refit Documentation

Release 0.3.0

Daniel Townsend

Apr 23, 2020

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Welcome to Refit - simple remote server configuration, using asyncio.

1.1 Installation

Make sure Python 3.7 or above is installed, then install the following (preferably in a virtualenv):

```
pip install refit
```

1.2 Creating a project

For the purposes of the documentation, we'll assume we're creating a pet shop web app.

```
refit scaffold pet_shop
```

This will create a `deployments` folder in the current directory, with a `pet_shop` folder inside, containing a `hosts.py` and `tasks.py` file.

You can create as many different deployments as you like - each one represents a collection of tasks which achieve some objective. For example deploying a web app, or configuring a database server.

1.3 Hosts

In the hosts file, we define the remote machines we want to connect to.

It's important that each `Host` gets registered with the `HostRegistry`. The recommended way of doing this is using the decorator syntax.

```
# Generated by Refit version: 0.1.0

from refit.host import Host
from refit.registry import HostRegistry


host_registry = HostRegistry()


@host_registry.register(environment="production")
```

(continues on next page)

(continued from previous page)

```
class PetShopProduction(Host):
    address = "127.0.0.1"
    environment_vars = {}


@host_registry.register(environment="test")
class PetShopTest(Host):
    address = "127.0.0.1"
    environment_vars = {}
```

Hint: Refit uses SSH to communicate with remote servers. In order to access your remote servers, make sure your SSH public key is present in the `known_hosts` file on each remote server. For example, `/home/my_user/.ssh/known_hosts`.

1.4 Tasks

Tasks are what get run on hosts. Examples are uploading files, or running a bash command.

Similarly to `Host`, it's important that each `Task` gets registered with the `TaskRegistry`, otherwise it won't get run.

```
# Generated by Refit version: 0.1.0

from refit.task import Task
from refit.registry import TaskRegistry


task_registry = TaskRegistry()


@task_registry.register
class PetShop(Task):
    """
    Provision PetShop.
    """

    async def run(self):
        path = "/tmp/hello_world"

        if not await self.path_exists(path):
            await self.create_folder(path)

        await self.raw('touch /tmp/hello_world/greetings.txt')
```

The order in which a `Task` is added to the registry determines the order in which it runs.

1.4.1 run

Each Task has an `async run` method, which performs the actual work.

You can do whatever you like within this method, but a lot of the time you'll be calling other Task methods, which implement the bulk of Refit's functionality. Under the hood, these are implemented as *Mixins*.

1.5 Running tasks

Once you have defined your hosts and tasks, you run them with the following command:

```
refit deploy --environment=test pet_shop
```

You'll notice in your hosts file that there's multiple `Host` subclasses, one for each environment. You need to specify which environment you want to deploy to when running your tasks.

CHAPTER

TWO

MIXINS

The `Task` class inherits from many mixins, which provide a lot of useful utilities for performing common server admin tasks.

2.1 AptMixin

class `refit.mixins.apt.`**AptMixin**

async apt_autoremove()

Return type `None`

async apt_install(**packages*)

Return type `None`

async apt_update()

Return type `None`

2.2 DockerMixin

class `refit.mixins.docker.`**DockerMixin**

async create_docker_network(*network_name*)

Return type `None`

async docker_compose_up(*compose_file_path*)

Parameters **compose_file_path** (`str`) – Path to the compose file on the remote machine.

Return type `None`

async docker_prune()

Remove any unused images, networks, and containers.

Return type `None`

async get_docker_network_names()

Return type `List[str]`

async pull_docker_image(*image_name*)

Return type `None`

2.3 FileMixin

class `refit.mixins.file.`**FileMixin**

async create_file(*path*)
Create an empty file on the remote server.

Return type `None`

async create_folder(*path*, *owner='root'*, *group='root'*, *permissions='755'*)
Creates folder, and all intermediate directories.

Only changes the group and owner of the deepest directory. If each folder in the chain needs certain permissions, call this function repeatedly for each folder.

Return type `None`

async path_exists(*path*)
Checks whether the path exists on the remote machine.

Return type `bool`

async upload_file(*local_path*, *remote_path*, *root=''*)
Upload a file using scp to the remote machine.

Return type `None`

2.4 PathMixin

class `refit.mixins.path.`**PathMixin**
Utilities for inspecting the path on the remote machine.

async in_path(*executable*, *raise_exception=False*)
Check whether an executable is available on the path.

