl install
```

This will set up links and creates necessary files, among other things copying the `ctl` script to the `$EXODM_ROOT` directory. This script can be used to start, stop, and upgrade the node, as well as attach to a running node.

0.1.2 Setting site-specific parameters

After install, the $EXODM_DIR directory will contain the `ctl` script, as well as the files `vm.args` and `sys.config`. If you want to run a robust production server, you will want to uncomment the `-heart` entry in `vm.args`. This will enable a watchdog, which monitors the Erlang node and restarts it if it dies or stops responding. In order for it to know how to restart the node, you must also set the environment variable `HEART_COMMAND`.

You can, for example, put this in the `~/.bashrc` file:

```
export EXODM_DIR=${HOME}/exodm
export HEART_COMMAND="(cd ${EXODM_DIR} && ./ctl start)"
```

0.1.3 Starting the server

To start a node running in the background - the normal way of starting the node:

```
$ ./ctl start
```

0.1.4 Configuring the server

After initial start, the account `exodm` is automatically created. The default admin user for the account is `exodm-admin`, with a password that is the same as the node cookie (by default, `exodm`, unless you change it in `vm.args`). This account can only be accessed from the same host as the server. You can set up access credentials for the shell scripts in the `exosense_specs/scripts` (git://github.com/Feuerlabs/exosense_specs.git) directory, by creating a file, `~/.exodmrc`, with the following contents:

If you want to view the activity at startup, and work interactively in the Erlang shell:

```
$ ./ctl console
```

0.1.5 Attaching to the Exosense server console

```
$ ./ctl attach
$ ...
```

Attaching to the console means jumping into the Erlang shell. Using the shell, you can inspect the system, interactively call various functions and modify logging parameters.

0.1.6 Stopping the Exosense Server

```
$ ./ctl stop
```

If the server is up and running, the command will return 'ok'. If it isn't running, the command will report that it cannot reach the node.

0.1.7 Upgrading the Exosense Server

In the current version, the server is upgraded by stopping, possibly transforming the database, and restarting the node. Before stopping the node, you should unpack the latest release. The upgrade command will determine the currently installed version and start a node that performs any necessary conversions, and terminates when done.

```
$ ./ctl stop
$ rel/lib/exodm_0.2.25/ctl upgrade
$ ./ctl start
```

If the above went well, you should be able to attach to the console.

0.1.8 Backing up data hosted by the Exosense Server

The server uses append-only data structures on disk, so performing a normal backup on the `$EXODM_ROOT` directory will be suffifient to back up the system.

0.1.9 Restoring Exosense Server backup data

Use normal restore procedures for the `$EXODM_ROOT` directory and start the system normally.

0.1.10 Installing a Plugin

Plugins are loaded dynamically from the rel/plugins/ directory. Any plugins residing in that directory will be automatically loaded at startup, but can also be loaded/upgraded in service.

Plugins are expected to follow the Erlang/OTP packaging concept, e.g.:

```
rel/plugins/
......my_plugin-1.0/
...........ebin/
.................my_plugin.app
.................*.beam
```

As an example, here is a plugin that has been started automatically by the system

```
(exodm_n1@uwair.local)2> [A || A <- application:which_applications(),
                                element(1,A) == exodm_ck3].
[{exodm_ck3,"Exosense DM-specific functions for Getaround CK3",
  "1.3.2"}]
```

If we want to upgrade the plugin to, say, version 1.3.4, we unpack that version under rel/plugins/

```
uwair:exodm uwiger$ ls -1 rel/plugins
exodm_ck3-1.3.2
exodm_ck3-1.3.4
```

In the exodm Erlang shell, we can now reload (note: all plugins that need upgrading will be upgraded):

```
(exodm_n1@uwair.local)3> exodm:reload().
[exodm_ck3 vsn "1.3.4"] code:add_patha(/.../rel/plugins/exodm_ck3-1.3.4/ebin)
[exodm_ck3 vsn "1.3.4"] soft upgrade from "1.3.2"
{ok,[]}
```