Ansible

...

Hands on Training

Agenda

- Fundamentals
- Key Components
- Best practices
- Spring Boot REST API Deployment
- CI with Ansible
- Ansible for AWS
- Provisioning a Docker Host
- Docker&Ansible

Fundamentals

...

Fundamentals

- What is Ansible?
- Why Ansible?
- Terms
  - Inventory
  - Host
  - Group
  - Playbook
  - Play
  - Task
  - Modules
  - Library

What is Ansible?

- Radically simple IT automation engine that automates
  - Cloud provisioning
  - Configuration management
  - Application deployment
  - Intra-service orchestration

Why Ansible?

- Simple
  - Easy to write, read, maintain and evolve- without writing scripts or custom code
- Fast to learn and setup
  - It uses a very simple language (YAML, in the form of Ansible Playbooks) that allow you to describe your automation jobs in a way that approaches plain English.

Why Ansible?

- Efficient
  - Doesn't require a custom agent or software to install
  - Ansible works by connecting to your nodes and pushing out small programs, called "Ansible modules" to them.
- Secure
  - No agent
  - Runs on OpenSSH

Inventory

Ansible works against multiple systems in your infrastructure at the same time. It does this by selecting portions of systems listed in Ansible's inventory file, which defaults to being saved in the location /etc/ansible/hosts.

```
[webservers]
192.168.35.140
192.168.35.141
192.168.35.142
192.168.35.143
```

```
[appservers]
192.168.100.1
192.168.100.2
192.168.100.3
```

```
[dbservers]
172.35.0.5
```

Host

A host is simply a remote machine that Ansible manages. They can have individual variables assigned to them, and can also be organized in groups.

[webservers]
192.168.35.140
192.168.35.141
192.168.35.142
192.168.35.143

[appservers]
192.168.100.1
192.168.100.2
192.168.100.3

[dbservers]
172.35.0.5

Group

A group consists of several hosts assigned to a pool that can be conveniently targeted together, and also given variables that they share in common.

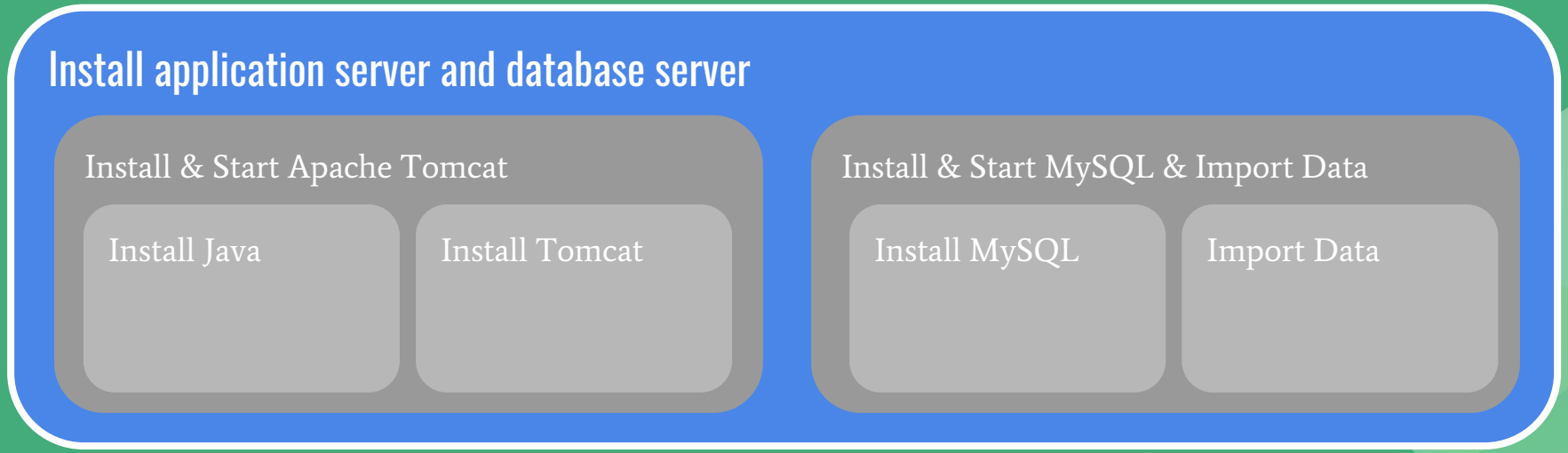
[webservers]
192.168.35.140
192.168.35.141
192.168.35.142
192.168.35.143

