

---

# gonzo Documentation

*Release 0.4.2*

**onefinestay**

**Oct 09, 2018**



---

## Contents

---

|           |  |           |
|-----------|--|-----------|
| <b>1</b>  | <b>Configuration</b>                   | <b>3</b>  |
| <b>2</b>  | <b>Command Line Interface</b>          | <b>5</b>  |
| <b>3</b>  | <b>Managing the instance lifecycle</b> | <b>7</b>  |
| <b>4</b>  | <b>Using gonzo with fabric</b>         | <b>9</b>  |
| <b>5</b>  | <b>Fabric task library</b>             | <b>11</b> |
| <b>6</b>  | <b>Using Gonzo With CloudInit</b>      | <b>13</b> |
| <b>7</b>  | <b>TODO</b>                            | <b>15</b> |
| <b>8</b>  | <b>Build status</b>                    | <b>17</b> |
| <b>9</b>  | <b>License</b>                         | <b>19</b> |
| <b>10</b> | <b>More Docs</b>                       | <b>21</b> |
| <b>11</b> | <b>Contents</b>                        | <b>23</b> |
| 11.1      | configuration . . . . .                | 23        |
| 11.2      | Cloud backends . . . . .               | 24        |
| <b>12</b> | <b>Indices and tables</b>              | <b>25</b> |



Instance and release management made easy

Manage instances running in *Amazon Web Services* or using *Openstack* using a single consistent interface:

```
$ gonzo launch production-web-app
...
$ gonzo list
```

| name                   | location   | type      | status  | owner | uptime        |
|------------------------|------------|-----------|---------|-------|---------------|
| production-web-app-001 | eu-west-1b | m1.small  | RUNNING | tom   | 0d 0h 1m 18s  |
| production-web-db-001  | eu-west-1a | m3.medium | RUNNING | amit  | 27d 1h 8m 13s |

Easily target instances or groups of instances with `fab` commands and manage your code deployments using included fabric tasks:

```
$ fab gonzo.group:production-ecommerce-web push_release rollforward
```



# CHAPTER 1

---

## Configuration

---

Setup your clouds





## CHAPTER 2

---

### Command Line Interface

---

Having setup multiple cloud environments and/or regions within, use the `gonzo config` command to chose where you want to deploy servers or projects to:

```
$ gonzo config
cloud: None
region: None
$ gonzo config --cloud aws
cloud: aws
region: us-west-1
$ gonzo config --region eu-west-1
cloud: aws
region: eu-west-1
```



## CHAPTER 3

---

### Managing the instance lifecycle

---

Having chosen the cloud and region you want to work within you can issue gonzo commands to control the spinning up, monitoring and termination of instances within.

To see a list of all running instance in the region:

```
$ gonzo list
name                type      status  owner      uptime
↪      location
production-web-app-001  m1.small  RUNNING  tom        408d 0h 42m
↪18s      eu-west-1a
production-web-app-002  m3.medium  RUNNING  amit       27d 1h 8m 13s
↪      eu-west-1b
production-web-db-011  m3.medium  RUNNING  amit       160d 2h 33m
↪18s      eu-west-1c
```

To add a new instance to the region, specifying the server type - having defined server types, and their sizes in your config:

```
$ gonzo launch production-web
```

To get more info on the commands available:

```
$ gonzo --help
```



## CHAPTER 4

---

### Using gonzo with fabric

---

You can then use gonzo to set a target server (or group of servers) for [fabric](#) commands.

Import gonzo in your fabfile to extend fab with gonzo functionality:

```
$ cat fabfile.py

from gonzo.tasks import gonzo
__all__ = ['gonzo']
```

You can then run a command on a single instance, specifying it through gonzo:

```
$ fab gonzo.instance:production-web-003 run_command
```

Or run the command on a group of instances:

```
$ fab gonzo.group:production-web run_command
```



## CHAPTER 5

---

### Fabric task library

---

To use the gonzo library of fabric tasks, simply import the relevant task modules for namespaced tasks into your fabfile:

```
from gonzo.tasks import apache
```

These can then be called using the standard fabric syntax:

```
$ fab gonzo.group:production-web apache.restart
```

Alternatively import the tasks directly:

```
from gonzo.tasks.apache import restart
```

These commands won't be namespaced:

```
$ fab gonzo.group:production-web restart
```

