Watchmaker Documentation

Release

Plus3 IT Systems

Aug 11, 2017

Contents

Applied Configuration Management

CHAPTER 1

Overview

Watchmaker is intended to help provision a system from its initial installation to its final configuration. It was inspired by a desire to eliminate static system images with embedded configuration settings (e.g. gold disks) and the pain associated with maintaining them.

Watchmaker works as a sort of task runner. It consists of "*managers*" and "*workers*". A *manager* implements common methods for multiple platforms (Linux, Windows, etc). A *worker* exposes functionality to a user that helps bootstrap and configure the system. *Managers* are primarily internal constructs; *workers* expose configuration artifacts to users. Watchmaker then uses a common *configuration file* to determine what *workers* to execute on each platform.

CHAPTER 2

Contents

2.1 Installation

2.1.1 From Python Package Index

The preferred method to install `watchmaker` is from the Python Package Index (PyPi), using `pip`. Without any other options, this will always install the most recent stable release.

```
pip install watchmaker
```

If you do not have Python or `pip`, this Python installation guide can guide you through the process.

2.1.2 From sources

Watchmaker can also be built and installed from source, using `git` and `pip`. The sources for `watchmaker` are available from the `GitHub repo`.

1. First clone the public repository to pull the code to your local machine:

   ```
   git clone https://github.com/plus3it/watchmaker.git --recursive && cd watchmaker
   ```

   This project uses submodules, so it's easiest to use the `--recursive` flag, as above. If you don't, you will need to pull in the submodules as well:

   ```
   git submodule update --init --recursive
   ```

2. If you want to install a specific branch or tag, check it out before installing watchmaker:

   ```
   git checkout <branch-tag-foo>
   ```

3. Then you can install Watchmaker:

```
pip install .
```

2.2 Configuration

Watchmaker is configured using a YAML file. Watchmaker's default config.yaml file should work out-of-the-box for most systems and environments. You can also use it as an example to create your own configuration file. The default config file will install Salt and use the bundled Salt formulas to harden the system according to the DISA STIG.

The configuration is a dictionary. The parent nodes (keys) are: `all`, `linux`, or `windows`. The parent nodes contain a list of workers to execute, and each worker contains parameters specific to that worker. The `all` node is applied to every system, and `linux` and `windows` are applied only to their respective systems.

You can create a file using the above format with your own set of standard values and use that file for Watchmaker. Pass the CLI parameter `--config` to point to that file.

2.2.1 Config.yaml Parent Nodes

all

Section for Worker configurations that affect the deployment of all platforms. The `all` section will override parameters that are set in the OS-specific sections of config.yaml.

linux

Section for Worker configurations that should only be applied to Linux-based systems.

windows

Section for Worker configurations that should only be applied to Windows-based systems.

2.2.2 Config.yaml Worker Nodes

Watchmaker includes the *workers* listed below. See the corresponding sections for details on their configuration parameters.

- *salt*
- *yum (Linux-only)*

salt

Parameters supported by the Salt Worker:

- `admin_groups` (*list*): The group(s) that you would like the admin accounts placed within.
- `admin_users` (*list*): The user(s) that would be created as admins.
- `computer_name` (*string*): The computer or hostname that should be applied.
- `environment` (*string*): Set for the environment in which the system is being built.

- `ou_path` (*string*): Specifies the full DN of the OU where the computer account will be created when joining a domain.

```
ou_path: "OU=Super Cool App,DC=example,DC=com"
```

- `salt_states` (*string, comma-separated*): User-defined salt states to apply.

```
salt_states: foo,bar
```

- `s3_source` (*boolean*): Use S3 utilities to retrieve content instead of http(s) utilities. For S3 utilities to work, the system must have boto credentials configured that allow access to the S3 bucket.
- `user_formulas` (*dict*): Map of formula names and URLs to zip archives of salt formulas. These formulas will be downloaded, extracted, and added to the salt file roots. The zip archive must contain a top-level directory that, itself, contains the actual salt formula. To "overwrite" bundled submodule formulas, make sure the formula name matches the submodule name.

```
user_formulas:
  foo-formula: https://path/to/foo.zip
```

