headintheclouds Documentation

Release 0.3.7

Andreas Jansson

August 07, 2014

Contents

headintheclouds is a bunch of Fabric tasks for managing cloud servers and orchestrating Docker containers. Currently EC2 and Digital Ocean are supported.

CHAPTER 1

Install

headintheclouds has been tested on Linux and OSX. Installation should be as simple as

```
pip install headintheclouds
```

CHAPTER 2

Docs

2.1 Tutorial

In this tutorial we'll create a Wordpress server in EC2. First, create a new directory for the project. In that directory, create `fabfile.py` with the contents

```
# fabfile.py
from headintheclouds.tasks import *
from headintheclouds import ec2
from headintheclouds import ensemble
from headintheclouds import docker
```

Define environment variables with your EC2 credentials:

```
export AWS_ACCESS_KEY_ID=...
export AWS_SECRET_ACCESS_KEY=...
export AWS_SSH_KEY_FILENAME=...
export AWS_KEYPAIR_NAME=...
```

On the command line, type

```
fab nodes
```

The `nodes` task lists all the running nodes you've created. Since we haven't created any yet the output will look something like

```
ec2 name  size  ip  internal_ip  state  created


Done.
```

In the same directory, create a file called `wordpress.yml` with the contents

```
# wordpress.yml
wordpress:
  provider: ec2
  image: ubuntu 14.04
  size: m1.small
  containers:
    wordpress:
      image: jbfink/docker-wordpress
      ports:
        - 80
```

On the command line, type

```
fab ensemble.up:wordpress
```

The `ensemble` task figures out what needs to change in order to meet the wordpress.yml manifest. Since we don't have any servers yet, it will (with your permission) create a new m1.small server in EC2 and install Docker. Once that's done it will download and start the jbfink/docker-wordpress Docker container, exposing port 80.

Now if we type `fab nodes` again, we'll see the new server running

```
ec2 name       size      ip            internal_ip    state    created
wordpress  m1.small  54.198.33.85  10.207.25.187  running  2014-03-16 17:21:41-04:00
```

We can see all the running docker processes with

```
fab -R wordpress docker.ps
```

This will output

```
[54.198.33.85] Executing task 'docker.ps'
name       ip          ports          created              image
wordpress  172.17.0.6  80:80, 22:None  2014-03-16 21:53:22  jbfink/docker-wordpress
```

If we open that IP (54.198.33.85 in this case) in a browser we see the Wordpress welcome page.

If we type `fab ensemble.up:wordpress` again, headintheclouds will realise that no changes need to be made and will just exit. We can kill the wordpress process with

```
fab -R wordpress docker.kill:wordpress
```

Now if we do `fab ensemble.up:wordpress` it will only run the container but it won't start a new server.

That's pretty much it for a super basic tutorial. Let's kill the server

```
fab -R wordpress terminate
```

Now `fab nodes` will be empty again.

A more interesting Wordpress example can be found in the /examples directory.

2.2 Providers

At the moment headintheclouds supports

- EC2
- Digital Ocean
- "Unmanaged" servers (machines you provision yourself, e.g. bare metal boxes)

In order for commands like `fab nodes` to list servers for a specific provider you need to import the cloud providers you plan to use:

```
# fabfile.py
from headintheclouds.tasks import *
from headintheclouds import ec2
from headintheclouds import digitalocean
from headintheclouds import unmanaged
```

2.2.1 Provider-specific setup

EC2

To manage EC2 servers you need to define the following environment variables:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_SSH_KEY_FILENAME`
- `AWS_KEYPAIR_NAME`

Digital Ocean

To manage EC2 servers you need these environment variables:

- `DIGITAL_OCEAN_CLIENT_ID`
- `DIGITAL_OCEAN_API_KEY`
- `DIGITAL_OCEAN_SSH_KEY_FILENAME`
- `DIGITAL_OCEAN_SSH_KEY_NAME`

Unmanaged servers

You can't really "manage" unmanaged servers, but in order to be able to log in to them and run commands, you may need to define

- `HITC_SSH_USER` (defaults to `root`)
- `HITC_KEY_FILENAME` (defaults to `~/.ssh/id_rsa`)

Since headintheclouds has no way of finding out which servers it doesn't manage, you should put the public ips/hostnames of your servers in a file called `unmanaged_servers.txt` in the same directory as your fabfile. The servers should be one per line, e.g.

```
116.152.12.61
116.152.12.62
116.152.19.17
```

2.3 Tasks

All tasks are executed from the command line as

```
fab TASK_NAME:ARGUMENT_1,ARGUMENT_2,ARG_NAME_1=ARG_VALUE_1
```

To only execute the task on specific servers, use

```
fab -H PUBLIC_IP_ADDRESS TASK_NAME
```

to run the task on that one server, or

```
fab -R NAME TASK_NAME
```

to run the task on all servers with name `NAME`. This includes servers with names like `NAME-1`, `NAME-2`, etc.

2.3.1 Global tasks

`headintheclouds.tasks.`**`nodes`**
List running nodes on all enabled cloud providers. Automatically flushes caches

`headintheclouds.tasks.`**`create`**
Create one or more cloud servers

Args:

- provider (str): Cloud provider, e.g. ec2, digitalocean
- count (int) =1: Number of instances
- name (str) =None: Name of server(s)
- **kwargs: Provider-specific flags

`headintheclouds.tasks.`**`terminate`**
Terminate server(s)

`headintheclouds.tasks.`**`reboot`**
Reboot server(s)

`headintheclouds.tasks.`**`rename`**
Rename server(s)

Args: new_name (str): New name

`headintheclouds.tasks.`**`uncache`**
Flush the cache

`headintheclouds.tasks.`**`ssh`**
SSH into the server(s) (sequentially if more than one)

Args: cmd (str) =’‘: Command to run on the server

`headintheclouds.tasks.`**`upload`**
Copy a local file to one or more servers via scp

Args: local_path (str): Path on the local filesystem remote_path (str): Path on the remote filesystem

`headintheclouds.tasks.`**`pricing`**
Print pricing tables for all enabled providers

2.3.2 Provider-specific create flags

EC2

- size=’m1.small’: See `fab pricing` for details
- placment=’us-east-1b’
- bid=None: Define this to make spot requests
- image=’ubuntu 14.04’: Either an AMI ID or a shorthand Ubuntu version. The defined shorthands are ‘ubuntu 14.04’, ‘ubuntu 14.04 ebs’, ‘ubuntu 14.04 hvm’, where no ‘ebs’ or ‘hvm’ suffix indicate instance backing.
- security_group=’default’

Digital Ocean

- size='512MB': Can be 512MB, 1GB, 2GB, [...], 96GB. See `fab pricing` for details
- placement='New York 1': Any Digital Ocean region, e.g. 'Singapore 1', 'Amsterdam 2'
- image='Ubuntu 14.04 x64': Can be any Digital Ocean image name, e.g. 'Ubuntu 14.04 x64', 'Fedora 19 x64', 'Arch Linux 2013.05 x64', etc.

2.3.3 Caching

headintheclouds caches some data in PyDbLite, most importantly the list of active nodes. This is so that calls like `fab ssh` doesn't take several seconds to run before actually logging in. It's possible to get into weird situations when other users create servers and you have the old cache. To flush the cache you can run `fab uncache`. `fab nodes` and `fab ensemble.up` both flush the cache indirectly.

2.3.4 Namespacing

By default, all cloud servers created by headintheclouds will have their names prefixed by HITC-. This is so that headintheclouds-managed infrastructure doesn't interfere with other servers you might have. You can change this prefix by putting the line

```
env.name_prefix = 'MYPREFIX-'
```

after you import * from `fabric.api`, but **before importing headintheclouds**. So, for example

```
from fabric.api import *

env.name_prefix = 'INFRA-'

from headintheclouds import ec2
from headintheclouds import digitalocean
from headintheclouds import unmanaged
from headintheclouds import docker
from headintheclouds import ensemble
from headintheclouds.tasks import *
```

2.4 Docker

2.4.1 Working with private repositories

If you want to run containers from private Docker repos, you will have to be signed in to that repo. Authentication sessions are stored in a file called `~/.dockercfg` and are created by `docker login`.

headintheclouds can take care of most of this for you. If you create a file called `dot_dockercfg` that's a copy of your `~/.dockercfg`, the `fab docker.setup` command will upload this file to the remote host as `~/.dockercfg`.

2.4.2 Tasks

`headintheclouds.docker.`**`ssh`**
: SSH into a running container, using the host as a jump host. This requires the container to have a running sshd process.

Args:

- container: Container name or ID
- cmd='': Command to run in the container
- user='root': SSH username
- password='root': SSH password

headintheclouds.docker.**ps**
Print a table of all running containers on a host

headintheclouds.docker.**bind**
Bind one or more ports to the container.

Args:

- container: Container name or ID
- *ports: List of items in the format CONTAINER_PORT[:EXPOSED_PORT][/PROTOCOL]

Example: fab docker.bind:mycontainer,80,"3306:3307","12345/udp"

headintheclouds.docker.**unbind**
Unbind one or more ports from the container.

