Confab Documentation

Release 1.1.1

locationlabs and contributors

2013-05-21

CONTENTS

Configuration Management with Fabric and Jinja2.

CHAPTER

ONE

CONTENTS

1.1 Quickstart

1. Install confab:

   ```
   pip install confab
   ```

2. Create a `settings.py` file:

   ```
   cat > settings.py << "EOF"
   environmentdefs = {
       'local': ['localhost']
   }

   roledefs = {
       'example': ['localhost']
   }
   EOF
   ```

3. Create a template:

   ```
   mkdir -p templates/example/tmp/
   echo '{{ value }}' > templates/example/tmp/hello.txt
   ```

4. Create data to populate the template:

   ```
   mkdir -p data
   echo 'value = "world"' > data/default.py
   ```

5. Review the difference between the template value and the value on the target host:

   ```
   confab diff
   ```

6. Push changes to the target host:

   ```
   confab push
   ```

7. Review the change:

   ```
   ssh localhost cat /tmp/hello.txt
   ```

1.2 Functions

Confab provides four basic functions:

1. It defines a data model where *hosts* belong to one or more *environments* and are assigned to one or more *roles*, which can be made up of *components*.
2. It defines a mechanism for loading configuration data based on a set of *defaults* and override values defined per *environment*, *host*, *role*, or *component*.
3. It defines a mechanism for loading Jinja2 templates for configuration files based on a *role* and/or *component*.
4. It defines Fabric tasks to faciliate publishing configuration files generated from applying the configuration data to the Jinja2 templates to *hosts*.

Confab's configuration data and *Templates* are expected to be checked in to version control so that changes to configuration are managed through a regular versioning and release process.

1.3 Tasks

Confab provides four default tasks:

`generate` Generate configuration files from templates.

`pull` Pull copies of configuration files from a remote host.

`diff` Show differences between generated and remote configuration files.

`push` Interactively push generated configuration files to a remote host.

The default tasks all expect a series of *Directories* as inputs.

1.4 Usage

Confab may be used in several ways.

1.4.1 Via `confab` – The Default Console Script

The distribution ships with the `confab` console script, which provides a simple command line usage based on common defaults:

```
confab -d /path/to/directory -H hosts -u user <command>
```

1.4.2 Via Inclusion in a `fabfile`

Confab's tasks may be included in another fabfile simply by adding:

```
from confab.api import diff, generate, generate_tasks, pull, push

# generate environment tasks
generate_tasks('/path/to/settings')
```

And then running:

```
fab <env_name>:{role1},{role2} <task>:<arguments>
```

Note that the settings path, roles list, and arguments are optional.

1.4.3 Via the API

Confab's lower level *API* can be invoked directly either to create new tasks or as part of some other script:

```
from confab.api import *
from confab.autotasks import generate_tasks

# load roledefs and environmentdefs from settings.py
load_model_from_dir('/path/to/directory')
# create tasks for each defined role and environment
generate_tasks()
```

Autotasks would then allow fab to run as:

```
fab role_{role} env_{environment} <task>:arguments
```

Confab's lower level API can also be invoked using customized data loading functions, either to create new tasks or to be called directly from a new console script.

1.5 Directories

Confab loads all of its definitions, generates *Templates*, and saves remote copies of configuration files in locations relative to a single base directory.

A normal Confab directory tree might look like:

```
base_dir/settings.py             # definitions
base_dir/templates/{component}/  # templates for {component}
base_dir/data/default.py         # default configuration data
base_dir/data/{environment}.py   # per-environment configuration data
base_dir/data/{role}.py          # per-role configuration data
base_dir/data/{component}.py     # per-component configuration data
base_dir/data/{host}.py          # per-host configuration data
base_dir/generated/{hostname}/   # generated configuration files for hostname
base_dir/remotes/{hostname}/     # copies of remote configuration files from hostname
```

Confab selects this base directory in one of several ways:

1. By default, the base directory is the same directory where `settings.py` was loaded.

   Both the `confab` console script and the `generate_tasks()` function load `settings.py` and construct an `EnvironmentDefinition`, which retains a reference to the directory where `settings.py` was loaded. This definition is saved in the Fabric environment as `env.environmentdef` for use by subsequent tasks.

2. The `diff()`, `generate()`, `pull()`, and `push()` tasks support an explicit directory argument.

3. If all else fails, Confab falls back to `os.getcwd()`.

1.6 Templates

Confab uses Jinja2's environment to enumerate configuration file templates. Any valid `Jinja2 environment` may be provided as long as it uses a Loader that supports `jinja2.Environment.list_templates()`. By default, Confab uses a `jinja2.FileSystemLoader`.

Templates are loaded from a directory tree based on the selected component(s). For example, given the following directory structure:

```
/path/to/base_dir/templates/foo/etc/motd.tail
/path/to/base_dir/templates/foo/etc/hostname
/path/to/base_dir/templates/bar/etc/cron.d/baz
```

If the `foo` component is selected, `/etc/motd.tail` and `/etc/hostname` will be loaded; if the `bar` component is selected, only `/etc/cron.d/baz` will be loaded. Note that configuration file names and paths may also be templates.

1.7 Data Loading

Configuration data is loaded from python modules named after the selected *environment*, *role*, *component*, and *host*, plus a standard set of defaults. For example, if Confab is operating on an environment named `foo`, a role named `bar`, a component named `baz`, and a host named `host`, configuration data would be loaded from `foo.py`, `bar.py`, `baz.py`, `host.py`, and `default.py`.

If a configuration data module is not found, Confab will also look for a file with a `.py_tmpl` suffix and treat it as a Jinja2 template for the same module, allowing configuration data to use Jinja2 template syntax (including `include`).

Confab uses the `__dict__` property of the loaded module to generate dictionaries, filtering out any entries starting with _. In other words, this module:

```
foo = 'bar'
_ignore = 'this'
```

results in this dictionary:

```
{'foo': 'bar'}
```

The dictionaries from all of the loaded modules (if any) are recursively merged into a single dictionary, which is then used to populate *Templates*. Merging operates in the following order:

1. Host-specific values are used first.
2. Environment-specific values are used next.
3. Role or component-specific values are used next.
4. Default values are used last.

Confab's recursive merge operation can be futher customized by using callable wrappers around configuration values. Confab will always delegate to a callable to define how values are overriden, e.g. allowing lists values to be appended/prepended to default values.

1.8 Glossary

component Components are slices of configuration files.

The configuration files that Confab manages are controlled by which components are selected.

confab confab (in all lowercase) is an included command line script.

For details see *Via confab – The Default Console Script*.

environment Environments are groups of *hosts* that work together for a single purpose.

It's common to have one environment for development, one for staging, one for production and so forth.

host Hosts are physical or virtual machines accessible via ssh.

Confab will normally identify hosts by their fully qualified domain name (FQDN), so hostnames matter.

role Roles are groups of zero or more *components* that achieve a common purpose.

In the degenerate case where a role has no components, the role itself is taken to be a component.

1.9 API Reference

1.9.1 confab.api

Non-init module for doing convenient * imports from.

Core

- ConfFiles

Settings

- Settings

Environment Tasks

- generate_tasks()

Jinja2 Environment Loading

- FileSystemEnvironmentLoader
- PackageEnvironmentLoader

Data Loading

- DataLoader

Options

- Options
- assume_yes()

Iterations

- iter_hosts_and_roles()
- iter_conffiles()
- make_conffiles()

Fabric Tasks

- diff()
- generate()
- pull()
- push()

1.9.2 confab.autotasks

Auto-generate Fabric tasks for roles and environments.

The autogenerate_tasks() function creates fabric tasks to set the current *role* or :term‘environment‘ and is intended to be used along side the other standard Confab tasks (e.g. pull()) to customize configuration data.

These tasks are somewhat experimental.

confab.autotasks.**autogenerate_tasks**()
Autogenerate role_ and env_ tasks for all defined roles and environments in the Fabric environment.

Normally the roledefs and environment defs will be configured using confab.model.load_model_from_dir() or similar.

1.9.3 confab.conffiles

Configuration file template object model.

class confab.conffiles.**ConfFile**(*template*, *data*)
Bases: object

Encapsulation of a configuration file template.

diff(*generated_dir*, *remotes_dir*, *output=False*)
Compute the diff between the generated and remote files.

If output is enabled, show the diffs nicely.

generate(*generated_dir*)
Write the configuration file to the dest_dir.

pull(*remotes_dir*)
Pull remote configuration file to local file.

push(*generated_dir*)
Push the generated configuration file to the remote host.

class confab.conffiles.**ConfFileDiff**(*remote_file_name*, *generated_file_name*, *conffile_name*)
Bases: object

Encapsulation of the differences between the (locally copied) remote and generated versions of a configuration file.

show()
Print the diff using pretty colors.

If confab is used on binary files, diffs are likely to render poorly.

class confab.conffiles.**ConfFiles**(*environment_loader*, *data_loader*)
Bases: object

Encapsulation of a set of configuration files.

diff(*generated_dir*, *remotes_dir*)
Show diffs for all configuration files.

generate(*generated_dir*)
Write all configuration files to generated_dir.

pull(*remotes_dir*)
Pull remote versions of files into remotes_dir.

push(*generated_dir*, *remotes_dir*)
Push configuration files that have changes, given user confirmation.