[appservers]
192.168.100.1
192.168.100.2
192.168.100.3

[dbservers]
172.35.0.5

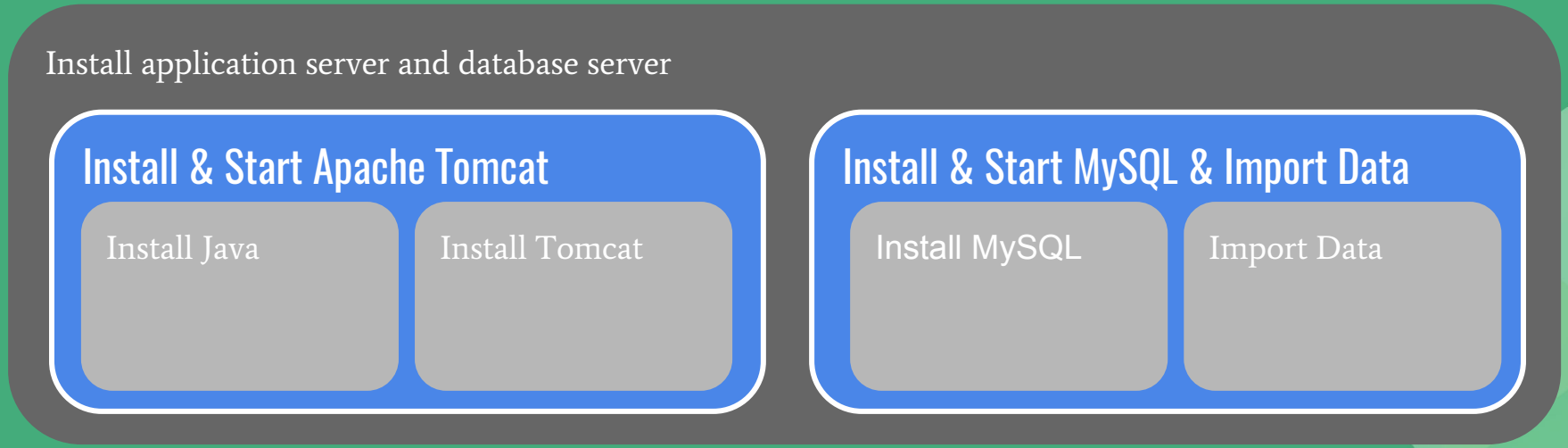
Playbook

Playbooks are the language by which Ansible orchestrates, configures, administers, or deploys systems. Playbooks contain Plays.



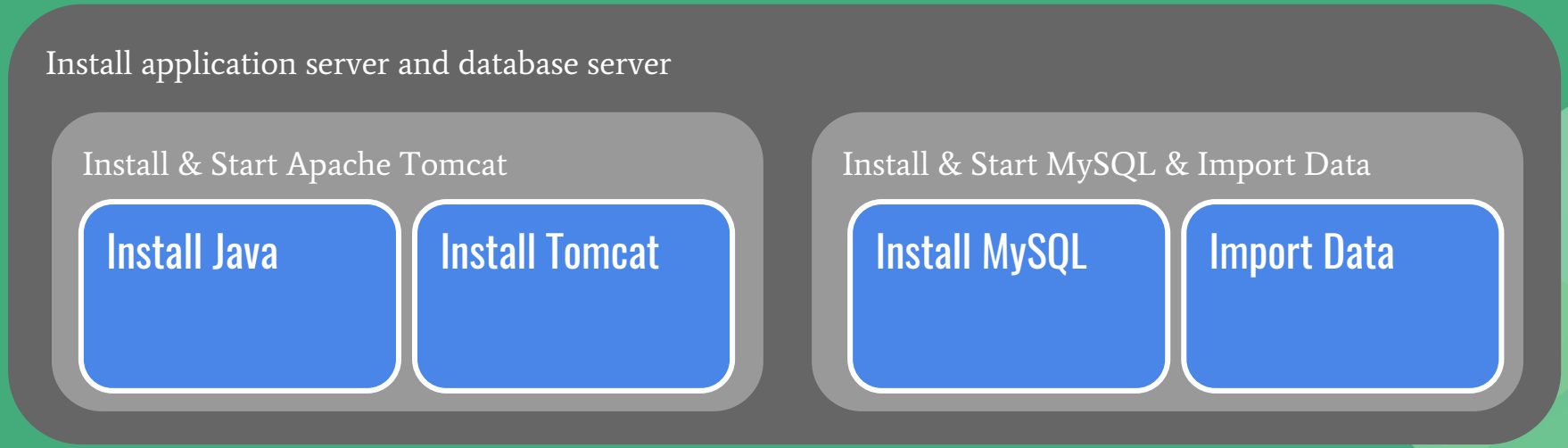
Play

A play is a mapping between a set of hosts selected by a host specifier and the tasks which run on those hosts to define the role that those systems will perform.



Task

Tasks combine an action with a name and optionally some other keywords (like looping directives). Tasks call modules .



Module

Modules are the units of work that Ansible ships out to remote machines. Ansible refers to the collection of available modules as a library.

Install Java

Download Oracle JDK

```
get_url:

url:http://download.oracle.com
    dest:jdk-1.8.0-linux-x64.rpm
```

Install Oracle JDK

```
yum:
    name:
jdk-1.8.0-linux-x64.rpm
    state: present
```

Library

A collection of modules made available to /usr/bin/ansible or an Ansible playbook.

Install Ansible

Latest Release via Yum

```
# install the epel-release RPM if needed on CentOS, RHEL, or Scientific Linux
$ sudo yum install ansible
```

Latest Release via Apt

```
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install ansible
```

Install Ansible

Latest Release via Pip

```
$ sudo easy_install pip
$ sudo pip install ansible
```

Control Machine System Requirements

Currently Ansible can be run from any machine with *Python 2.6 or 2.7* installed (Windows isn't supported for the control machine).

Node Machine System Requirements

On the managed nodes, you need a way to communicate, which is normally *ssh*. By default this uses sftp. If that's not available, you can switch to scp in ansible.cfg.

You also need Python 2.4 or later. If you are running less than *Python 2.5* on the remotes, you will also need:

- python-simplejson

LAB #1

...