Args:

- container: Container name or ID
- *port: List of items in the format CONTAINER_PORT[:EXPOSED_PORT][/PROTOCOL]

Example: fab docker.unbind:mycontainer,80,"3306:3307","12345/udp"

headintheclouds.docker.**setup**
Prepare a vanilla server by installing docker, curl, and sshpass. If a file called `dot_dockercfg` exists in the current working directory, it is uploaded as `~/.dockercfg`.

Args:

- version=None: Docker version. If undefined, will install 0.7.6. You can also specify this in env.docker_version

headintheclouds.docker.**run**
Run a docker container.

Args:

- image: Docker image to run, e.g. orchardup/redis, quay.io/hello/world
- name=None: Container name
- command=None: Command to execute
- environment: Comma separated environment variables in the format NAME=VALUE
- ports=None: Comma separated port specs in the format CONTAINER_PORT[:EXPOSED_PORT][/PROTOCOL]
- volumes=None: Comma separated volumes in the format HOST_DIR:CONTAINER_DIR

Examples:

- fab docker.run:orchardup/redis,name=redis,ports=6379
- fab docker.run:quay.io/hello/world,name=hello,ports="80:8080,1000/udp",volumes="/docker/hello/log:/var/log"
- fab docker.run:andreasjansson/redis,environment="MAX_MEMORY=4G,FOO=bar",ports=6379

`headintheclouds.docker.`**`kill`**
Kill a container

Args:

- container: Container name or ID
- rm=True: Remove the container or not

`headintheclouds.docker.`**`pull`**
Pull down an image from a repository (without running it)

Args: image: Docker image

`headintheclouds.docker.`**`inspect`**
Inspect a container. Same as running `docker inspect CONTAINER` on the host.

Args: container: Container name or ID

`headintheclouds.docker.`**`tunnel`**
Set up an SSH tunnel into the container, using the host as a gateway host.

Args:

- container: Container name or ID
- local_port: Local port
- remote_port=None: Port on the Docker container (defaults to local_port)
- gateway_port=None: Port on the gateway host (defaults to remote_port)

2.5 Ensemble

This is really the sugar on the doughnut. headintheclouds.ensemble is an orchestration tool for Docker that will manage dependencies between containers, intelligently figure out what needs to change to meet the desired configuration, start servers and containers, and manage firewalls. It uses a simple YAML-based config format, and it's doing as much as possible in parallel.

I built ensemble on top of headintheclouds to manage thisismyjam.com. We're using it now for our production setup and it seems to hang together so far. The configuration format is heavily influenced by Orchard's Fig.

2.5.1 Tasks

`headintheclouds.ensemble.`**`up`**
Create servers and containers as required to meet the configuration specified in _name_.

Args:

- name: The name of the yaml config file (you can omit the .yml extension for convenience)

Example: fab ensemble.up:wordpress

2.5.2 Configuration YAML schema

```
# The name of the server. If count > 1, the names will be
# SERVER_NAME, SERVER_NAME-1, SERVER_NAME-2, [...].
# If SERVER_NAME is an IP address, it is implied that it is
# "unmanaged".
```

```
SERVER_NAME:

  # An optional template, see below.
  template: TEMPLATE

  # Provider is required unless SERVER_NAME is an IP address.
  # Valid options are currently ec2 and digitalocean
  provider: PROVIDER

  # Optional. The number of copies of this server will
  # be created. Default=1.
  count: COUNT

  # Provider-specific settings, see the section on
  # provider-specific create flags in the Tasks section
  # Examples for an EC2 instance:
  size: m1.small
  image: ubuntu 14.04
  security_group: web_ssh

  # The containers to run
  containers:

    # The name of the container. Again, if container count > 1,
    # names will be suffixed with '', '-1', '-2', etc.
    CONTAINER_NAME:

      # Required. E.g. orchardup/redis.
      image: IMAGE

      # Optional. A list of ports to open in the format
      # CONTAINER_PORT[:EXPOSED_PORT][/PROTOCOL]
      ports:

        # Examples:
        - 80
        - 3306:3366
        - 1234/tcp
        - 1234:2345/udp

      # Optional hash of environment variables to pass to
      # docker run
      environment:

        # Optional template for env vars
        template: TEMPLATE

        # Examples:
        FOO: BAR
        hello: 123

      # Optional hash of volumes to bound mount in the
      # format HOST_DIR:CONTAINER_DIR
      volumes:

        # Examples:
        /docker-vol/web/tmp: /tmp
        /data/logs: /var/log
```

```
      # The number of instances of this container to run.
      # Default=1
      count: COUNT

  # Optional firewall configuration. If defined, only the
  # ports specified here will be open, all others will be
  # closed.
  firewall:

    # firewall also accepts an optional template
    template: TEMPLATE

    # The open ports are defined as a hash of PORT[/PROTOCOL]
    # to IP or list of IPs or "*" or $internal_ips, e.g.:
    3306: 10.1.1.12
    8125/udp: 10.1.1.15

    # "*" opens a port to the world
    22: "*"

    # $internal_ips is a special variable (see Variables and
    # dependencies below) that will expand to a list of all
    # internal IPs for the servers in the same configuration
    # file, effectively opening a port to all of them.
    6379: $internal_ips

    # Ports can also be wildcarded, like this
    "*/*": $internal_ips

templates:
  TEMPLATE_NAME:
    # anything goes here
```