1.9.4 confab.data

Functions for loading configuration data.

class confab.data.**DataLoader**(*data_dir*)
Bases: object

Load and merge configuration data.

Configuration data is loaded from python files by type, where type is defined to include defaults, per-role values, per-component values, per-environment values and per-host values.

Configuration data also includes the current environment and host string values under a confab key.

confab.data.**import_configuration**(*module_name*, *data_dir*)
Load configuration from file as python module.

Returns publicly names values in module's __dict__.

1.9.5 confab.diff

Determine the difference between remote and generated configuration files.

confab.diff.**diff**
Show configuration file diffs.

1.9.6 confab.generate

Generate configuration files into generated_dir.

confab.generate.**generate**
Generate configuration files.

1.9.7 confab.loaders

Jinja2 Environment loading helper functions.

Confab uses the environments jinja2.Environment.list_templates() method to abstract template location from rendering and synchronization.

Note that the default Jinja2 Loaders assume a charset (default: utf-8).

class confab.loaders.**ConfabFileSystemLoader**(*searchpath*, *encoding='utf-8'*)
Bases: jinja2.loaders.FileSystemLoader

Adds support for binary templates when loading an environment from the file system.

Binary config files cannot be loaded as Jinja2 templates by default, but since confab's model is built around Jinja2 environments we need to make sure we can still represent them as Jinja2 Templates.

Since confab only renders templates from text config files (see confab.conffiles.Conffile.generate() and confab.options.should_render()) we can workaround this by returning a dummy template for binary config files with the appropriate metadata. When generating the configuration, confab, instead of rendering the template, will just copy the template file (the binary config file) verbatim to the generated folder.

class confab.loaders.**EmptyLoader**
Bases: jinja2.loaders.BaseLoader

Jinja template loader with no templates.

class confab.loaders.**FileSystemEnvironmentLoader**(*dir_name*)
Bases: object

Loads Jinja2 environments from directories.

class confab.loaders.**PackageEnvironmentLoader**(*package_name*, *templates_path='templates'*)
Bases: object

Loads Jinja2 environments from python packages.

1.9.8 confab.main

Main function declaration for confab console_script.

Confab may be used from within a fabfile or as a library. The main function here is provided as a simple default way to invoke confab's tasks:

- A single directory root is assumed, with templates, data, generated and remotes directories defined as subdirectories.
- A host list must be provided on the command line.

For more complex invocation, a custom fabfile may be more appropriate.

confab.main.**main**()
Main command line entry point.

confab.main.**parse_options**()
Parse command line options.

Directory and host are required, though directory defaults to the current working directory.

1.9.9 confab.merge

Configuration data structure merging functions.

We expect to obtain hierarchical configurations as native dictionaries and want to merge the higher precedence (override) data into the lower precedence (default) data, potentially multiple times.

Merge rules are as follows:

- Dictionaries are merged recursively by default.

- Lists and primitives are replaced by default.
- Callables are called to provide custom extension.

For example, a list of hosts in the default dictionary will normally be replaced by values defined in the override dictionary; however if the override dictionary's list is a callable, it can be made to do something else, such as append a new host to the default list.

class confab.merge.**Append**(**args*)
Bases: list

Customized callable list that appends its values to the default.

class confab.merge.**Prepend**(**args*)
Bases: list

Customized callable list that prepends its values to the default.

class confab.merge.**UniqueUnion**(**args*)
Bases: list

Customized callable list that adds its values to the default list preserving unique values.

confab.merge.**merge**(**args*)
Recursively merge multiple dictionaries.

1.9.10 confab.model

Functions for interacting with the defined *hosts*, *environments*, and *roles*.

confab.model.**get_components_for_role**(*role*)
Get all component paths for the given role.

confab.model.**get_hosts_for_environment**(*environment*)
Get all hosts for an environment.

Assumes an environmentsdef structure in Fabric's env.

confab.model.**get_roles_for_host**(*host*)
Get all roles that a host belongs to.

Delegates to Fabric's env roledefs.

confab.model.**load_model_from_dict**(*settings*)
Load model data (environments, roles, hosts) from settings dictionary.

confab.model.**load_model_from_dir**(*dir_name*, *module_name='settings'*)
Load model data (environments, roles, hosts) from settings module.

1.9.11 confab.options

Options for managing Confab.

class confab.options.**Options**(***kwargs*)
Bases: object

Context manager to temporarily set options.

confab.options.**assume_yes**
Set the option to assume_yes in other tasks.