- `salt_debug_log` (*string*): Path to the debug logfile that salt will write to.
- `salt_content` (*string*): URL to the Salt content file that contains further configuration specific to the salt install.
- `install_method` (*string*): (Linux-only) The method used to install Salt. Currently supports: `yum`, `git`
- `bootstrap_source` (*string*): (Linux-only) URL to the salt bootstrap script. This is required if `install_method` is set to `git`.
- `git_repo` (*string*): (Linux-only) URL to the salt git repo. This is required if `install_method` is set to `git`.
- `salt_version` (*string*): (Linux-only) A git reference present in `git_repo`, such as a commit or a tag. If not specifid, the HEAD of the default branch will be used. `installer_url` (*string*): (Windows-only) URL to the Salt Minion installer for Windows.

yum (linux-only)

Parameters supported by the Yum Worker:

- `repo_map` (list of maps): There be dragons here! Please be careful making changes to the default config. Thoroughly test your configuration. The default config specifies yum repos that contain the salt-minion. If the default repos are not included, and the salt-minion is not available, the Salt Worker will fail. You can add repos here that you would like enabled, but be wary of removing the default repos. Each map must contain the following keys:
  - `dist` (*list*): Distributions that would install this repo. Some repos are supported by multiple distros. (Currently supported distros are redhat, centos, and amazon.)
  - `el_version` (_string_): The Enterprise Linux version for this repo, as in el6 or el7. Expected values are `'6'` or `'7'`.
  - `url` (*string*): URL location of the repo file to be added to the system. This file will be copied to `/etc/yum.repos.d/`

  Example:

```
repo_map:
  - dist:
      - redhat
      - centos
    el_version: 6
    url: http://someplace.com/my.repo
```

2.2.3 Example config.yaml

```
all:
  - salt:
      admin_groups: None
      admin_users: None
      computer_name: None
      environment: False
      ou_path: None
      salt_content: https://s3.amazonaws.com/watchmaker/salt-content.zip
      salt_states: Highstate
      s3_source: False
      user_formulas:
        # To add extra formulas, specify them as a map of
        #    <formula_name>: <archive_url>
        # The <formula_name> is the name of the directory in the salt file_root
        # where the formula will be placed. The <archive_url> must be a zip
        # file, and the zip must contain a top-level directory that, itself,
        # contains the actual salt formula. To "overwrite" submodule formulas,
        # make sure <formula_name> matches submodule names. E.g.:
        #ash-linux-formula: https://s3.amazonaws.com/salt-formulas/ash-linux-formula-
↪master.zip
        #scap-formula: https://s3.amazonaws.com/salt-formulas/scap-formula-master.zip

linux:
  - yum:
      repo_map:
        #SaltEL6:
        - dist:
            - redhat
            - centos
          el_version: 6
          url: https://s3.amazonaws.com/systemprep-repo/linux/saltstack/salt/yum.
↪repos/salt-reposync-el6.repo
        - dist: amazon
          el_version: 6
          url: https://s3.amazonaws.com/systemprep-repo/linux/saltstack/salt/yum.
↪repos/salt-reposync-amzn.repo
        #SaltEL7:
        - dist:
            - redhat
            - centos
          el_version: 7
          url: https://s3.amazonaws.com/systemprep-repo/linux/saltstack/salt/yum.
↪repos/salt-reposync-el7.repo
  - salt:
      salt_debug_log: None
      install_method: yum
      bootstrap_source: None
```

```
      git_repo: None
      salt_version: None

windows:
  - salt:
      salt_debug_log: None
      installer_url: https://s3.amazonaws.com/systemprep-repo/windows/salt/Salt-
↪Minion-2016.11.6-AMD64-Setup.exe
```

2.3 Usage

2.3.1 `watchmaker` from the CLI

Once watchmaker is *installed* and a *configuration file* has been created (or you have decided to use the default configuration), using watchmaker as a CLI utility is as simple as executing `watchmaker`. Below is the output of `watchmaker --help`, showing the CLI options.

```
# watchmaker --help
Usage: watchmaker [OPTIONS]

  Entry point for Watchmaker cli.

Options:
  --version                       Show the version and exit.
  -c, --config TEXT               Path or URL to the config.yaml file.
  -l, --log-level [info|debug|critical|warning|error]
                                  Set the log level. Case-insensitive.
  -d, --log-dir DIRECTORY         Path to the directory where Watchmaker log
                                  files will be saved.
  -n, --no-reboot                 If this flag is not passed, Watchmaker will
                                  reboot the system upon success. This flag
                                  suppresses that behavior. Watchmaker
                                  suppresses the reboot automatically if it
                                  encounters a failure.
  -s, --salt-states TEXT          Comma-separated string of salt states to
                                  apply. A value of 'None' will not apply any
                                  salt states. A value of 'Highstate' will
                                  apply the salt highstate.
  --s3-source                     Use S3 utilities to retrieve content instead
                                  of http/s utilities. Boto3 must be
                                  installed, and boto3 credentials must be
                                  configured that allow access to the S3
                                  bucket.
  -A, --admin-groups TEXT         Set a salt grain that specifies the domain
                                  groups that should have root privileges on
                                  Linux or admin privileges on Windows. Value
                                  must be a colon-separated string. E.g.
                                  "group1:group2"
  -a, --admin-users TEXT          Set a salt grain that specifies the domain
                                  users that should have root privileges on
                                  Linux or admin privileges on Windows. Value
                                  must be a colon-separated string. E.g.
                                  "user1:user2"
  -t, --computer-name TEXT        Set a salt grain that specifies the
                                  computername to apply to the system.
```

```
  -e, --env TEXT                  Set a salt grain that specifies the
                                  environment in which the system is being
                                  built. E.g. dev, test, or prod
  -p, --ou-path TEXT              Set a salt grain that specifies the full DN
                                  of the OU where the computer account will be
                                  created when joining a domain. E.g.
                                  "OU=SuperCoolApp,DC=example,DC=com"
  --help                          Show this message and exit.
```

2.3.2 `watchmaker` as EC2 userdata

Calling watchmaker via EC2 userdata is a variation on using it as a CLI utility. The main difference is that you must account for installing watchmaker first, as part of the userdata. Since the userdata syntax and dependency installation differ a bit on Linux and Windows, we provide methods for each as examples.

Note: The `pip` commands in the examples are a bit more complex than necessarily needed, depending on your use case. In these examples, we are taking into account limitations in FIPS support in the default PyPi repo. This way the same `pip` command works for all platforms.

Linux

For **Linux**, you must ensure `pip` is installed, and then you can install `watchmaker` from PyPi. After that, run `watchmaker` using any option available on the *CLI*. Here is an example:

```
#!/bin/sh
PIP_URL=https://bootstrap.pypa.io/get-pip.py
PYPI_URL=https://pypi.org/simple

# Install pip
curl "$PIP_URL" | python - --index-url="$PYPI_URL"

# Install watchmaker
pip install --index-url="$PYPI_URL" --upgrade pip setuptools watchmaker

# Run watchmaker
watchmaker --log-level debug --log-dir=/var/log/watchmaker
```

Alternatively, cloud-config directives can also be used on **Linux**:

```
#cloud-config

runcmd:
  - |
    PIP_URL=https://bootstrap.pypa.io/get-pip.py
    PYPI_URL=https://pypi.org/simple

    # Install pip
    curl "$PIP_URL" | python - --index-url="$PYPI_URL"

    # Install watchmaker
    pip install --index-url="$PYPI_URL" --upgrade pip setuptools watchmaker
```

```
    # Run watchmaker
    watchmaker --log-level debug --log-dir=/var/log/watchmaker
```

Windows

For **Windows**, the first step is to install Python. `Watchmaker` provides a simple bootstrap script to do that for you. After installing Python, install `watchmaker` using `pip` and then run it.

```
<powershell>
$BootstrapUrl = "https://raw.githubusercontent.com/plus3it/watchmaker/master/docs/
↪files/bootstrap/watchmaker-bootstrap.ps1"
$PythonUrl = "https://www.python.org/ftp/python/3.6.2/python-3.6.2-amd64.exe"
$PypiUrl = "https://pypi.org/simple"

# Download bootstrap file
$BootstrapFile = "${Env:Temp}\$(${BootstrapUrl}.split('/')[-1])"
(New-Object System.Net.WebClient).DownloadFile("$BootstrapUrl", "$BootstrapFile")

# Install python
& "$BootstrapFile" -PythonUrl "$PythonUrl" -Verbose -ErrorAction Stop

# Install watchmaker
pip install --index-url="$PypiUrl" --upgrade pip setuptools watchmaker

# Run watchmaker
watchmaker --log-level debug --log-dir=C:\Watchmaker\Logs
</powershell>
```

2.3.3 `watchmaker` as a CloudFormation template

Watchmaker can be integrated into a CloudFormation template as well. This project provides a handful of CloudFormation templates that launch instances or create autoscaling groups, and that install and execute Watchmaker during the launch. These templates are intended as examples for you to modify and extend as you need.

Cloudformation templates

Cloudformation parameter-maps

Sometimes it is helpful to define the parameters for a template in a file, and pass those to CloudFormation along with the template. We call those "parameter maps", and provide one for each of the templates above.

2.3.4 `watchmaker` as a library

Watchmaker can also be used as a library, as part of another python application.

```
import watchmaker

arguments = watchmaker.Arguments()
arguments.config_path = None
arguments.no_reboot = False
arguments.salt_states = 'None'
arguments.s3_source = False
```

```
client = watchhmaker.Client(arguments)
client.install()
```

Note: This demonstrates only a few of the arguments that are available for the `watchmaker.Arguments()` object. For details on all arguments, see the *API Reference*.

2.4 Frequently Asked Questions

2.4.1 How do I know if watchmaker has installed?

To determine whether watchmaker is installed, the simplest method is to run the command `watchmaker --help`. If it displays the cli help page, watchmaker is installed. Another option is to check `pip list | grep watchmaker`.

2.4.2 What do I do if watchmaker failed to install?

First, review the *installation* document. Then double-check the output of a failed installation. Usually, the output points pretty clearly at the source of the problem. Watchmaker can be re-installed over itself with no problem, so once the root cause is resolved, simply re-install watchmaker.

2.4.3 How do I know if watchmaker has completed without errors?

By default, watchmaker will reboot the system after a sucessful execution. Therefore, if the system reboots, watchmaker executed successfully. If you are investigating sometime after watchmaker completed, check the logs for errors. If anything fails, watchmaker will suppress the reboot. (Though note that the `--no-reboot` flag can be used to suppress the reboot even after a successful execution.)

You can also test the watchmaker exit code programmatically. If watchmaker fails, it will return a non-zero exit code. If watchmaker completes successfully, it will return an exit code of zero. You would typically pass the `--no-reboot` flag if you intend to test the exit code and determine what to do from there.

2.4.4 What do I do if watchmaker failed to complete or completes with errors?

Start by checking the logs generated by watchmaker. The logs are stored in the directory specified by the `--log-dir` argument. Search the log for entries that have `[ERROR]`, this will give you a starting point to begin troubleshooting. Also, if a salt state failed, look for the pattern `Result: False`. If it is not an obvious or simple issue, feel free to create an issue on the watchmaker github page. If there is a salt_call.debug.log in the watchmaker log directory, you can look for `[ERROR]` messages in there as well. However, this log file can be very noisy and a message with the error label may not be related to the error you are encountering.

2.4.5 Does watchmaker support Enterprise Linux 7?

Watchmaker is supported on RedHat 7 and CentOS 7. However, there may be issues intermittently with the salt formulas as they are actively being developed.

2.5 API Reference

2.5.1 watchmaker

Watchmaker module.

class `watchmaker.`**`Arguments`**(*config_path=None*, *log_dir=None*, *no_reboot=False*, *log_level=None*, **args*, ***kwargs*)

Bases: `dict`

Create an arguments object for the *`watchmaker.Client`*.

Parameters

- **config_path** – (`str`) Path or URL to the Watchmaker configuration file. If `None`, the default config.yaml file is used. (*Default*: `None`)
- **log_dir** – (`str`) Path to a directory. If set, Watchmaker logs to a file named `watchmaker.log` in the specified directory. Both the directory and the file will be created if necessary. If the file already exists, Watchmaker appends to it rather than overwriting it. If this argument evaluates to `False`, then logging to a file is disabled. Watchmaker will always output to stdout/stderr. Additionaly, Watchmaker workers may use this directory to keep other log files. (*Default*: `None`)
- **no_reboot** – (`bool`) Switch to control whether to reboot the system upon a successful execution of *`watchmaker.Client.install()`*. When this parameter is set, Watchmaker will suppress the reboot. Watchmaker automatically suppresses the reboot if it encounters an error. (*Default*: `False`)
- **log_level** – (`str`) Level to log at. Case-insensitive. Valid options include, from least to most verbose:
  - `critical`
  - `error`
  - `warning`
  - `info`
  - `debug`

Important: For all **Keyword Arguments**, below, the default value of `None` means Watchmaker will get the value from the configuration file. Be aware that `None` and `'None'` are two different values, with different meanings and effects.

Keyword Arguments

- **`admin_groups`** – (`str`) Set a salt grain that specifies the domain _groups_ that should have root privileges on Linux or admin privileges on Windows. Value must be a colon-separated string. On Linux, use the `^` to denote spaces in the group name. (*Default*: `None`)

```
admin_groups = "group1:group2"

# (Linux only) The group names must be lowercased. Also, if
# there are spaces in a group name, replace the spaces with a
# '^'.
admin_groups = "space^out"
```

```
# (Windows only) No special capitalization nor syntax
# requirements.
admin_groups = "Space Out"
```

- **admin_users** – (str) Set a salt grain that specifies the domain _users_ that should have root privileges on Linux or admin privileges on Windows. Value must be a colon-separated string. (*Default*: None)

```
admin_users = "user1:user2"
```

- **computer_name** – (str) Set a salt grain that specifies the computername to apply to the system. (*Default*: None)
- **environment** – (str) Set a salt grain that specifies the environment in which the system is being built. For example: dev, test, or prod. (*Default*: None)
- **salt_states** – (str) Comma-separated string of salt states to apply. A value of 'None' (the string) will not apply any salt states. A value of 'Highstate' will apply the salt highstate. (*Default*: None)
- **s3_source** – (bool) Use S3 utilities to retrieve content instead of http/s utilities. For S3 utilities to work, boto3 must be installed, and the system must have boto credentials configured that allow access to the S3 bucket. (*Default*: None)
- **ou_path** – (str) Set a salt grain that specifies the full DN of the OU where the computer account will be created when joining a domain. (*Default*: None)

```
ou_path="OU=Super Cool App,DC=example,DC=com"
```

- **extra_arguments** – (list) A list of extra arguments to be merged into the worker configurations. The list must be formed as pairs of named arguments and values. Any leading hypens in the argument name are stripped. (*Default*: [])

```
extra_arguments=['--arg1', 'value1', '--arg2', 'value2']

# This list would be converted to the following dict and merged
# into the parameters passed to the worker configurations:
{'arg1': 'value1', 'arg2': 'value2'}
```

class watchmaker.**Client**(*arguments*)

Bases: object

Prepare a system for setup and installation.

Keyword Arguments **arguments** – (*Arguments*) A dictionary of arguments. See *watchmaker.Arguments*.

install()

Execute the watchmaker workers against the system.

Upon successful execution, the system will be properly provisioned, according to the defined configuration and workers.

watchmaker.managers

Watchmaker managers module.

watchmaker.managers.base

Watchmaker base manager.

class `watchmaker.managers.base.`**`ManagerBase`**(*system_params*, **args*, ***kwargs*)
Bases: `object`

Base class for operating system managers.

All child classes will have access to methods unless overridden by an identically-named method in the child class.

Parameters **system_params** – (`dict`) Attributes, mostly file-paths, specific to the system-type (Linux or Windows). The dict keys are as follows:

prepdir: Directory where Watchmaker will keep files on the system.

readyfile: Path to a file that will be created upon successful completion.

logdir: Directory to store log files.

workingdir: Directory to store temporary files. Deleted upon successful completion.

restart: Command to use to restart the system upon successful completion.

shutdown_path: (Windows-only) Path to the Windows `shutdown.exe` command.

`download_file`(*url*, *filename*, *sourceiss3bucket=False*)
Download a file from a web server or S3 bucket.

Parameters

- **url** – (`str`) URL to a file.
- **filename** – (`str`) Path where the file will be saved.
- **sourceiss3bucket** – (`bool`) Switch to indicate that the download should use boto3 to download the file from an S3 bucket. (*Default*: `False`)

`create_working_dir`(*basedir*, *prefix*)
Create a directory in `basedir` with a prefix of `prefix`.

Parameters

- **prefix** – (`str`) Prefix to prepend to the working directory.
- **basedir** – (`str`) The directory in which to create the working directory.

Returns Path to the working directory.

Return type `str`

`call_process`(*cmd*, *log_pipe=u'all'*, *raise_error=True*)
Execute a shell command.

Parameters

- **cmd** – (`list`) Command to execute.
- **log_pipe** – (`str`) Controls what to log from the command output. Supports three values: `stdout`, `stderr`, `all`. (*Default*: `all`)
- **raise_error** – (`bool`) Switch to control whether to raise if the command return code is non-zero. (*Default*: `True`)

Returns Dictionary containing three keys: `retcode` (`int`), `stdout` (`bytes`), and `stderr` (`bytes`).

Return type dict

cleanup()
Delete working directory.

extract_contents(*filepath*, *to_directory*, *create_dir=False*)
Extract a compressed archive to the specified directory.