2.5.3 Templates

To avoid having to write the same chunk of YAML over and over again, templates can be used as a sort of preprocessor macro. Anything that is defined in the main configuration will override the value in the template. For example, if you have a config that looks like this

```
myserver:
  template: foo
  containers:
    template: bar

yourserver:
  template: foo
  containers:
    template: bar
    image: hello/world:other
    environment:
      template: baz

templates:
  foo:
    provider: digitalocean
    size: 1GB
```

```
  bar:
    image: hello/world
    ports:
      - 80:9000
    environment:
      HELLO: 123
  baz:
    WORLD: 456
```

it will expand to

```
myserver:
  provider: digitalocean
  size: 1GB
  containers:
    image: hello/world
    ports:
      - 80:9000
    environment:
      HELLO: 123

yourserver:
  provider: digitalocean
  size: 1GB
  containers:
    image: hello/world:other
    ports:
      - 80:9000
    environment:
      WORLD: 456
```

2.5.4 Variables and dependencies

Often you want to connect containers and servers, but you probably don't know the address of the server or container in advance. Enter variables and dependency management!

Here's an example:

```
web:
  provider: ec2
  containers:
    web:
      image: hello/web
    ports:
      - 80
    environment:
      REDIS_HOST: ${redis.ip}

redis:
  provider: ec2
  containers:
    redis:
      image: orchardup/redis
    ports:
      - 6379
```

When you "up" this ensemble manifest from a vanilla setup with no running servers, the order of operations will be:

- Start "web" and "redis" servers in parallel

- Resolve `${redis.ip}` to the actual IP of the redis server
- Start the redis and web containers in parallel

If the web container would need to wait for the redis **container** to start, you could put in an environment variable like

```
# [snip]
    web:
      environment:
        REDIS_HOST: ${redis.ip}
        _DEPENDS: ${redis.containers.redis.ip}
```

headintheclouds.ensemble abstracts all the scheduling and will complain if you try to set up cyclical dependencies, so you can set up pretty complex dependency graphs without thinking too much about what's going on behind the scenes.

As a side note, headintheclouds doesn't use docker links, instead you point containers to the IPs of other servers and containers.

2.5.5 Idempotence and statelessness

The only state that headintheclouds keeps is the internal caches, and these can be wiped without any negative side effects. Instead of storing state locally, the state of servers and containers is interrogated on the fly by logging in to the servers and checking what is actually running.

When you run `fab ensemble.up:myensemble`, it will log in to any existing servers with the same names as in the manifest, and check if they're equivalent to what the configuration says. Then it will check the Docker containers and firewall rules on each host to see if they match the manifest.

This is how headintheclouds.ensemble is idempotent. You can run `fab ensemble.up:myensemble` any number of times with no effect on your servers, provided you don't change `myensemble.yml`.

Before going out starting servers and containers, headintheclouds will prompt you to confirm the changes that will be made.

The only caveat is that headintheclouds doesn't currently delete servers and containers if you remove them from the manifest, you have to do that manually with the `terminate` and `docker.kill` commands. That's just so you don't go and tear things don't by accident.

2.5.6 Server names and roles

If you have a conf YAML file like this

```
foo:
  provider: ec2
  [snip]

bar:
  [snip]
  containers:
    blah:
      image: some/image
      environment:
        FOO_NAME: ${foo.name}
  count: 2
```

`foo` and `bar` are the *names* of the servers. But when using Fabric, *role* is synonymous with name. So you could do

```
fab -R bar ping
```

to ping both of the `bar` server. To access a single one, you'd have to use the `-H` Fabric flag, e.g.

```
fab -H 123.45.67.89 ssh
```

(assuming 123.45.67.89 is the IP of one of the `bar` servers).

h