1.9.12 confab.output

Output control utilities.

confab.output.**configure_output**(*verbosity=0*, *quiet=False*)
Configure verbosity level through Fabric's output managers.

confab.output.**debug**(*message*, ***kwargs*)
Generate fabric-style output if and only if debug output has been selected.

confab.output.**status**(*message*, ***kwargs*)
Generate fabric-style output if and only if status output has been selected.

confab.output.**warn_via_fabric**(*message*, *category*, *filename*, *lineno=None*, *line=None*)
Adapt Python warnings to Fabric's warning output manager.

1.9.13 confab.pull

Pull configuration files from remote host into remotes_dir.

confab.pull.**pull**
Pull remote configuration files.

1.9.14 confab.push

Push generated configuration files to remote host.

confab.push.**push**
Push configuration files.

1.9.15 confab.resolve

Resolve environment, role, and host choices into actions.

If a user specifies only an *environment*, confab should target all hosts and roles in that environment. If one or more roles – or one or more hosts – are specified explicilty, confab should target a subset.

confab.resolve.**resolve_hosts_and_roles**(*environment*, *hosts=None*, *roles=None*)
Given an environment, (possibly empty) list of hosts, and a (possibly empty) list of roles, return a mapping from host to roles to target.

Raises an exception if any targeted host would have no roles after resolution.

1.9.16 confab.validate

Functions for validating user input to tasks.

confab.validate.**validate_all**(*templates_dir*, *data_dir*, *generated_dir*, *remotes_dir*)
Validate templates_dir, data_dir, generated_dir, remotes_dir, and host.

confab.validate.**validate_data_dir**(*data_dir*)
Data directory must be defined and exist.

confab.validate.**validate_generate**(*templates_dir*, *data_dir*, *generated_dir*)
Validate templates_dir, data_dir, generated_dir, and host.

confab.validate.**validate_generated_dir**(*generated_dir*)
Generated directory must be defined and not be a regular file.

confab.validate.**validate_host**()
Fabric host_string must be defined.

confab.validate.**validate_pull**(*templates_dir*, *data_dir*, *remotes_dir*)
Validate templates_dir, data_dir, remotes_dir, and host.

confab.validate.**validate_remotes_dir**(*remotes_dir*)
Remotes directory must be defined and not be a regular file.

confab.validate.**validate_templates_dir**(*templates_dir*)
Template directory must be defined and exist.

1.10 Future Work

1. Confab needs a better **push** command line interface, and the following is a possible option:

```
The following configuration files have changed for localhost:

   no | filename                                                | changed
   ---+----------------------------------------------------------+--------
   1  | /etc/iptables.rules                                     | new
   2  | /etc/iptables.rules.services                            | yes
   3  | /opt/wm/etc/sprint_sms_gateway/gateway.properties       | no

Select files to push? [all/None/..1,2..] 1,3
```

2. Similarly **diff** should offer a similar option to select files to show diffs:

```
The following configuration files have changed for localhost:

   no | filename                                                | changed
   ---+----------------------------------------------------------+--------
   1  | /etc/iptables.rules                                     | new
   2  | /etc/iptables.rules.services                            | yes
   3  | /opt/wm/etc/sprint_sms_gateway/gateway.properties       | no

See changes for file(s)? [all/..1,2..] 1,3
```

1.11 Change History

1.11.1 1.3 - unreleased

- Add Sphinx documentation with API docs. Publish on RTD.
- Convert `CHANGES.md` to RST to make them nicely embeddable in the docs.
- Replace the old `README.md` with a new `README.rst` that points to the documentation on RTD.
- Add a `long_description` to `setup.py` to provide more info on PyPI.

1.11.2 1.2 - 2013-05-02

- Major refactor of underlying data model:
  - Autogenerated `env` tasks to use with `fab`
  - `definitions.py` replaces `model.py` and `resolve.py`
  - Provides a simpler object and iteration model for navigation confab settings.
  - Moves host and role iteration out of `main.py` into `iterconffiles` function.
- No prevents multiple roles/components from defining the same template on the same host.
- Allows multiple loading of data from identically named files in different directories.
- Ensures that role data may override component data.
- Supports customized `module_as_dict` conversion.

1.11.3 1.1 - 2013-03-25

- Allows roles to have components: components may have templates and may be reused across different roles, with customization defined in role configuration data.
- Abstracts template and data loading into callable objects to allow ConfFiles to load templates and data for specific components.
- Allows data modules to be templates (`*.py_tmpl`) as well as Python files (`*.py`).
- Adds output verbosity control and more verbose output.
- Instead of enforcing that all hosts have the same role when invoking the `confab` CLI, "does the right thing."
- Changes uses of run() to sudo() and always uses `use_sudo` when calling put().
- Switches merge resolution order to respect environment configuration before role or component configuration.

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

C