Parameters

- **filepath** – (str) Path to the compressed file. Supported file extensions:
  - *.zip*
  - *.tar.gz*
  - *.tgz*
  - *.tar.bz2*
  - *.tbz*
- **to_directory** – (str) Path to the target directory
- **create_dir** – (bool) Switch to control the creation of a subdirectory within to_directory named for the filename of the compressed file. (*Default*: False)

class watchmaker.managers.base.**LinuxManager**(*system_params*, **args*, ***kwargs*)
Bases: *watchmaker.managers.base.ManagerBase*

Base class for Linux Managers.

Serves as a foundational class to keep OS consitency.

class watchmaker.managers.base.**WindowsManager**(*system_params*, **args*, ***kwargs*)
Bases: *watchmaker.managers.base.ManagerBase*

Base class for Windows Managers.

Serves as a foundational class to keep OS consitency.

class watchmaker.managers.base.**WorkersManagerBase**(*system_params*, *workers*, **args*, ***kwargs*)
Bases: object

Base class for worker managers.

Parameters

- **system_params** – (dict) Attributes, mostly file-paths, specific to the system-type (Linux or Windows).
- **workers** – (collections.OrderedDict) Workers to run and associated configuration data.

watchmaker.managers.workers

Watchmaker workers manager.

class watchmaker.managers.workers.**LinuxWorkersManager**(*system_params*, *workers*, **args*, ***kwargs*)
Bases: *watchmaker.managers.base.WorkersManagerBase*

Manage the worker cadence for Linux systems.

worker_cadence()
Manage worker cadence.

cleanup()
Execute cleanup function.

class watchmaker.managers.workers.**WindowsWorkersManager**(*system_params*, *workers*, **args*, ***kwargs*)
Bases: *watchmaker.managers.base.WorkersManagerBase*

Manage the worker cadence for Windows systems.

worker_cadence()
Manage worker cadence.

cleanup()
Execute cleanup function.

watchmaker.workers

Watchmaker workers module.

watchmaker.workers.salt

Watchmaker salt worker.

class watchmaker.workers.salt.**SaltBase**(**args*, ***kwargs*)
Bases: *watchmaker.managers.base.ManagerBase*

Cross-platform worker for running salt.

Parameters

- **salt_debug_log** – (`list`) Filesystem path to a file where the salt debug output should be saved. When unset, the salt debug log is saved to the Watchmaker log directory. (*Default*: `''`)
- **s3_source** – (`bool`) Use S3 utilities to download salt content and user formulas from an S3 bucket. If `True`, you must also install `boto3` and `botocore`. Those dependencies will not be installed by Watchmaker. (*Default*: `False`)
- **salt_content** – (`str`) URL to a salt content archive (zip file) that will be uncompressed in the salt "srv" directory. This typically is used to create a top.sls file and to populate salt's file_roots. (*Default*: `''`)
  - *Linux*: `/srv/salt`
  - *Windows*: `C:\Salt\srv`
- **salt_states** – (`str`) Comma-separated string of salt states to execute. Accepts two special keywords (case-insensitive). (*Default*: `''`)
  - `none`: Do not apply any salt states.
  - `highstate`: Apply the salt "highstate".
- **user_formulas** – (`dict`) Map of formula names and URLs to zip archives of salt formulas. These formulas will be downloaded, extracted, and added to the salt file roots. The zip archive must contain a top-level directory that, itself, contains the actual salt formula. To "overwrite" bundled submodule formulas, make sure the formula name matches the submodule name. (*Default*: `{}`)

- **admin_groups** – (str) Sets a salt grain that specifies the domain groups that should have root privileges on Linux or admin privileges on Windows. Value must be a colon-separated string. E.g. "group1:group2" (*Default*: '')
- **admin_users** – (str) Sets a salt grain that specifies the domain users that should have root privileges on Linux or admin privileges on Windows. Value must be a colon-separated string. E.g. "user1:user2" (*Default*: '')
- **environment** – (str) Sets a salt grain that specifies the environment in which the system is being built. E.g. dev, test, prod, etc. (*Default*: '')
- **ou_path** – (str) Sets a salt grain that specifies the full DN of the OU where the computer account will be created when joining a domain. E.g. "OU=SuperCoolApp, DC=example,DC=com" (*Default*: '')

run_salt(*command*, ***kwargs*)
Execute salt command.

Parameters **command** – (str or list) Salt options and a salt module to be executed by salt-call. Watchmaker will always begin the command with the options --local, --retcode-passthrough, and --no-color, so do not specify those options in the command.

service_status(*service*)
Get the service status using salt.

Parameters **service** – (obj:*str*) Name of the service to query.

Returns

('running', 'enabled') First element is the service running status. Second element is the service enabled status. Each element is a bool representing whether the service is running or enabled.

Return type tuple

service_stop(*service*)
Stop a service status using salt.

Parameters **service** – (str) Name of the service to stop.

Returns True if the service was stopped. False if the service could not be stopped.

Return type bool

service_start(*service*)
Start a service status using salt.

Parameters **service** – (str) Name of the service to start.

Returns True if the service was started. False if the service could not be started.

Return type bool

service_disable(*service*)
Disable a service using salt.

Parameters **service** – (str) Name of the service to disable.

Returns True if the service was disabled. False if the service could not be disabled.

Return type bool

service_enable(*service*)
Enable a service using salt.

Parameters **service** – (`str`) Name of the service to enable.

Returns `True` if the service was enabled. `False` if the service could not be enabled.

Return type `bool`

process_grains()
Set salt grains.

process_states(*states*)
Apply salt states.

Parameters **states** – (`str`) Comma-separated string of salt states to execute. Accepts two special keywords (case-insensitive):

- `none`: Do not apply any salt states.
- `highstate`: Apply the salt "highstate".

class watchmaker.workers.salt.**SaltLinux**(**args*, ***kwargs*)
Bases: *watchmaker.workers.salt.SaltBase*, *watchmaker.managers.base.LinuxManager*

Run salt on Linux.

Parameters

- **install_method** – (`str`) **Required**. Method to use to install salt. (*Default*: `yum`)
  - `yum`: Install salt from an RPM using yum.
  - `git`: Install salt from source, using the salt bootstrap.
- **bootstrap_source** – (`str`) URL to the salt bootstrap script. Required if `install_method` is `git`. (*Default*: `''`)
- **git_repo** – (`str`) URL to the salt git repo. Required if `install_method` is `git`. (*Default*: `''`)
- **salt_version** – (`str`) A git reference present in `git_repo`, such as a commit or a tag. If not specified, the HEAD of the default branch is used. (*Default*: `''`)

install()
Install salt and execute salt states.

class watchmaker.workers.salt.**SaltWindows**(**args*, ***kwargs*)
Bases: *watchmaker.workers.salt.SaltBase*, *watchmaker.managers.base.WindowsManager*

Run salt on Windows.

Parameters

- **installer_url** – (`str`) **Required**. URL to the salt installer for Windows. (*Default*: `''`)
- **ash_role** – (`str`) Sets a salt grain that specifies the role used by the ash-windows salt formula. E.g. `"MemberServer"`, `"DomainController"`, or `"Workstation"` (*Default*: `''`)

install()
Install salt and execute salt states.

watchmaker.workers.yum

Watchmaker yum worker.

class watchmaker.workers.yum.**Yum**(**args*, ***kwargs*)
Bases: *watchmaker.managers.base.LinuxManager*

Install yum repos.

Parameters **repo_map** – (list) List of dictionaries containing a map of yum repo files to systems. (*Default*: [])

get_dist_info()
Validate the Linux distro and return info about the distribution.

install()
Install yum repos defined in config file.

2.6 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

2.6.1 Bug Reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

2.6.2 Documentation Improvements

Watchmaker could always use more documentation, whether as part of the official Watchmaker docs, in docstrings, or even on the web in blog posts, articles, and such. The official documentation is maintained within this project in docstrings or in the docs directory. Contributions are welcome, and are made the same way as any other code. See *Development*.

2.6.3 Feature Requests and Feedback

The best way to send feedback is to file an issue on GitHub.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a community-driven, open-source project, and that code contributions are welcome. :)

2.6.4 Development

To set up `watchmaker` for local development:

1. Fork watchmaker (look for the "Fork" button).
2. Clone your fork locally and update the submodules:

```
git clone https://github.com/your_name_here/watchmaker.git && cd watchmaker
git submodule update --init --recursive
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

4. Now you can make your changes locally.
5. When you're done making changes, use tox to run the linter, the tests, and the doc builder:

```
tox
```

> **NOTE**: This will test the package in all versions of Python listed in `tox.ini`. If `tox` cannot find the interpreter for the version, the test will fail with an `InterpreterNotFound` error. This is ok, as long as at least one interpreter runs and the tests pass. You can also specify which *tox environments* to execute, which can be used to restrict the Python version required.
>
> You can also rely on Travis and Appveyor to run the tests after opening the pull request. They will be slower though...

6. In addition to building the package and running the tests, `tox` will build any docs associated with the change. They will be located in the `dist/docs` directory. Navigate to the folder, open index.html in your browser, and verify that the doc text and formatting are as you intended.

   If you *only* want to build the docs, run:

```
tox -e docs
```

7. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

2.6.5 Pull Request Guidelines

If you need some code review or feedback while you are developing the code just open the pull request.

For pull request acceptance, you should:

1. Include passing tests (Ensure `tox` is successful).
2. Update documentation whenever making changes to the API or functionality.
3. Add a note to `CHANGELOG.md` about the changes. Include a link to the pull request.

2.6.6 Tox Tips

1. The *primary* tox environments for `watchmaker` include:
   - `check`
   - `docs`
   - `py26`
   - `py27`
   - `py35`
   - `py36`
2. To run a subset of environments:

```
tox -e <env1>,<env2>,<env3>,etc
```

3. To run a subset of tests:

```
tox -e <environment> -- py.test -k <test_myfeature>
```

4. To run all the test environments in *parallel*, use `detox`:

```
pip install detox
detox
```

2.6.7 Build a Development Branch in EC2

To install and run a development branch of watchmaker on a new EC2 instance, specify something like this for EC2 userdata:

- **For Linux**: Modify `GIT_REPO` and `GIT_BRANCH` to reflect working values for your development build. Modify `PIP_URL` and `PYPI_URL` as needed.

```
#!/bin/sh
GIT_REPO=https://github.com/<your-github-username>/watchmaker.git
GIT_BRANCH=<your-branch>

PIP_URL=https://bootstrap.pypa.io/get-pip.py
PYPI_URL=https://pypi.org/simple

# Install pip
curl "$PIP_URL" | python - --index-url="$PYPI_URL"

# Install git
yum -y install git

# Upgrade pip and setuptools
pip install --index-url="$PYPI_URL" --upgrade pip setuptools

# Clone watchmaker
git clone "$GIT_REPO" --branch "$GIT_BRANCH" --recursive

# Install watchmaker
cd watchmaker
pip install --index-url "$PYPI_URL" --editable .
```

```
# Run watchmaker
watchmaker --log-level debug --log-dir=/var/log/watchmaker
```

- **For Windows**: Modify `GitRepo` and `GitBranch` to reflect working values for your development build. Optionally modify `BootstrapUrl`, `PythonUrl`, `GitUrl`, and `PypiUrl` as needed.

```
<powershell>
$GitRepo = "https://github.com/<your-github-username>/watchmaker.git"
$GitBranch = "<your-branch>"

$BootstrapUrl = "https://raw.githubusercontent.com/plus3it/watchmaker/master/docs/
↪files/bootstrap/watchmaker-bootstrap.ps1"
$PythonUrl = "https://www.python.org/ftp/python/3.6.2/python-3.6.2-amd64.exe"
$GitUrl = "https://github.com/git-for-windows/git/releases/download/v2.13.3.
↪windows.1/Git-2.13.3-64-bit.exe"
$PypiUrl = "https://pypi.org/simple"

# Download bootstrap file
$BootstrapFile = "${Env:Temp}\$(${BootstrapUrl}.split("/")[-1])"
(New-Object System.Net.WebClient).DownloadFile($BootstrapUrl, $BootstrapFile)

# Install python and git
& "$BootstrapFile" `
    -PythonUrl "$PythonUrl" `
    -GitUrl "$GitUrl" `
    -Verbose -ErrorAction Stop

# Upgrade pip and setuptools
pip install --index-url="$PypiUrl" --upgrade pip setuptools

# Clone watchmaker
git clone "$GitRepo" --branch "$GitBranch" --recursive

# Install watchmaker
cd watchmaker
pip install --index-url "$PypiUrl" --editable .

# Run watchmaker
watchmaker --log-level debug --log-dir=C:\Watchmaker\Logs
</powershell>
```

CHAPTER 3

Supported Operating Systems

- Enterprise Linux 7 (RHEL/CentOS/etc)
- Enterprise Linux 6 (RHEL/CentOS/etc)
- Windows Server 2016
- Windows Server 2012 R2
- Windows Server 2008 R2
- Windows 10
- Windows 8.1

CHAPTER 4

Supported Python Versions

- Python 3.3 and later
- Python 2.6 and later

Python Module Index

W

Index

Y