You can extend the functionality by patching your own commands into the gonzo namespaces to provide a clean CLI:

```
# ~/apache_maintenance_mode.py
from fabric.api import task, sudo
from gonzo.tasks import apache

def maintenance_mode(off=False):
    """ Set server into maintenance mode.
    """

    if off:
        sudo("a2ensite onefinestay && a2dissite 00maintenance")
    else:
        sudo("a2ensite 00maintenance && a2dissite onefinestay")

apache.maintenance_mode = task(maintenance_mode)
```





---

## Using Gonzo With CloudInit

---

CloudInit can be used to personalise the instances you launch. The user data scripts passed to new instances for CloudInit to process can be specified for each cloud by using the `DEFAULT_USER_DATA` config item in `config.py`:

```
CLOUDS = {
    'cloudname': {
        ...
        'DEFAULT_USER_DATA': 'http://example.com/my-cloudinit-config.txt',
        ...
    }
```

Additionally, user data scripts can be specified per instance by using the launch argument `--user-data <file | url>`:

```
# gonzo launch --user-data ~/.gonzo/cloudinit_web_app production-web-app
```

User data scripts can be specified as a file path or URL.

Before user data scripts are passed to new instances, they're first rendered as a template, allowing them to be parameterised. By default a few are already available, such as `hostname`, `domain` and `fqdn`. These can be supplemented by defining a `USER_DATA_PARAMS` cloud config dictionary:

```
CLOUDS = {
    'cloudname': {
        ...
        'DEFAULT_USER_DATA': 'http://example.com/my-cloudinit-config.txt',
        'USER_DATA_PARAMS': {
            'puppet_address': 'puppetmaster.example.com',
        }
        ...
    }
```

Again, these parameters can also be supplemented or overridden at launch time by using the command line argument `--user-data-params key=val[,key=val...]`:

```
# gonzo launch --user-data ~/.gonzo/cloudinit_web_app \  
  --user-data-params puppet_address=puppetmaster2.example.com \  
  production-web-app
```

## CHAPTER 7

---

TODO

---

- **project based stuff**
  - project name [for /srv/project\_name] (git setting?)
  - Document how to use for release control



## CHAPTER 8

---

Build status

---

build passing



## CHAPTER 9

---

### License

---

Apache 2.0 - see LICENSE for details





## CHAPTER 10

---

[More Docs](#)

---

[Full documentation on Read the Docs](#)



## 11.1 configuration

Configuring gonzo

### 11.1.1 Configuration

Place your configuration in `~/gonzo/config.py` following this structure:

```
""" Define your cloud connections.

Currently supports Openstack or AWS based clouds. Specify each backend you
use, and matching authentication details.
"""

CLOUDS = {
    'default': {
        # For Openstack based clouds
        'BACKEND': 'openstack',
        # For AWS based clouds
        # 'BACKEND': 'ec2',

        # Openstack authentication details
        'TENANT_NAME': 'service',
        'USERNAME': '',
        'PASSWORD': '',
        'AUTH_URL': "http://cloud-host.example.com:5000/",

        # AWS authentication details
        #
        # Always required! When working with OpenStack, keys will be used for
        # for recording hostnames and counting server types with Route53.
        'AWS_ACCESS_KEY_ID': '',
```

(continues on next page)

(continued from previous page)

```
'AWS_SECRET_ACCESS_KEY': '',

# Common to either clouds backend
# Glance or AMI image ID
'IMAGE_ID': 'ami-f3bea887',

# Regions to deploy to, new instances will be evenly distributed
'REGIONS': ['RegionOne', ],

# Key to be injected into new instances by the cloud provider
# You typically upload the public key to the provide and give it a
# name for internal reference
'PUBLIC_KEY_NAME': 'master',
'PRIVATE_KEY_FILE': "~/.ssh/id_rsa",

# domain to hold host information
'DNS_ZONE': 'example.com',
'DNS_TYPE': 'CNAME',

# Default cloud-init script to pass when creating new instances.
# Can be overridden with --user-data.
# Will be parsed as a template. See templates/userdata_template for
# more info.
'DEFAULT_USER_DATA': None,
# Extra params to use when rendering user data template.
'USER_DATA_PARAMS': {},
},

}

""" Define the size of each server type you want to deploy

``default`` is required. Server types listed will use the specified instance
sizes. The list of server types will also be used for bash completion.
"""

SIZES = {
    'default': 'm1.small',
    'ecommerce-web': 'm1.large',
}
```

## 11.2 Cloud backends

### 11.2.1 Interface (abstract)

### 11.2.2 Amazon Web Services

### 11.2.3 Openstack

## CHAPTER 12

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`