Install Ansible

Install Ansible

Let's start with cloning the repository we will walk during the trainig

```
$ git clone https://github.com/maaydin/ansible-tutorial.git
$ cd ansible-tutorial
```

Provision the Control Machine and install ansible

```
$ vagrant up
$ vagrant ssh control
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible
$ sudo apt-get update
$ sudo apt-get install ansible
```

Validate Ansible Installation

Check the ansible version

```
$ ansible --version
ansible 2.2.0.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = Default w/o overrides
```

Ad-Hoc Commands on Local Machine

Ping the localhost

```
$ ansible -m ping localhost
 [WARNING]: provided hosts list is empty, only localhost is available

localhost | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Key Components

...

Inventory

...

Inventory Concepts

- Hosts & Groups
- Host & Group Variables
- Groups of Groups
- Inventory Parameters
- Dynamic Inventory

Hosts & Groups

The format for */etc/ansible/hosts* is an INI-like format and looks

```
mail.example.com


[webservers]
foo.example.com
bar.example.com


[dbservers]
one.example.com
two.example.com
three.example.com
```

Hosts & Groups

Different SSH port:

```
web1:2222
```

Using aliases:

```
web2 ansible_port=22 ansible_host=192.168.35.102
```

Ranges:

```
[webservers]
www[01:50].example.com
```

Host & Group Variables

Assign variables to hosts that will be used later in playbooks

```
[webservers]
web1 http_port=80 https_port=443
web2 http_port=8080 https_port=8443
```

Variables can also be applied to an entire group at once

```
[webservers:vars]
ntp_server=tr.pool.ntp.org
proxy=proxy.example.com
```

Groups of Groups

To make groups of groups use the *:children* suffix.

```
[euwest]
host1

[eucentral]
host2

[eu:children]
euwest
eucentral
```

Inventory Parameters

ansible_user

The default ssh user name to use.

ansible_ssh_private_key_file

Private key file used by ssh. Useful if using multiple keys and you don't want to use SSH agent.

ansible_become

Equivalent to ansible_sudo or ansible_su, allows to force privilege escalation

Dynamic Inventory

Inventory can also be gathered on demand from other sources dynamically. Those sources include:

- Cobbler (http://cobbler.github.io/)
- Cloud APIs
  - Rackspace
  - Amazon
  - Digital Ocean
  - OpenStack

LAB #2

...

Create the Inventory

Create the Hosts in the Inventory

Create the Ansible Inventory for given hosts:

```
web1 192.168.35.101
web2 192.168.35.102
app 192.168.35.103
db 192.168.35.104
```

Create the Groups in the Inventory

Create the Inventory for given groups consist of below servers & groups

```
webservers: web1 & web2
appservers: app
dbservers: db
dc: webservers & appservers & dbservers
```

Inventory

```
web1 ansible_host=192.168.35.101
web2 ansible_host=192.168.35.102
app ansible_host=192.168.35.103
db ansible_host=192.168.35.104

[webservers]
web1
web2

[appservers]
app

[dbservers]
db

[dc:children]
webservers
appservers
dbservers
```

Ad-Hoc Commands on Inventory

Ping the hosts and groups you defined

```
$ ansible -m ping web1
$ ansible -m ping app
$ ansible -m ping webservers
$ ansible -m ping dc
```

Tip #1: SSH Keys

To set up SSH agent to avoid retyping passwords, you can add the private key

```
$ vagrant ssh control
$ ssh-agent bash
$ ssh-add /vagrant/keys/key
```

Creating a New SSH Key Pair

```
$ ssh-keygen
```

Tip #2: Host Key Checking

If you wish to disable host key checking, you can do so by editing

/etc/ansible/ansible.cfg or *~/.ansible.cfg*:

```
[defaults]
host_key_checking = False
```

Alternatively this can be set by an environment variable:

```
$ export ANSIBLE_HOST_KEY_CHECKING=False
```

Ad-Hoc Commands on Inventory

Run some shell commands on the hosts and groups you defined

```
$ ansible -m shell -a 'ls -al' web1
$ ansible -m shell -a 'whoami' app
$ ansible -m shell -a 'ifconfig' webservers
$ ansible -m shell -a 'hostname' dc
```

Tip #3: Patterns

A pattern usually refers to a set of groups (which are sets of hosts)

```
$ ansible -m ping all
$ ansible -m ping web*
$ ansible -m ping 'appservers:dbservers'
$ ansible -m ping 'dc:!webservers'
$ ansible -m ping 'dc:&webservers'
```

Tasks

...

Tasks

A task is a discrete action that is a declaration about the state of a system.

- Example Tasks:
- Directory should exist
- Package should be installed
- Service should be running
- Cloud Instance should exist

Tasks as Ad-Hoc Commands

Ansible can execute single tasks on sets of hosts to full-fill an ad-hoc declarations.

```
$ ansible webservers -m file -a "path=/var/www/html/assets state=directory"
$ ansible webservers -m apt -a "name=nginx state=present"
$ ansible webservers -m service -a "name=nginx enabled=yes state=started"
```

Modules

...

Modules

Modules are the bits of code copied to the target system to be executed to satisfy the task declaration.

- Code need not exist on remote host -- ansible copies it over
- Many modules come with Ansible -- "batteries included"
- Custom modules can be developed easily
- Command/shell modules exists for simple commands
- Script module exists for using existing code
- Raw module exists for executing raw commands over ssh

Modules Documentation

- Module listing and documentation via ansible-doc

```
$ ansible-doc -l
$ ansible-doc apt
```

- Module index

http://docs.ansible.com/ansible/modules_by_category.html

LAB #3

...

Using Common Modules

Install Nginx with Ad-Hoc Commands

Install the nginx server on webservers with apt module

```
$ ansible -m apt -a "name=nginx state=present update_cache=yes" web1
```

Tip #4: Become (Privilege Escalation)

Ansible can use existing privilege escalation systems to allow a user to execute tasks as another.

Ansible allows you to ‘become’ another user, different from the user that logged into the machine (remote user). This is done using existing privilege escalation tools, which you probably already use or have configured, like sudo, su, pfexec, doas, pbrun, dzdo, ksu and others.

```
$ ansible -m shell -a "whoami" web1 --become
```

Install Nginx with Ad-Hoc Commands

Install the nginx server on webservers with apt module

```
$ ansible -m apt -a "name=nginx state=present update_cache=yes" web1 --become
```

Ensure service enabled and started on webservers with service module

```
$ ansible -m service -a "name=nginx state=started enabled=yes" webservers
--become
```

Install Nginx with Ad-Hoc Commands

Ensure /usr/share/nginx/html directory exists on webservers with file module

```
$ ansible -m file -a "path=/usr/share/nginx/html state=directory" webservers
--become
```

Update /usr/share/nginx/html/index.html file a custom file with copy module

```
$ ansible -m copy -a "src=index.html dest=/usr/share/nginx/html/index.html"
webservers --become
```

Modules Exercises

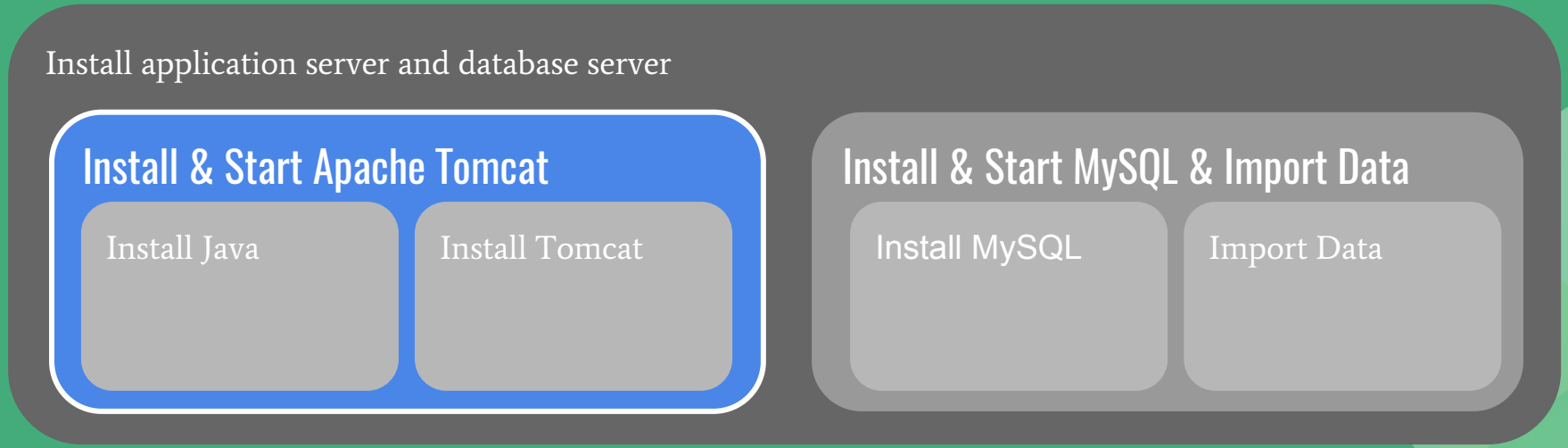
- Ensure *default-jdk* package installed on appservers.
- Ensure *greeting* user created on appservers.
- Ensure */var/log/greeting* directory owned by *greeting* user created on appservers.
- Ensure *mongodb-server* package installed on dbservers.

Plays

...

Plays

Plays are ordered sets of tasks to execute against host selections from your inventory.



Plays

```
---
- name: Nginx Play
  hosts: webservers
  vars:
    assets_dir: /var/www/html/static
  become: true
  tasks:
  - name: ensure nginx is installed
    apt: name=nginx state=present
  - name: ensure directory exists
    file: path={{ assets_dir }} state=directory
```

Naming

Variables

Tasks

Host Selection

Privilege Escalation

Conditionals

```
tasks:
  - command: /bin/false
    register: result
    ignore_errors: True

  - command: /bin/something
    when: result|failed

  - command: /bin/something_else
    when: result|succeeded

  - command: /bin/still/something_else
    when: result|skipped
```

Loops

```
tasks:
  - command: /bin/false
    register: result
    ignore_errors: True

  - command: /bin/something
    when: result|failed

  - command: /bin/something_else
    when: result|succeeded

  - command: /bin/still/something_else
    when: result|skipped
```

Handlers

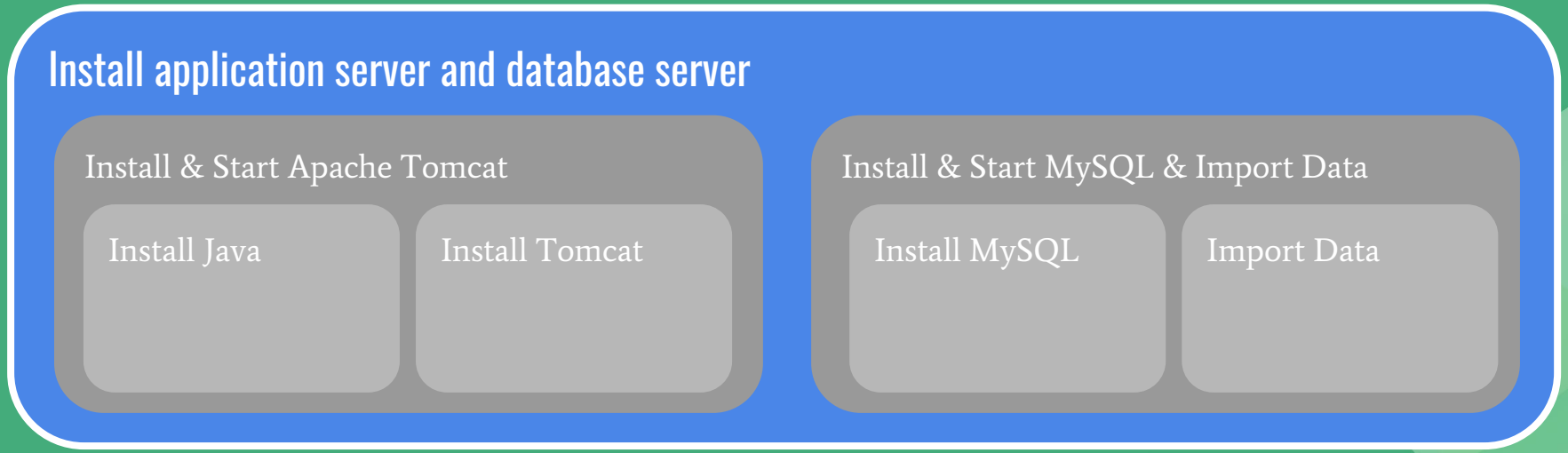
```yaml
tasks:
  - name: Update nginx default config
    copy: src=default.conf dest=/etc/nginx/sites-enabled/default
    notify:
      - Test nginx configuration
      - Reload nginx configuration

handlers:
  - name: Test nginx configuration
    command: nginx -t

  - name: Reload nginx configuration
    command: nginx -s reload
```

Playbooks

...

Playbooks

Playbooks are ordered sets of plays to execute against inventory selections.



Running Playbooks

To run a play book use *ansible-playbook* command.

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

Hosts can be changed by providing a inventory file

```
$ ansible-playbook -i production play.yml
```

Environment variables can be set globally

```
$ ansible-playbook -e "assets_dir=/var/www/html/assets/" play.yml
```

Running Playbooks

Hosts can be limited by providing a subset

```
$ ansible-playbook -i production play.yml
```

Number of parallel processes to use can be specified (default=5)

```
$ ansible-playbook -f 30 play.yml
```

LAB #4

...

Running Playbooks

Install Nginx with a Single Play

Install the nginx server on webservers

```
---
- hosts: webservers
  become: true
  tasks:
  - name: Install nginx
    apt: name=nginx state=present
  - name: Start nginx
    service: name=nginx state=started enabled=yes
```

Install Nginx with a Single Play

```
$ ansible-playbook /vagrant/lab-04/install-nginx.yml -l web1

PLAY [webservers] ***********************************************************

TASK [setup] ****************************************************************
ok: [web1]

TASK [Install nginx] ********************************************************
ok: [web1]

TASK [Start nginx] **********************************************************
ok: [web1]

PLAY RECAP ******************************************************************
web1                       : ok=3    changed=0    unreachable=0    failed=0
```

Install Nginx & JDK & MongoDB in a Playbook

- Install the nginx server on webservers
- Install JDK on appservers
- Install MongoDB on dbservers

Roles

...

Roles

Roles are portable units of task organization in playbooks and is the best way to organize your playbooks.

Roles are just automation around ‘include’ directives, and really don’t contain much additional magic beyond some improvements to search path handling for referenced files.

However, that can be a big thing!

Example Project Structure

```
site.yml
webservers.yml
fooservers.yml
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
   webservers/
     tasks/
```

Example Playbook

```
---
- hosts: webservers
  roles:
     - common
     - webservers
```

Example Role

```
---
- name: Install EPEL repo
  yum: name=epel-release state=present

- name: Install nginx server
  yum: name=nginx enablerepo=epel state=present

- name: Create static content directory
  file: path=/usr/share/nginx/static state=directory
```

LAB #5

...

Running Playbooks with Roles

Install NTP & Nginx with in Roles

Install the ntp service & nginx server on webservers and deploy static content

```
---
- hosts: webservers
  become: true
  roles:
    - ntp
    - nginx
    - deploy_static_content
```

Install Nginx & JDK & MongoDB with in Roles

- Install ntp service on all servers
- Install the nginx server on webservers
- Deploy static content on webservers
- Install JDK on appservers
- Install MongoDB on dbservers

Best Practices

...

Complexity Kills

- Strive for simplification
- Optimize for readability
- Think declaratively

Project Layout

```
├── config.yml
├── provision.yml
├── roles
│   ├── myapp
│   │
│   │
│   │
│   │
│   ├── nginx
│   │   └── etc.etc
│   └── proxy
├── tasks
│   └── main.yml
└── etc.etc
│       └── etc.etc
└── site.yml
```

Meaningful Inventory Names

```
10.1.2.75
10.1.5.45
10.1.4.5
10.1.0.40

w14301.acme.com
w17802.acme.com
W19203.acme.com
w19304.acme.com
```

```
db1 ansible_host=10.1.2.75
db2 ansible_host=10.1.5.45
db3 ansible_host=10.1.4.5
db4 ansible_host=10.1.0.40

web1 ansible_host=w14301.acme.com
web2 ansible_host=w17802.acme.com
web3 ansible_host=w19203.acme.com
web4 ansible_host=w19203.acme.com
```

Vertical Reading is Easier

```
- name: install telegraf
  yum: name=telegraf-{{ telegraf_version
}} state=present update_cache=yes
disable_gpg_check=yes enablerepo=telegraf
```

```
- name: install telegraf
yum: YES
    name: telegraf-{{ telegraf_version }}
    state: present
    update_cache: yes
    disable_gpg_check: yes
    enablerepo: telegraf
```

Meaningful Task Names

```
- hosts: web
   tasks:
    - yum:
        name: httpd
        state: latest
    - service:
        name: httpd
        state: started
        enabled: yes
```

```
- hosts: web
  name: installs and starts apache
  tasks:
    - name: install apache packages
      yum:
        name: httpd
        state: latest
    - name: starts apache service
      service:
        name: httpd
        state: started
        enabled: yes
```

Meaningful Task Names

```
PLAY [web]
*****************************************

TASK [setup]
*****************************************
ok: [web1]

TASK [yum]
*****************************************
ok: [web1]

TASK [service]
*****************************************
ok: [web1]
```

```
PLAY [installs and starts apache]
*****************************************

TASK [setup]
*****************************************
ok: [web1]

TASK [install apache packages]
*****************************************
ok: [web1]

TASK [starts apache service]
*****************************************
ok: [web1]
```

Use Smoke Tests

```
- name: check for proper response
   uri:
     url: http://localhost/myapp
     return_content: yes
   register: result
   until: '"Hello World" in result.content'
   retries: 10
   delay: 1
```

Consider Writing a Module

```
- hosts: all
  vars:
  cert_store: /etc/mycerts
  cert_name: my cert
  tasks:
  - name: check cert
    shell: certify --list --name={{ cert_name }}
    register: output
  - name: create cert
    command: certify --create --user=chris
    when: output.stdout.find(cert_name)" != -1
    register: output
  - name: sign cert
    command: certify --sign
```

```
- hosts: all
  vars:
  cert_store: /etc/mycerts
  cert_name: my cert
  tasks:
  - name: create and sign cert
    certify:
      state: present
      sign: yes
      user: chris
      name: "{{ cert_name }}"
      cert_store: "{{ cert_store }}"
```

LAB #6

...

Spring Boot Rest API Deployment

Deploy Greeting REST Service

Download and build the sample REST service from github and deploy on appservers.

```
$ git clone https://github.com/spring-guides/gs-rest-service.git
$ cd gs-rest-service/complete
$ mvn package
```

* Requires java 8

Tip #5: Installing JDK 8 on Ubuntu 14.04

You should add ‘`ppa:openjdk-r/ppa`’ `repo first:`

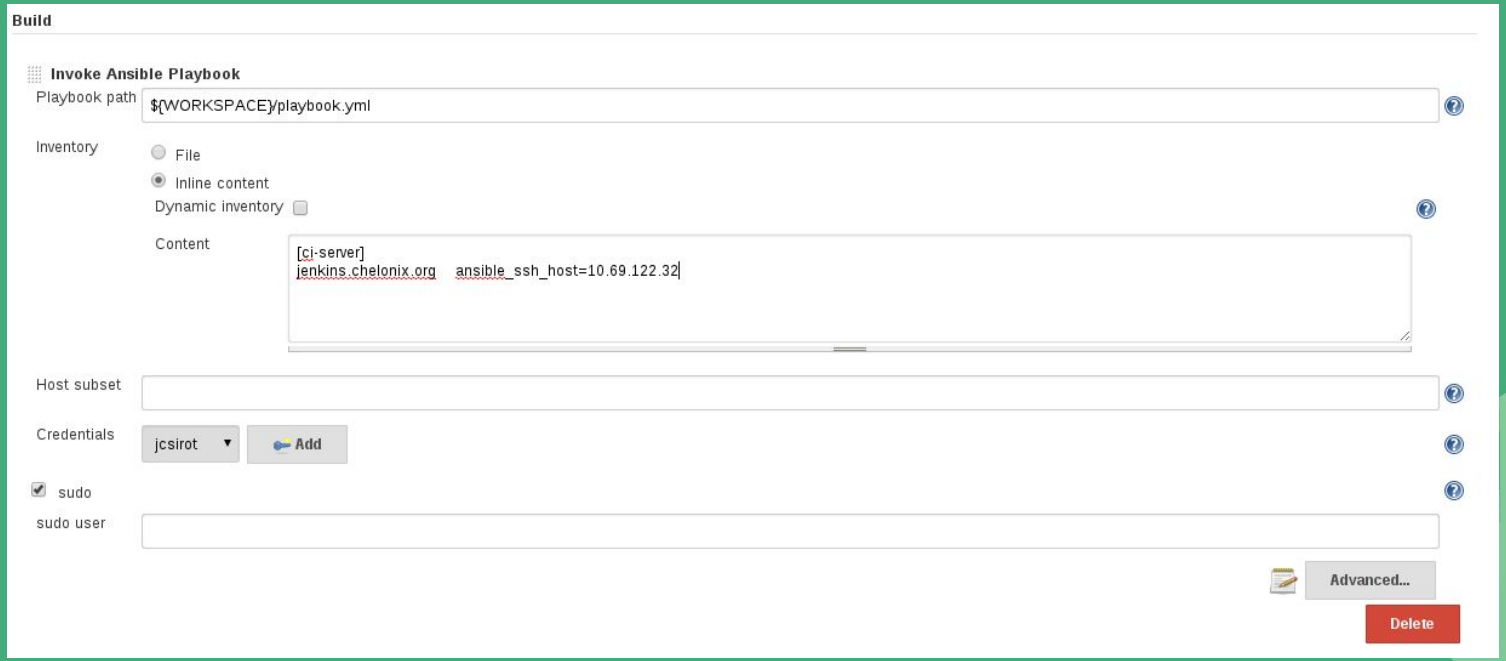
```
---
- name: Install openjdk repository
  apt_repository: repo='ppa:openjdk-r/ppa'
- name: Install openjdk
  apt: name=openjdk-8-jdk state=present
```

Continuous Integration with Ansible

...

Jenkins Ansible Plugin

- https://wiki.jenkins-ci.org/display/JENKINS/Ansible+Plugin



Jenkins Ansible Plugin

Jenkins › jenkins-deploy-ansible › #11

```
TASK: [geerlingguy.jenkins | Define jenkins_repo_key_url] *********************
ok: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Ensure dependencies are installed.] **************
skipping: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Ensure Jenkins repo is installed.] ***************
skipping: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Add Jenkins repo GPG key.] ***********************
skipping: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Ensure Jenkins is installed.] ********************
skipping: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Ensure dependencies are installed.] **************
ok: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Add Jenkins apt repository key.] *****************
ok: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Add Jenkins apt repository.] *********************
ok: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Ensure Jenkins is installed.] ********************
ok: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Ensure Jenkins is started and runs on startup.] ***
ok: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Wait for Jenkins to start up before proceeding.] ***
ok: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Get the jenkins-cli jarfile from the Jenkins server.] ***
changed: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Create Jenkins updates folder.] ******************
ok: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Update Jenkins plugin data.] *********************
skipping: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Permissions for default.json updates info.] ******
changed: [jenkins.chelonix.org]

TASK: [geerlingguy.jenkins | Install Jenkins plugins.] ************************
changed: [jenkins.chelonix.org] => (item=git)
changed: [jenkins.chelonix.org] => (item=sonar)
changed: [jenkins.chelonix.org] => (item=ssh)

NOTIFIED: [geerlingguy.jenkins | restart jenkins] *****************************
changed: [jenkins.chelonix.org]

PLAY RECAP ********************************************************************
jenkins.chelonix.org       : ok=19   changed=4    unreachable=0    failed=0

Finished: SUCCESS
```

Help us localize this page

Page generated: Apr 30, 2015 11:39:50 AM REST API Jenkins ver. 1.611

Jenkins Ansible Plugin

Ansible

Ansible installations

Ansible

Name: ansible 1.9.1

Path to ansible executables directory: /usr/local/bin

Install automatically

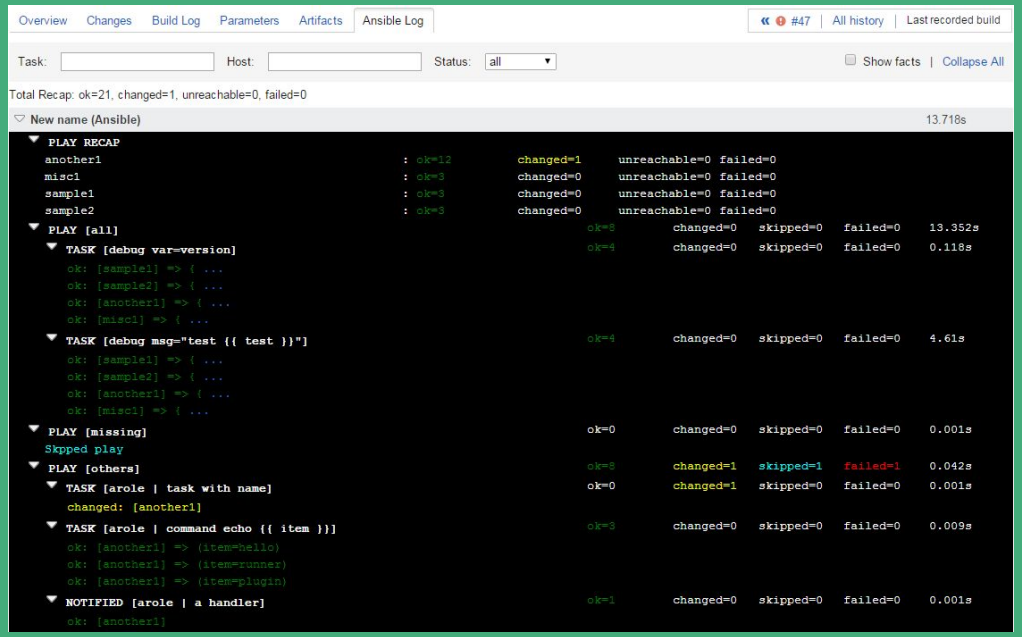
Delete Ansible

Add Ansible

List of Ansible installations on this system

Teamcity Ansible Plugin

- https://github.com/andreizhuk/tc-ansible-runner



Ansible for AWS

...

Boto

Boto is a Python package that provides interfaces to Amazon Web Services. Currently, all features work with Python 2.6 and 2.7. Ansible uses boto to communicate with AWS API.

It can be installed via OS package manager or pip.

```
$ apt-get install python-boto
```

```
$ pip install boto
```

Amazon EC2 Inventory Management

To get started with dynamic inventory management, you'll need to grab the EC2.py script and the EC2.ini config file. The EC2.py script is written using the Boto EC2 library and will query AWS for your running Amazon EC2 instances.

```
$ wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.py
$ wget https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.ini
```

Amazon EC2 Inventory Management

```
$ export AWS_ACCESS_KEY_ID='YOUR_AWS_API_KEY'
$ export AWS_SECRET_ACCESS_KEY='YOUR_AWS_API_SECRET_KEY'
$ export ANSIBLE_HOSTS=/etc/ansible/ec2.py
$ export EC2_INI_PATH=/etc/ansible/ec2.ini

$ ssh-agent bash
$ ssh-add ~/.ssh/keypair.pem
```

Amazon EC2 Inventory Management

```
$ /etc/ansible/ec2.py --list


$ ansible -m ping tag_Ansible_Slave
10.1.2.137 | success >> {
    "changed": false,
    "ping": "pong"
}
10.1.2.136 | success >> {
    "changed": false,
    "ping": "pong"
}
```

Ansible Cloud Modules

From the beginning, Ansible has offered deep support for AWS. Ansible can be used to define, deploy, and manage a wide variety of AWS services. Even the most complicated of AWS environments can be easily described in Ansible playbooks.

- http://docs.ansible.com/ansible/list_of_cloud_modules.html

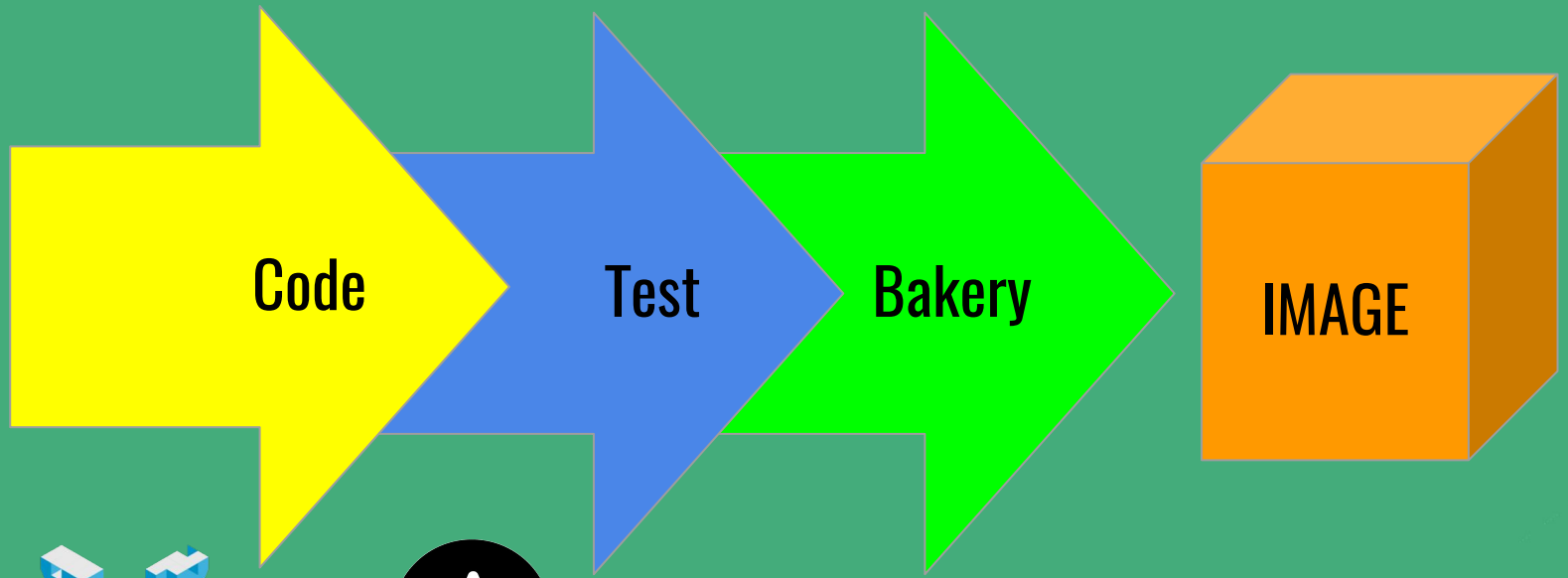
Create Elastic Load Balancer

```
- name: Configure Load Balancer
  ec2_elb_lb:
    name: y-lb-{{suffix}}
    state: present
    region: "{{ec2_region}}"
    connection_draining_timeout: 60
    cross_az_load_balancing: yes
    security_group_ids: "{{lb_security_group.group_id}}"
    subnets: "{{subnet_az_a}},{{subnet_az_b}}"
    listeners:
      - protocol: http
        load_balancer_port: 80
        instance_port: 80
```

Configure Autoscaling Group

```
- name: Configure Autoscaling Group
  ec2_asg:
    name: y_asg_{{suffix}}
    region: "{{ec2_region}}"
    launch_config_name: "{{launch_config.name}}"
    load_balancers: "y-lb-{{suffix}}"
    availability_zones: "{{az_a}},{{az_b}}"
    health_check_period: 60
    health_check_type: ELB
    replace_all_instances: yes
    min_size: "{{min_size}}"
    max_size: "{{max_size}}"
    desired_capacity: "{{desired_capacity}}"
    vpc_zone_identifier: "{{subnet_az_a}},{{subnet_az_b}}"
    wait_timeout: 600
```

Say Goodbye to "Works on my Machine" Bugs



VAGRANT

ANSIBLE

PACKER

LAB #7

...

Deploy Greeting REST Service to AWS

Deploy Greeting REST Service to AWS

For instructions visit https://github.com/maaydin/ansible-tutorial/tree/master/lab-07 .

```
PLAY [localhost] ***************************************************************

TASK [setup] *******************************************************************
ok: [localhost]

TASK [ec2-auto-scale : Configure Launch Configuration Security Group] **********
ok: [localhost]

TASK [ec2-auto-scale : debug] **************************************************
ok: [localhost] => {
    "msg": "Launch Configuration Security Group id=sg-961ee6f0"
}
```

Provisioning Docker Host

...

Installing Docker

To get the latest version of docker it is better (and easier) to install from the script provided by docker.

- https://get.docker.com/

It is also required to install *docker-py* via *pip* to manage your containers from Ansible.

Docker & Ansible

...

Ansible Makes Docker Better

- If you know docker-compose, you know Ansible (almost).
- Because you need to configure the system that your containers are running on.
- Because you want to call out to other systems to configure things.
- Because you want to build testing directly into your container deployment process.

Ansible Docker Modules

- docker_container - manage docker containers
- docker_image - Manage docker images.
- docker_image_facts - Inspect docker images
- docker_login - Log into a Docker registry.
- docker_network - Manage Docker networks
- docker_service - Manage docker services and containers.

Creating a Container

With docker_container module you can manage your docker containers.

```
---
- name: Create a redis container
  docker_container:
    name: myredis
    image: redis
    state: present
```

LAB #8

...

Install Docker & Create a Container

Install Docker on Ansible Controller

```
---
- name: Download Installation Script
  get_url:
    url: https://get.docker.com/
    dest: /tmp/docker.sh

- name: Install docker
  shell: sh /tmp/docker.sh
  args:
    creates: /usr/bin/docker
```

Create a Redis Container

```
---
- name: Create a redis container
  docker_container:
    name: myredis
    image: redis
    command: redis-server --appendonly yes
    state: present
    exposed_ports:
      - 6379
```

Have Fun with

...

ANSIBLE