Ansible

and SaltStack

Mantas Zimnickas

VilniusPHP 0x21

2015-08-06

Who am I?

- I'm not a server administrator.
- 2002 - 2010 - started my career as a PHP developer.
- 2010 - switched to Python.
- Mostly I do backend web development.

What is Ansible?

- Multi-node software deployment and configuration management tool.
- In other words..

  You can deploy your app multiple times on different servers with one command.

Ansible compared to others

- **Fabric**

  You can learn it in 5 minutes, better than shell scripting.

- **Anslibe**

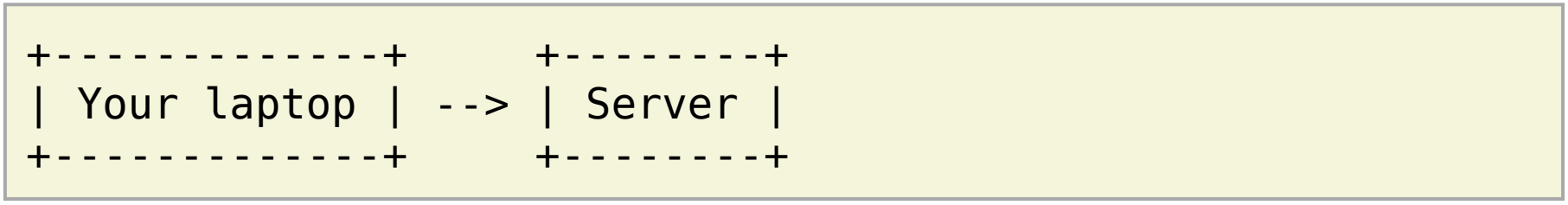
  You can learn it in one day, better than Fabric scripting.

- **SaltStack**, **Chef**, **Puppet** and friends

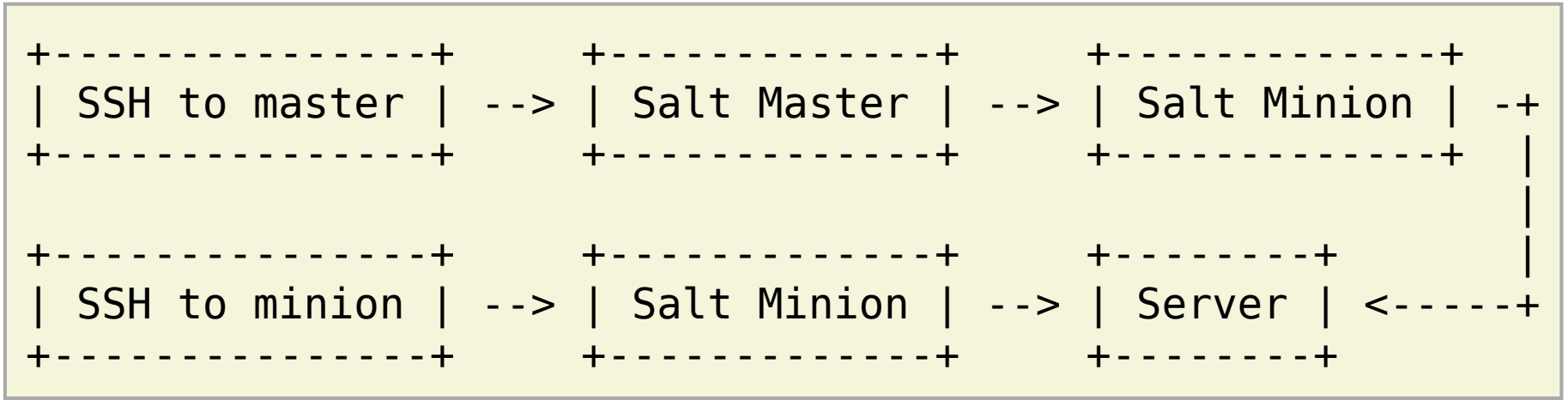
  You can learn it in two days, better than Ansible if you have a cloud of servers.

Ansible compared to SaltStack

Ansible



SaltStack



How I use Ansible

- One server.
- Many projects.
- Very few users using those projects.
- Apache or Nginx with mod_wsgi or uWSGI.
- PostgreSQL database (MySQL for older projects).
- Mostly Python 3.

My expirience with Ansible

akl.lt: Python 3, Nginx, uWSGI, PostgreSQL

https://github.com/python-dirbtuves/akl.lt/tree/master/deployment

atviriduomenys.lt: Python 3, Apache, mod_wsgi, PostgreSQL

https://github.com/akllt/infrastructure/tree/master/websites/atviriduomenys.lt

manoseimas.lt: Python 2, Apache, mod_wsgi, MySQL, CouchDB

https://github.com/ManoSeimas/manoseimas.lt/tree/master/deployment

pylab.lt: Python 3, Apache, mod_wsgi, PostgreSQL

https://github.com/akllt/infrastructure/tree/master/websites/pylab.lt

My expirience with SaltStack

pylab.lt: Python 3, Apache, mod_wsgi, PostgreSQL

https://github.com/akllt/infrastructure/tree/saltstack/websites/pylab.lt

How to install Ansible?

- If you run a *Debian* based distro:

```
$ apt install ansible
```

- `pip install` also works:

```
$ pip install ansible
```

Simplest possible way to make Ansible do something:

```
$ ansible host -c local -i host, -m ping
host | success >> {
    "changed": false,
    "ping": "pong"
}
```

- **ansible** command
- on **host**
- defined in **-i host,** inventory line
- using **-c local** connection backend
- runs **-m ping** module

Ansible Playbooks

playbook.yml:

```
---
- hosts: host
  tasks:
  - ping:
```

```
$ ansible-playbook -c local -i host, playbook.yml
```

It does same thing as:

```
$ ansible host -c local -i host, -m ping
```

Let's add some defaults using **ansible.cfg**:

```
[defaults]
inventory = inventory.cfg
```

inventory.cfg:

```
host ansible_connection=local
```

Now, I don't have to specify inventory file and connection:

```
$ ansible host -m ping
```

```
$ ansible-playbook playbook.yml
```

Modules have arguments:

```
$ ansible host -m command -a uptime
host | success | rc=0 >>
 up 1 day, 1:17
```

Default module is `command`:

```
$ ansible host -a uptime
host | success | rc=0 >>
 up 1 day, 1:18
```

Argument can be a YAML expression or `key=value` string.

Same thing using playbook **playbook.yml**:

```
---
- hosts: host
  gather_facts: no
  tasks:
  - command: uptime
```

```
$ ansible-playbook playbook.yml
PLAY [host] ********************************************************

TASK: [command uptime] *********************************************
changed: [host]

PLAY RECAP *********************************************************
host : ok=1  changed=1  unreachable=0  failed=0
```

Playbook structure

```
---
- hosts: host group name

  vars: a dict of variables

  tasks: list of tasks

  handlers: list of handlers
```

Roles

```
roles/
   role/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
```

playbook.yaml:

```
---
- hosts: host
  roles:
  - role
```

Writing your own modules

mymodule.py:

```
from ansible.module_utils.basic import *

def main():
    module = AnsibleModule(
        argument_spec = dict(
            state     = dict(default='present', choices=['present', 'absent']),
            name      = dict(required=True),
            enabled   = dict(required=True, choices=BOOLEANS),
            something = dict(aliases=['whatever'])
        )
    )

    module.exit_json(changed=True, something_else=12345)

if __name__ == '__main__':
    main()
```

Ansible Galaxy

https://galaxy.ansible.com/

```
$ ansible-galaxy install rolename
```

Category	Total Roles
system	1421
development	788
web	721
monitoring	289
networking	258
packaging	248
database	189
...	...

Ansible: apt module

```
apt: pkg={{ item }} state=latest
with_items:
- build-essential
- postgresql
- python-psycopg2
- python-dev
- python-pip
- python-virtualenv
- apache2
- libapache2-mod-wsgi-py3
- git
```

SaltStack: apt module

```
myproject:
  pkg.installed:
    - pkgs:
      - build-essential
      - postgresql
      - python-psycopg2
      - python-dev
      - python-pip
      - python-virtualenv
      - apache2
      - libapache2-mod-wsgi-py3
      - git
```

Ansible: user module

```
user: >
  name=myproject
  system=yes
  group=www-data
  home={{ home }}
```

For this to work, you need `sudo: yes` and `home` variable:

```
hosts: host
sudo: yes
vars:
  home: /opt/myproject
```

SaltStack: user module

```
{% set home = '/opt/myproject' %}

 myproject:
   user.present:
     - gid: {{ salt['group.info']('www-data').gid }}
     - home: {{ home }}
     - system: yes
```

Ansible: PostgreSQL modules

```
- postgresql_db: name=myproject
  sudo_user: postgres

- postgresql_user: db=myproject name=myproject
  sudo_user: postgres
```

SaltStack: PostgreSQL modules

```
myproject:

  postgres_user.present:
    - require:
      - pkg: myproject
      - user: myproject

  postgres_database.present:
    - owner: myproject
    - require:
      - pkg: myproject
      - postgres_user: myproject
```

Ansible: template module

```
template: >
  src=templates/apache.conf
  dest=/etc/apache2/sites-enabled/myproject.conf
notify: reload apache
```

```
handlers:
- name: reload apache
  service: name=apache2 state=reloaded
```

SaltStack template module

```
{% set home = '/opt/myproject' %}
{% set path = home + '/app' %}
{% set server_name = salt['pillar.get']('server_name',
                                        'myproject.lt') %}

/etc/apache2/sites-enabled/myproject.conf:
  file.managed:
    - template: jinja
    - source: salt://apache.conf
    - context:
        server_name: {{ server_name }}
        path: {{ path }}

apache2:
  service.running:
    - watch:
      - file: /etc/apache2/sites-enabled/myproject.conf
```

Ansible: Dealing with passwords

```
- stat: path=/root/.my.cnf
  register: root_my_cnf

- mysql_user: >
    name=root host=localhost state=present
    password={{ lookup('password', 'secrets/mysqlroot') }}
  when: not root_my_cnf.stat.exists

- template: >
    src=templates/root_my.cnf
    dest=/root/.my.cnf owner=root mode=0600
  when: not root_my_cnf.stat.exists
```

```
[client]
user = root
password = {{ lookup('password', 'secrets/mysqlroot') }}
default-character-set = utf8
```

Ansible: git module

```
git: >
  repo=https://github.com/me/myproject
  dest={{ path }}
  force=yes
notify: reload source code
sudo_user: myproject
```

```
handlers:
- name: reload source code
  command: touch --no-create {{ path }}/bin/django.wsgi
```

Ansible: git module

```
myproject:
  git.latest:
    - name: https://github.com/me/myproject
    - target: {{ path }}
    - user: myproject
    - rev: master
    - require:
      - pkg: myproject
      - user: myproject

reload:
  cmd.wait:
    - name: touch --no-create {{ path }}/bin/django.wsgi
    - user: myproject
    - watch:
      - git: myproject
```

Ansible: command module

```
command: bin/django migrate --noinput chdir={{ path }}
sudo_user: myproject
```

```
command: bin/django collectstatic --noinput chdir={{ path }}
sudo_user: myproject
```

SaltStack: command module

```
migrate:
  cmd.wait:
    - name: bin/django migrate --noinput
    - cwd: {{ path }}
    - user: myproject
    - watch:
      - git: myproject
    - require:
      - cmd: make
      - postgres_database: myproject
```

Ansible: Environments

```
vars:
  vars: production

vars_files:
  - vars/{{ vars }}.yml
```

Changing environment from command line:

```
$ ansible-playbook playbook.yml -e vars=staging
```

SaltStack: Environments

/etc/salt/minion-id:

```
production
```

pillar/top.sls:

```
base:
  production:
    - production
```

pillar/production.sls:

```
server_name: myproject.lt
```

states/myproject.sls:

```
{% set server_name = salt['pillar']['server_name'] %}
```

Testing deployment scripts

Vagrantfile:

```
Vagrant.configure('2') do |config|
  config.vm.define 'box' do |box|
    box.vm.box = 'ubuntu/trusty64'
    box.vm.network :forwarded_port, guest: 80, host: 8080
    box.vm.synced_folder '.', '/vagrant', disabled: true
    config.vm.provision "ansible" do |ansible|
      ansible.playbook = "deploy.yml"
      ansible.extra_vars = {
        vars: "vagrant",
      }
    end
  end
end
```

```
vagrant provision
```

One command to deploy

Ansible:

```
$ ansible-playbook deploy.yml
```

Master-less SaltStack:

```
$ ssh server
$ sudo salt-call --config-dir=/srv/salt/myproject \
                 --local state.highstate
```

Conclusions: Ansible

Pros

- Quite easy to learn.
- Easy to set up.
- Better than Fabric or shell scripting (thanks to many modules).

Cons

- Very slow.

Conclusions: SaltStack

Pros

- Faster than Ansible.
- Cleaner and more flexible configuration.

Cons

- Requires more time to understand the big picture.

  And requires a lot more time to understand the whole picture.

- Requires extra time set up minion.

Thank you
for your attention.