Return type `bool`

2.5 PythonMixin

class `refit.mixins.python.`**PythonMixin**

async pip(*package*)
Install a Python package using pip.

Return type `None`

2.6 SystemdMixin

2.7 TemplateMixin

class refit.mixins.template.**TemplateMixin**

> **async upload_template**(*local_path*, *remote_path*, *context*, *root=''*)
> Render a jinja template using the provided context, and upload it to the remote server using scp.

2.8 Custom Mixins

There's nothing magical about the builtin mixins - you can develop your own, and inherit from them.

```
from refit.task import Task


class MyMixin():
    def hello_world(self):
        print('hello world')


class MyTask(Task, MyMixin):
    async def run(self):
        self.hello_world()
```

CHAPTER

THREE

TASK SEQUENCING

You can use Refit to execute commands sequentially on a single server. However, much of it's power is in running several commands at the same time - either on a single machine, or across multiple machines.

3.1 TaskRegistry

Instead of adding your `Task` to the `TaskRegistry` using the decorator syntax, you can also use `register` or `gather`.

3.1.1 register

This tells the task registry to execute the given tasks sequentially.

```
from ..shared.tasks import AddKeysTask, CreateDatabaseTask

task_registry = TaskRegistry()
task_registry.register(AddKeysTask, CreateDatabaseTask)
```

3.1.2 gather

This tells the task registry to execute the given tasks concurrently.

```
from ..shared.tasks import AddKeysTask, CreateDatabaseTask

task_registry = TaskRegistry()
task_registry.gather(AddKeysTask, CreateDatabaseTask)
```

CHAPTER FOUR

TAGS

Tags are how you associate Tasks with certain Hosts.

If you don't specify any tags when you register your Tasks, the Task will be run for each Host in the current environment.

If you specify any tags, then the Task will only get run on Hosts with a matching tag. An example tag is `'database'`, for a database server.

```
task_registry = TaskRegistry()
task_registry.register(TaskOne, tags=['database'])

@task_registry.register(tags=['load_balancer'])
class TaskTwo(Task):
    async def run(self):
        print("Running on load_balancer servers")
```

CHAPTER

FIVE

CONTRIBUTING

5.1 Mixins

The easiest way to contribute is to add new Mixins, or extend existing ones, with common functionality.

5.2 Style Guide

The Black formatter is used, with a line length of 79 to make it consistent with PEP8.

5.3 Tests

Some tests require Docker to be running, so Refit can actually SSH onto a server to execute commands.

CHAPTER SIX

MODULES

6.1 task

class refit.task.**Concurrent**(*host_class*)

Bases: *refit.task.Task*

Bundles several tasks to be run concurrently.

async run()

Override in subclasses. This is what does the actual work in the task, and is awaited when the Task is run.

class refit.task.**Task**(*host_class*)

Bases: *refit.mixins.apt.AptMixin*, *refit.mixins.docker.DockerMixin*, *refit.mixins.file.FileMixin*, *refit.mixins.path.PathMixin*, *refit.mixins.python.PythonMixin*, *refit.mixins.template.TemplateMixin*

classmethod create(*host_registry*, *environment*)

Creates and runs a task for all matching hosts.

Return type None

async entrypoint()

Kicks off the task, along with printing some info.

Return type None

async raw(*command*, *raise_exception=True*)

Execute a raw shell command on the remote server.

abstract async run()

Override in subclasses. This is what does the actual work in the task, and is awaited when the Task is run.

Return type None

sub_tasks = []

tags = ['all']

refit.task.**new_gathered_task**(*tasks*)

Task definitions are classes, not instances, hence why we require this.

Parameters **tasks** (Iterable[Type[*Task*]]) – A list of Task classes to execute.

Return type Type[*Concurrent*]

6.2 registry

class refit.registry.**HostRegistry**

get_host_classes (*environment*, *tags*)
Returns hosts matching the given tags.

If no tags are given, all hosts match.

Return type `Sequence[Type[Host]]`

register (**host_classes*, *environment='production'*, *tags=[]*)
Register hosts as possible deployment targets.

Return type `Union[Callable, Type[Host]]`

async run_tasks (*tasks*, *environment*)
Create and execute a Task for each matching host.

Return type `None`

class refit.registry.**TaskRegistry**

gather (**task_classes, tags=['all']*)
Register tasks, which will execute concurrently.

Return type `None`

register (**task_classes, tags=['all']*)
Register tasks for execution - used either directly, or as a decorator.

Return type `Union[Callable, Type[Task]]`

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX