Deploying Applications With Ansible

Presented by James Meickle
May 30th & 31st

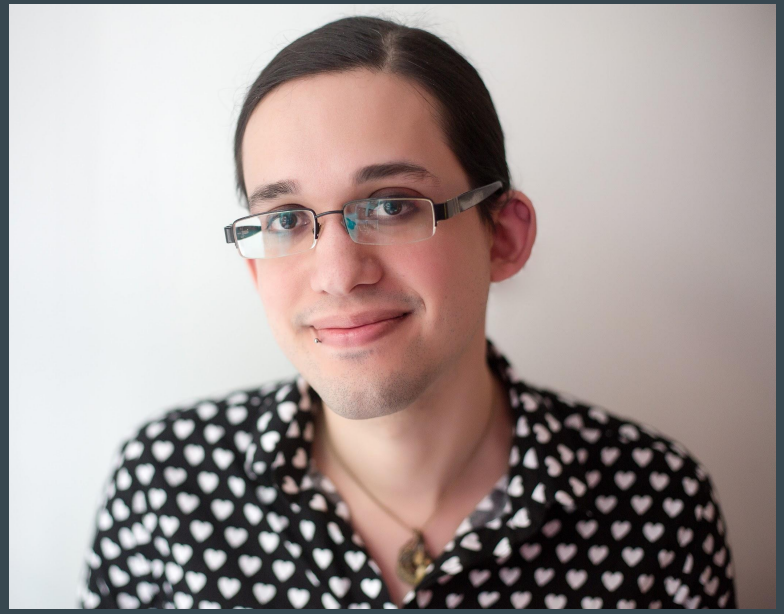
1.1 Introduction & Overview

(5 minute lecture)

- About the instructor
- About the course
- Introduction to Ansible
- Overview of course goals and outline

About the Instructor

- Daily user of Ansible for >2 years
- Site Reliability Engineer at Quantopian
  - Crowd-sourced quantitative finance
  - Infrastructure and operations work
  - Python, Ansible, and Vault in AWS
- Organizer at DevOpsDays Boston
  - Talk selection
- Formerly:
  - Site Reliability Engineer, Harvard
  - Developer Evangelist, AppNeta
  - Release Engineer, Romney for President



Email: eronarn@gmail.com
LinkedIn: https://www.linkedin.com/in/eronarn/
Twitter: https://twitter.com/jmeickle
Web: https://permadeath.com

About the Course

- Three hour sessions on two adjacent days
- Recommended reading for each day
- Mixture of lectures, demos, and hands-on exercises in AWS
- **Slides are available online!**
  - **Don't try to read them all now, they're text-heavy and include links so you can come back later.**
- Follow along in Slack:
  - https://oreillyonlinetraining.slack.com
  - #apps-w-ansible-05-30

Deploying Applications With Ansible

Ansible is a "batteries included" automation, configuration management, and orchestration tool that's fast to learn and flexible enough for any architecture. It's the perfect choice for automating repetitive tasks, increasing deployment frequency, and running infrastructure at scale. Companies are increasingly turning to Ansible to meet the challenges of transitioning from the data center to the cloud, from servers to containers, and from silos to devops.

In this course you'll learn Ansible's key concepts and how to develop workflows that solve your challenges. The curriculum focuses on practical techniques, with an eye towards reusable and maintainable code. On completion you'll have enough hands-on experience with Ansible to feel comfortable using it in your own environment on a daily basis.

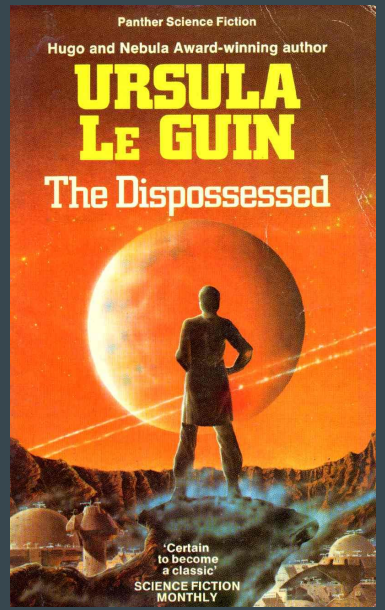
What Is Ansible?

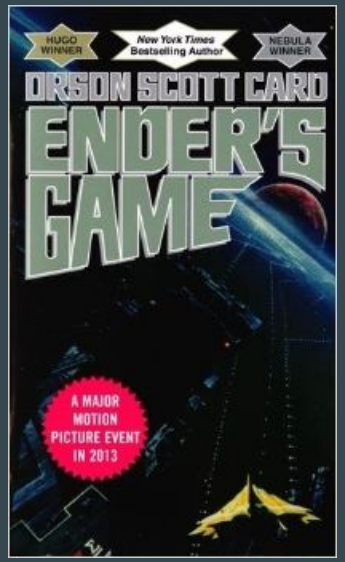
- Software library, tools, and ecosystem for:
  - Automation
  - Configuration management
  - Orchestration
- Open source
- Highly extensible (roles, modules, plugins)

Where Did Ansible Come From?

“It win be a device that will permit communication without any time interval between two points in space. The device will not transmit messages, of course; simultaneity is identity. But to our perceptions, that simultaneity will function as a transmission, a sending. So we will be able to use it to talk between worlds, without the long waiting for the message to go and the reply to return that electromagnetic impulses require. It is really a very simple matter. Like a kind of telephone.”

The Dispossessed, Ursula K. Le Guin





Where Did Ansible Come From?

- Created by Michael DeHaan (February 2012)
  - Originally AnsibleWorks; later Ansible, Inc.
- Acquired by Red Hat around v1.9.4 (October 2015)
- Major internal rework in v2.0 (January 2016)
- Recently released v2.3 (April 2017)
- Recent focus on enterprise features:
  - Windows
  - Network devices
  - Containers
  - Ansible Tower

What Do People Use Ansible For?

Docker, git, S3, Django, Azure, Cisco, Datadog, Homebrew, apt, Pingdom, SELinux, Composer, OpenStack, pip, Pagerduty, yum, Mercurial, CloudFormation, dnf, Vmware, Puppet, DigitalOcean, cron, Windows, New Relic, rabbitMQ, svn, CloudStack, EC2, iptables, Rackspace, npm, Sensu...

What Do I Use Ansible For?

- Automating deploys via Ansible Tower
- Building AMIs
- Provisioning Vagrant VMs
- One-off orchestration tasks

- Managing a lot of “pets” (no cattle)
- Additional configuration within someone else’s Puppet environment
- Ansible-based CI system
  - Deployed with Ansible!
- Building and managing legacy web apps
- Building Docker containers without Dockerfiles

Course Goals

During this course, you'll learn...

- Why companies are adopting Ansible over other tools
- Key Ansible concepts: variables, modules, playbooks, roles, and more
- Best practices for writing reusable, maintainable code in Ansible

After the course, you'll be able to...

- Write Ansible roles and playbooks to automate routine operations tasks
- Orchestrate deployments of multi-tier applications
- Manage your infrastructure as code, whether in the cloud or the data center
- Drive adoption of Ansible within your team through incremental adoption

Course Outline

Day One:

- Cover course goals
- Introduce Ansible and its history
- Connect to training environment
- Ansible concepts:
  - Modules
  - Playbooks
  - Roles
- Practice writing playbooks and roles
- Deploy a single-node web application
- Compare Ansible to other tools

Day Two:

- Advanced module and role tips
- Explore cloud-focused features
- Deploy a multi-tier, multi-instance application in AWS
- Discuss code quality for Ansible
- Develop a custom Ansible module
- Learn about the Ansible ecosystem
- Recap what we've learned and revisit course goals

1.2
Installing Ansible

(5 minute lecture/exercise)

- Learn how Ansible is installed, and try to do it locally
- Run your first ad hoc (non-playbook) Ansible command

Installing Ansible locally is not required for this training!

Python

- Ansible is (mostly) written in Python
  - The control machine (your computer) requires Python 2.6+
- Most Ansible functionality is Python code executed remotely
  - The managed node (the server you connect to) requires Python 2.6+
- No unusual dependencies
  - Can be pip installed like other Python packages
- Works out of the box almost anywhere!
  - Except on Windows...

Information current as of Ansible v2.3.0.0 - remember to always check your version and read the matching documentation!

Environment Conflicts

- “Infrastructure as code” implies consistency and reproducibility
- Like any other code, Ansible’s behavior is affected by runtime environment
- Environment changes will frequently change behavior:
  - Variables loading differently or failing to load
  - Playbooks suddenly not finding or connecting to hosts
  - Includes or file references no longer found
- Ansible development tends to be “move fast and break things”
  - Occasionally, breaking changes don’t make it into the release notes

Common Sources of Environment Conflicts

Affects the control machine:

- Ansible config file
  - Don't use global Ansible config files
  - Commit a config file per repo
- Ansible version
  - Repo-level: pip requirements.txt
  - Playbook-level: assert on Ansible version

Watch out! Ansible roles allow you to specify a "minimum Ansible version", but it isn't checked at runtime.

Affects both control and managed nodes:

- Python 2 vs. Python 3
- Python minor version
- Python module versions
  - pip freeze > requirements.txt
- OpenSSH version
- SSH configuration options
  - Especially ~/.ssh/config

Maintainable Ansible Installations

Don't try these at home:

- `yum install ansible`
- `apt-get install ansible`
  - `apt-add-repository ppa:ansible/ansible`
- `brew install ansible`
- `sudo pip install ansible`

These commands **will** install Ansible, but at the cost of...

- Less control over version
- Less recent versions
- Difficulty managing multiple versions
- Difficulty reproducing your Ansible installation on on other systems

Maintainable Ansible code starts with maintainable Ansible installations!

Installing Ansible on Linux

```
# Install pyenv and pyenv-virtualenv:
# https://github.com/pyenv/pyenv-installer

pyenv install 2.7.10
pyenv global 2.7.10
pyenv virtualenv ansible
pyenv activate ansible

# Even better: specify an ansible version in
# a requirements.txt file in the repo!
pip install ansible==2.3.0.0
ansible --version
```

pyenv installs multiple versions of Python side by side without touching your system Python

virtualenv creates isolated Python package environments with known versions of Python as well as independent, non-root pip installs

pyenv-virtualenv uses pyenv to manage virtualenvs stored in your home directory and loaded by name from anywhere

Installing Ansible on OSX

Install pyenv and pyenv-virtualenv:
https://github.com/pyenv/pyenv#homebrew-on-mac-os-x
https://github.com/pyenv/pyenv-virtualenv#installing-with-homebrew-for-os-x-users

pyenv install 2.7.10
pyenv global 2.7.10
pyenv virtualenv ansible
pyenv activate ansible

Even better: specify an ansible version in
a requirements.txt file in the repo!
pip install ansible==2.3.0.0
ansible --version

pyenv installs multiple versions of Python side by side without touching your system Python

virtualenv creates isolated Python package environments with known versions of Python as well as independent, non-root pip installs

pyenv-virtualenv uses pyenv to manage virtualenvs stored in your home directory and loaded by name from anywhere

Installing Ansible on Windows

- You can use Ansible to manage Windows hosts: http://docs.ansible.com/ansible/list_of_windows_modules.html
  - Shell commands
  - Windows Update
  - IIS
- *Windows is **NOT** fully supported as a control machine.*
  - There is experimental support here: http://docs.ansible.com/ansible/intro_windows.html
- **If you are running Windows, consider installing Ansible on a Linux VM.**

ansible

- 'ansible' is the most basic Ansible command
- Runs an Ansible 'module' as an isolated command
  - Restart a service on every server
  - Get current disk use
  - Count active SSH sessions
- Can run on many nodes in parallel
- Installed locally? Try it out:
  - ansible -m ping localhost
  - ansible -m setup localhost
  - ansible -a "df -h" localhost

```
(ansible) vagrant@vagrant:~/ansible$ ansible -m ping localhost
localhost | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
(ansible) vagrant@vagrant:~/ansible$ ansible -m setup localhost
localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.0.2.15",
            "192.168.33.10"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::a00:27ff:fee8:89c6",
            "fe80::a00:27ff:fe51:4661"
        ],
        "ansible_apparmor": {
            "status": "enabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_version": "VirtualBox",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/vmlinuz-4.4.0-75-generic",
            "quiet": true,
            "ro": true,
            "root": "/dev/mapper/vagrant--vg-root"
        },
        "ansible_date_time": {
            "date": "2017-05-14",
            "day": "14",
            "epoch": "1494781895",
            "hour": "17",
            "iso8601": "2017-05-14T17:11:35Z",
            "iso8601_basic": "20170514T171135756412",
            "iso8601_basic_short": "20170514T171135",
```

1.3 Your Ansible Deployment

10 minute lecture
5 minute exercise

- Introduce Ansible modules
- Describe how Ansible discovers and connects to servers
- Describe the AWS architecture for this course
- SSH into your EC2 instance and try running modules

Modules

- Extend Ansible to perform new tasks
- “Batteries included” - ships with a huge number of common OS tasks, plus many third party modules
- Core Ansible modules written in Python
  - We’ll talk about other languages later
- Modules typically generate a script, push it to the remote node, and execute it

The “-m” in ‘ansible -m ping all’ stands for ‘module’. Any module can be run this way, but it’s only recommended for the simplest modules.

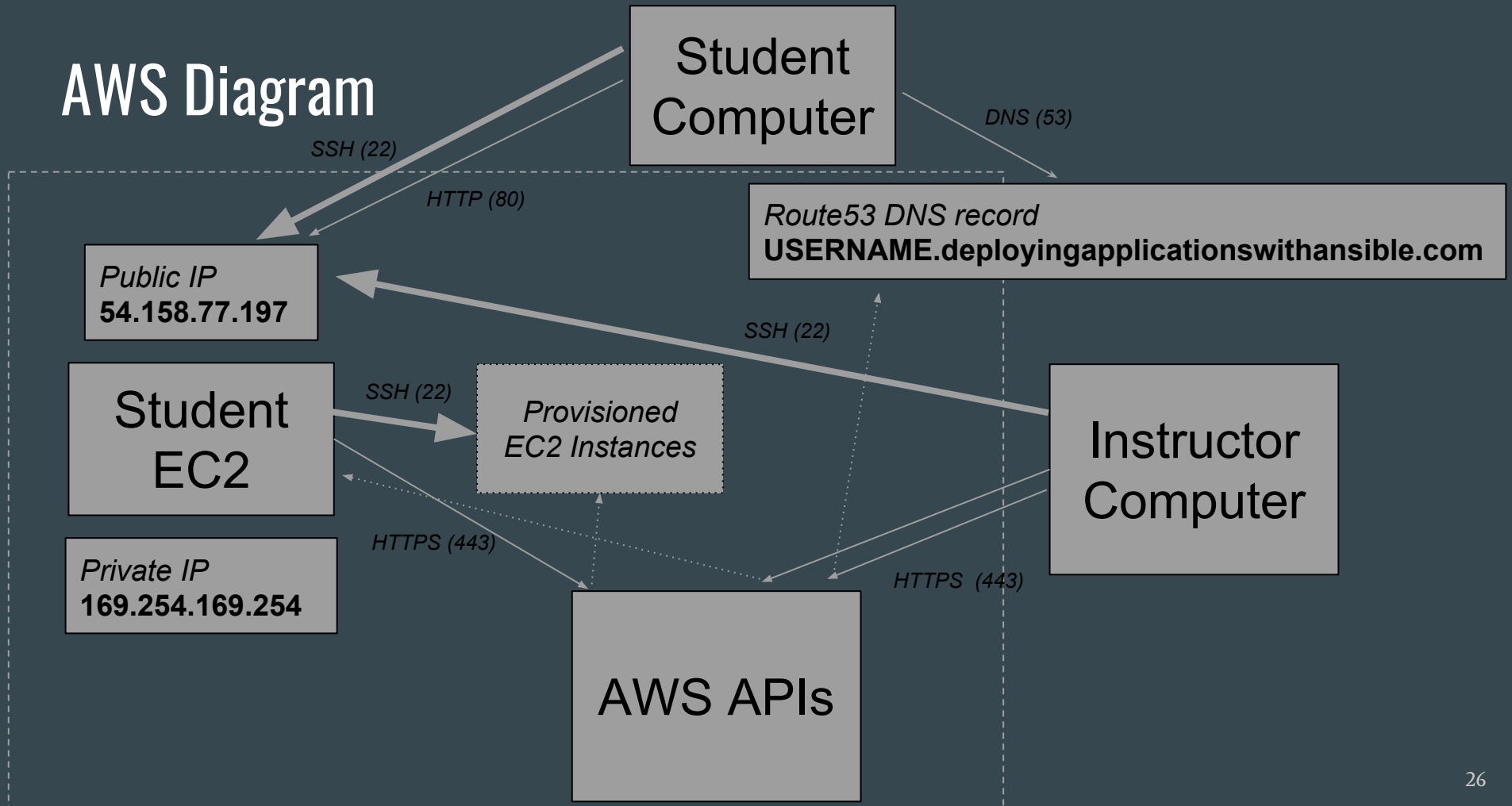
Amazon

- aws_kms - Perform various KMS management tasks.
- cloudformation - Create or delete an AWS CloudFormation stack
- cloudformation_facts - Obtain facts about an AWS CloudFormation stack
- cloudfront_facts - Obtain facts about an AWS CloudFront distribution
- cloudtrail - manage CloudTrail create, delete, update
- cloudwatchevent_rule - Manage CloudWatch Event rules and targets
- dynamodb_table - Create, update or delete AWS Dynamo DB tables.
- ec2 - create, terminate, start or stop an instance in ec2
- ec2_ami - create or destroy an image in ec2
- ec2_ami_copy - copies AMI between AWS regions, return new image id
- ec2_ami_find - Searches for AMIs to obtain the AMI ID and other information
- ec2_ami_search (D) - Retrieve AWS AMI information for a given operating system.
- ec2_asg - Create or delete AWS Autoscaling Groups
- ec2_asg_facts - Gather facts about ec2 Auto Scaling Groups (ASGs) in AWS
- ec2_customer_gateway - Manage an AWS customer gateway
- ec2_eip - manages EC2 elastic IP (EIP) addresses.
- ec2_elb - De-registers or registers instances from EC2 ELBs
- ec2_elb_facts - Gather facts about EC2 Elastic Load Balancers in AWS
- ec2_elb_lb - Creates or destroys Amazon ELB.
- ec2_eni - Create and optionally attach an Elastic Network Interface (ENI) to an instance
- ec2_eni_facts - Gather facts about ec2 ENI interfaces in AWS
- ec2_facts - Gathers facts about remote hosts within ec2 (aws)

SSH

- Ansible's default "transport" is SSH (sftp or scp)
- Uses your user's ~/.ssh/config AND Ansible-specific configuration overrides
  - Pro: Supports any topology (e.g. bastion hosts) the same way you would SSH into them
  - Con: This is a common source of inconsistency between users executing playbooks locally!
- SSH keys are strongly recommended
- OpenSSH is strongly recommended
  - Use a recent OpenSSH!
  - Paramiko (limited-functionality Python SSH client) is available otherwise
- ControlPersist and connection pipelining are strongly recommended
  - ControlPersist is enabled by default, but can catastrophically fail and block plays
  - Pipelining is **disabled** by default

AWS Diagram


Student Computer
SSH (22)
HTTP (80)
DNS (53)
Route53 DNS record
USERNAME.deployingapplicationswithansible.com
Public IP
54.158.77.197
SSH (22)
Student EC2
SSH (22)
Provisioned EC2 Instances
Instructor Computer
Private IP
169.254.169.254
HTTPS (443)
HTTPS (443)
AWS APIs

Your Ansible Deployment

- ~/ansible
  - git checkout of the repository linked at the bottom of these slides
- ~/ansible/README.md
  - Course information and repository instructions
- ~/ansible/ansible.cfg
  - Configuration file used when your current directory is ~/ansible
- ~/ansible/environments/aws
  - Inventory directory selected via ~/ansible/ansible.cfg
- ~/ansible/playbooks
  - Each YAML file is a playbook
- ~/ansible/roles
  - Each folder is a role
- ~/ansible/roles/requirements.yml
  - For installing roles with ansible-galaxy
- ~/ansible/files
  - Static files used in playbooks
- ~/ansible/templates
  - Jinja template files used in playbooks
- ~/.setup_pyenv
  - Script to enable pyenv

Exercise:

Retrieve your SSH username and password, and SSH to:

USERNAME@USERNAME.deployingapplicationswithansible.com

Logged in? 'cd ~/ansible' and check out 'ONELINERS.md'!

5 Minute Q&A/Break

1.4
Ansible Playbooks: Concepts

5 minute lecture
5 minute demo

- Cover YAML and Jinja syntax
- Introduce tasks, plays, and playbooks
- Demonstrate higher complexity modules

YAML

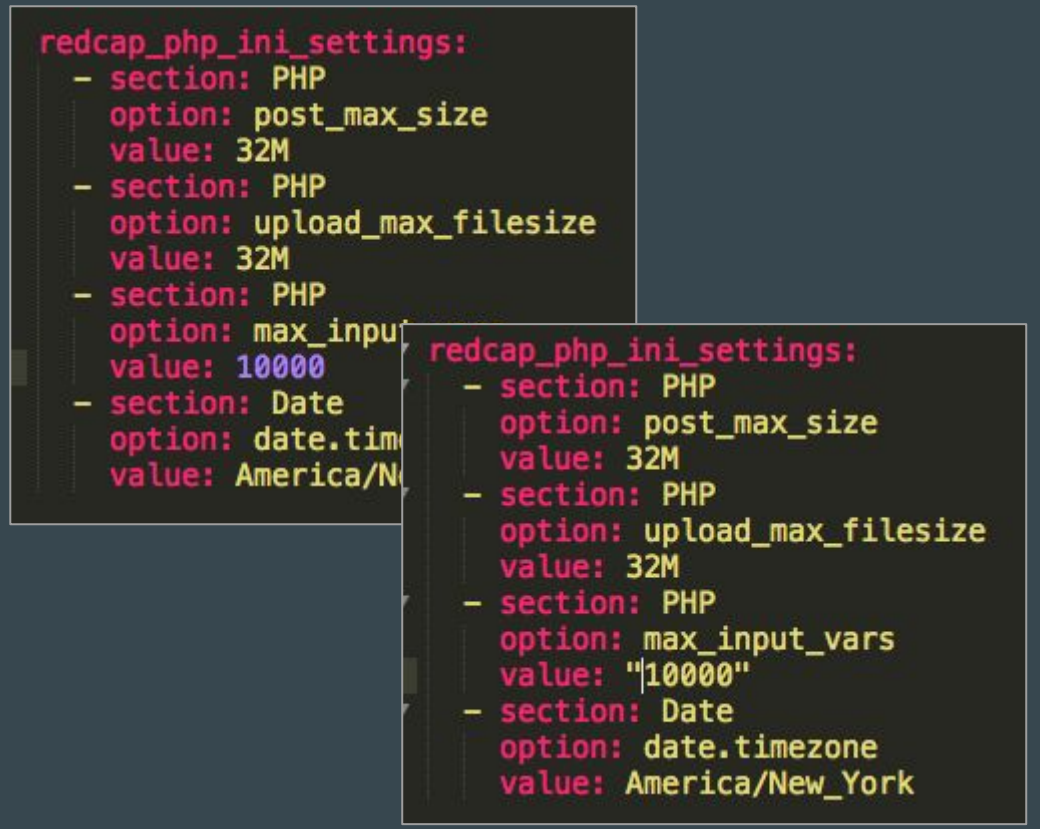
- Markup language for describing data
- Used for both Ansible playbooks ("code") and variable files ("data")
- YAML is a superset of JSON
  - Supports comments!
- Data types:
  - int: 123
  - float: 123.0
  - string: "123"
  - bool: y|Y|yes|Yes|YES|n|N|no|No|NO|true|True|TRUE|false|False|FALSE|on|On|ON|off|Off|OFF
  - lists: ['foo', 'bar'] or indent + '-'
  - dicts/hashes: {'foo': 'bar'} or indent + 'foo:'

```
logstash_forwarder_logstash_server_port: 5000
logstash_forwarder_ssl_src_dir: '{{inventory_dir}}/files/ssl'
logstash_forwarder_ssl_certificate_file: logstash.crt

logstash_forwarder_files:
  # CentOS syslog paths
  - paths:
      # Syslog
      - /var/log/messages
      # User logins
      - /var/log/secure
    fields:
      type: syslog
```

YAML Typing

- YAML is aggressive about type inference, so use syntax highlighting!
  - "{{ foo }}" - templated variable
  - { foo } - set containing a string
  - {{ foo }} - nested set containing a string
- Quote values that confuse the parser:
  - [] = empty list, “[]” = two square brackets
  - “” = empty string, ““” = two double quotes
  - {test} = invalid dict, “{test}” = string
- Or cast values with Jinja filters:
  - {{1|float}}
  - {{“10000”|int}}
  - {{character_string|list}}
  - {{ some_variable | from_json }}



Jinja2

- Python templating language
- Used for Ansible playbook logic and variable definition, as well as to template
  - Jinja “inline” in Ansible playbooks is much more limited in functionality
- Supports:
  - Tests
  - Filters
  - Loops
  - Nested templates/inheritance
  - Extension with Python code

```
include /etc/nginx/conf.d/*.{{item.name}}.upstream;

server {
    listen      80;
    server_name {{item.host}};
    rewrite     ^   https://$server_name$request_uri? permanent;
}

server {
  listen                443;

  {% if item.ssl_cert is defined or item.ssl_key is defined %}
  ssl    on;
  ssl_certificate       {{nginx_ssl_dest}}/{{item.ssl_cert}};
  ssl_certificate_key   {{nginx_ssl_dest}}/{{item.ssl_key}};
  {% endif %}

  server_name           {{item.host}};
  access_log            {{item.log}};

  include /etc/nginx/conf.d/*.{{item.name}}.location;
}
```

Tasks

- YAML dictionary containing a module, execution arguments, control flow statements, and metadata
- Stored within a strictly linear list of tasks
- Shares an execution context with other tasks:
  - Tags to select/exclude tasks
  - Register task output as a variable
  - Conditional execution based on variables
- Task lists can be included but never executed directly

```
# System level installation of buildb

# We install GCC to compile Twisted b
- name: Install buildbot dependencies
  tags: buildbot
  yum: name={{ item }} state=present
  sudo: yes
  with_items:
    - python-devel
    - python-pip
    -
    -
    -

- name
  tags
  pip:
  sudo
```

```
- name: Install nginx dependencies
  sudo: yes
  yum: name={{item}}
  with_items:
    - pam-devel
    - pcre
    - pcre-devel

- name: Configure nginx makefile
  shell: >
    cd {{nginx_workdir}}/nginx-{{nginx_version}}
    && ./configure
    --prefix=/usr/share/nginx
    --sbin-path=/usr/sbin/nginx
    --conf-path=/etc/nginx/nginx.conf
```

Ansible Execution Order

Tasks

A YAML list of tasks with no additional metadata

Task

A YAML dictionary of metadata and execution arguments wrapping a module

Plays

- If a task is an argument and metadata container for a module, then a play is a similar wrapper around a list of tasks
- Execution order is still strictly linear, but includes multiple stages:
  - vars_files
  - vars_prompts
  - pre_tasks
  - roles
  - tasks
  - post_tasks
  - (plus handlers!)
- Plays are still never executed directly

```
# PLAY 1: Set up Cachet
- hosts: cachet
  tags: [cachet, docker]
  tasks:
    - name: Run the Cachet app container
      docker_container:
        name: '{{cachet_app_docker_name}}'
        image: '{{cachet_app_docker_image}}:{{cachet_app
        command: /var/www/html/entrypoint.sh
        state: started
        expose: 9000
        memory_swappiness: 0
        ports:
          - '{{cachet_app_http_port}}:9000'
        env:
          APP_DEBUG: true
          APP_URL: '{{cachet_app_url}}'
          DB_HOST: '{{cachet_db_host}}'
          DB_DATABASE: '{{cachet_db_database}}'
          DB_DRIVER: '{{cachet_db_type}}'
          DB_USERNAME: '{{cachet_db_username}}'
          DB_PASSWORD: '{{cachet_db_password}}'
          MAIL_HOST: '{{cachet_smtp_host}}'
          MAIL_PORT: '{{cachet_smtp_port}}'
          MAIL_ADDRESS: '{{cachet_smtp_address}}'
          MAIL_NAME: '{{cachet_smtp_name}}'
          MAIL_ENCRYPTION: '{{cachet_smtp_encryption}}'
          QUEUE_DRIVER: '{{cachet_queue_driver}}'
      register: cachet_app_container

    # WHY DOES THIS KEEP FAILING ON THE FIRST RUN AAAA
    - name: Run database migrations
```

Ansible Execution Order

Play — A YAML dictionary containing metadata, connection info, and roles/tasks

Tasks — A YAML list of tasks with no additional metadata

Task — A YAML dictionary of metadata and execution arguments wrapping a module

Playbooks

- A playbook is a file containing a YAML list of plays
  - ...but most playbooks are just one play!
- No additional arguments or metadata, but shared runtime scope
- Execution order is still strictly linear, forming a nested loop
- Playbooks can include each other:
  - ansible-playbook my_website.yml
    - include: provision_servers.yml
    - include: install_web_tier.yml
    - include: Install_database.yml
- ansible-playbook command runs playbooks, not plays or tasks or roles.

Ansible Execution Order

Playbook — A file that contains a list of plays

Play — A YAML dictionary containing metadata, connection info, and roles/tasks

Tasks — A YAML list of tasks with no additional metadata

Task — A YAML dictionary of metadata and execution arguments wrapping a module

Demo:

ansible-playbook playbooks/1.4_cron.yml

1.5
Ansible Playbooks: Structure

15 minute lecture

- Cover key play and playbook concepts:
  - Hosts
  - Variables
  - Tasks
  - Handlers
- Multiple plays and variable scoping

Play Hosts

- 'hosts': is the host, or groups of hosts, that the play is executed across
- 'remote_user:' changes the user (default: your current user) used to *connect to* each host executing the play
- ‘become: yes’ and 'become_user:' (default: root) switch to a different user to *run commands on* each host executing the play
  - This used to be 'sudo', but that syntax is now deprecated.

```
- name: apply common configuration to all nodes
  hosts: all
  remote_user: root

  roles:
    - common

- name: configure and deploy the webservers and application code
  hosts: webservers
  remote_user: root

  roles:
    - web

- name: deploy MySQL and configure the databases
  hosts: dbservers
  remote_user: root

  roles:
    - db
```

Play Variables

- Scoped to last through a play
- Three common sources:
  - Provided in playbook directly via ‘vars:’
  - Collected from user at runtime via ‘vars_prompt:’
  - Included from YAML files at runtime via ‘vars_files:’

ubuntu-dev.yml

```yaml
# Override package manager
package_manager: apt
```

```yaml
- hosts: control_machine
  become: yes

  vars:
    # Default to a dev environment
    env: dev
    # Default to yum as a package manager
    package_manager: yum

  vars_prompt:
    name: distro
    prompt: Choose which distro to install
    default: ubuntu

  vars_files:
    - "../vars/{{distro}}-{{env}}.yml"

  tasks:
    - apt:
        name: git
      when: package_manager == 'apt'

    - yum:
        name: git
      when: package_manager == 'yum'
```

Play Tasks

- Each play has three task lists:
  - pre_tasks (before roles)
  - tasks (after roles*)
  - post_tasks (after roles)
- Task lists can include YAML files of tasks
  - Not YAML playbooks, though!
- Tasks support 'become:' syntax, which will override the play-level 'become:'

- Change task outputs:
  - 'register:'
  - 'changed_when'
  - 'failed_when:'
  - 'ignore_errors:'
- Change execution flow:
  - 'when:'
  - 'with_*:',
  - notify
  - tags

Ansible Execution Order

Playbook — A file that contains a list of plays

Play — A YAML dictionary containing metadata, connection info, and roles/tasks

Tasks — A YAML list of tasks with no additional metadata

Task — A YAML dictionary of metadata and execution arguments wrapping a module

Play Handlers

- Defined in 'handlers:' list of a play
- If a 'changed' task has a 'notify:' argument, it will notify any associated handlers
  - Changed config file -> restart service
  - Installed package -> recompile code
- Notified handlers run...
  - At the end of the current section
  - In the order they were defined
    - *Not the order they were notified!*
  - Only on the hosts that notified them
  - Only once, regardless of how many times they were notified

```
- name: write the apache config file
  template: src=/srv/httpd.j2 dest=/etc/httpd.conf
  notify:
  - restart apache

handlers:
  - name: restart apache
    service: name=httpd state=restarted
```

Multiple Plays

- Playbooks can contain more than one play
  - Or include all plays from another playbook.
- All hosts will finish executing a play before the next play starts.
- Beware of different variable scopes when using multiple plays!
  - Global: config, environment variables and the command line
  - Play: each play and contained structures, vars entries (vars; vars_files; vars_prompt), role defaults and vars. **These variables are 'global' but don't persist across plays.**
  - Host: variables directly associated to a host, like inventory, include_vars, facts or registered task outputs. **Persists on a per-host basis across any plays the host is included in.**

1.6
Ansible Playbooks: Hands-On

15 minute exercise

- Point out the pre-provided playbooks repository on the instance
- Extend a provided demo playbook with additional tasks to deploy a basic web application

Exercise:

Fill in incomplete tasks and get this playbook to complete:

ansible-playbook playbooks/1.6_webapp.yml

5 Minute Q&A/Break

1.7
Ansible Roles: Concepts

15 minute lecture

- Introduce roles, Ansible's primary unit of reusable functionality
- Show a more typical, role-heavy playbook

Introduction to Roles

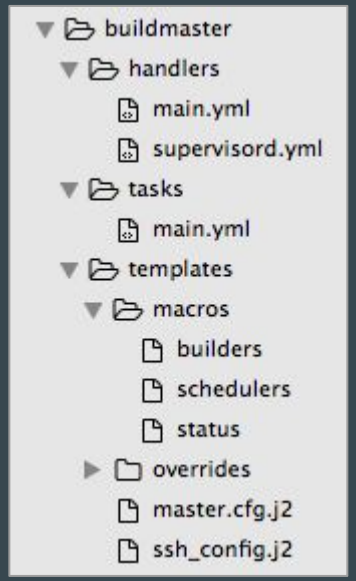
- Roles are packages of reusable Ansible tasks that promote code reuse and composability
  - Roles use ansible-galaxy as a package manager, and it's not a good one
- They can be:
  - Ansible Galaxy public roles
  - Forks of public roles
  - Custom "in-house" roles
- Roles are never executed directly; they are only executed during plays

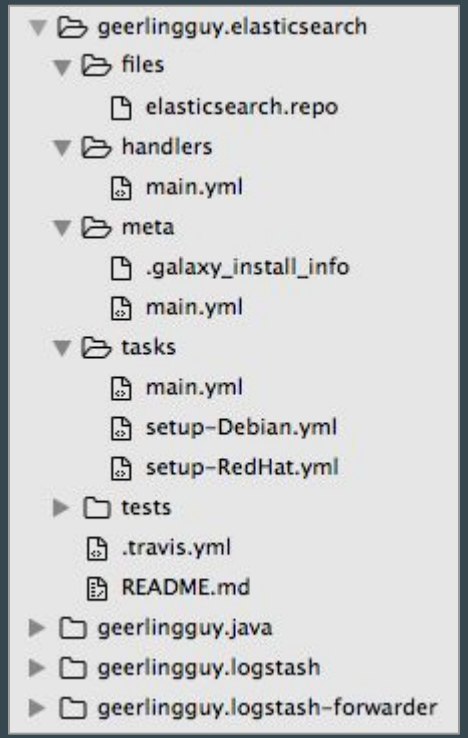
Roles can bundle:

- Tasks
- Files and templates
- Handlers
- Variables
- Modules (rare)

Role Structure

- Typically, each role is its own repository
  - Git repos can be installed directly as roles via *ansible-galaxy*
- Well defined directory structure that can be generated automatically with *ansible-galaxy*
- Roles package Ansible code along with:
  - README.md
  - meta/main.yml (for Ansible Galaxy)
  - tests





Ansible Execution Order

- **Playbook** — A file that contains a list of plays
  - **Play** — A YAML dictionary containing metadata, connection info, and roles/tasks
    - **Tasks** — A YAML list of tasks with no additional metadata
      - **Task** — A YAML dictionary of metadata and execution arguments wrapping a module

Role Tasks

- Stored in "tasks" folder
- Defaults to executing "main.yml"
- Common pattern:
  - main.yml only has include statements for other .task ymls, and logic for when to include them
  - Other .ymls are for clearly defined steps: compile, install, configure, etc.
  - Environment specific includes are split out with conditionals: compile-Debian.yml vs. compile-RedHat.yml
  - Include statements in main.yml are tagged to permit running or excluding steps

```
- include: dependencies.yml
  tags: ['dependencies', 'internet']

- include: install.yml
  tags: ['install']

- include: run-aws.yml
  tags: ['aws']

- include: run-gcp.yml
  tags: ['gcp']

- include: run-vagrant.yml
  tags: ['vagrant']
```

Roles in Playbooks

- Roles are executed after 'tasks:'
- Plays can execute the same role multiple times
- Roles have access to variables in the current play's and the global scope
- Plays can pass named variables and/or tags into a role

```
- hosts: xnat_proxies
  remote_user: "{{root_user}}"
  environment: '{{default_env}}'

  pre_tasks: []

  roles:
    # Install nginx with SSL and PAM
    - {role: nginx, tags: nginx}
    - role: nginx_ssl
      tags: [nginx, nginx_ssl]
      nginx_servers: '{{xnat_proxy_servers}}'
    - {role: nginx_pam, tags: [nginx, nginx_pam]}
    - {role: xnat_proxy, tags: xnat_proxy}
    # - {role: logstash-forwarder, tags: logstash-forwarder}

  tasks: []

- hosts: xnat_servers
  remote_user: "{{root_user}}"
  environment: '{{default_env}}'

  roles:
    # - {role: common, tags: common}
    - {role: xnat_app, tags: xnat_app}
    - {role: rsnapshot, tags: rsnapshot}
    # - {role: logstash-forwarder, tags: logstash-forwarder}
```

Ansible Execution Order

- **Playbook** — A file that contains a list of plays
  - **Play** — A YAML dictionary containing metadata, connection info, and roles/tasks
    - **Role** — A folder containing tasks, handlers, metadata, variables, etc.
      - **Tasks** — A YAML list of tasks with no additional metadata
        - **Task** — A YAML dictionary of metadata and execution arguments wrapping a module
      - **Tasks** — A YAML list of tasks with no additional metadata
        - **Task** — A YAML dictionary of metadata and execution arguments wrapping a module

Role Files and Templates

- Roles often ship with configuration files of the templates that they manage
- 'template' tasks will first look for files in 'templates':
  - src: config.ini.j2
- Other tasks that load files will first look for files in 'files':
  - src: python_script.py
- Rarely, you may need to access files via relative paths, starting from the playbook:
  - src: ../roles/a_different_role/files/my_file

Ansible documentation:

Any copy, script, template or include tasks (in the role) can reference files in roles/x/{files,templates,tasks}/ (dir depends on task) without having to path them relatively or absolutely

Role Variables

- Role “defaults” (defaults/main.yml or defaults/vars.yml) are always included, but are the lowest precedence of all variables
- Roles often also include a *vars* folder that can be explicitly included at runtime, often based on a condition:
  - Environment (Staging, Production)
  - OS
  - Cloud provider
- **Make sure to use prefixes, because there is no namespacing for Ansible variables**

```
logstash_forwarder_ssl_src_dir: '{{inventory_dir}}/files/ssl
logstash_forwarder_ssl_certificate_file: logstash.crt

logstash_forwarder_files:
  # CentOS syslog paths
  - paths:
      # Syslog
      - /var/log/messages
      # User logins
      - /var/log/secure
    fields:
      type: syslog
```

```
nginx_ssl_key: docker.key
nginx_ssl_cert: docker.crt

nginx_servers:
  - name: xnat_proxy_docker
    host: nrg-buildmaster.rc.fas.harvard.edu
    log: /var/log/nginx/access.log timed_combined
    ssl_key: xnat_docker.key
    ssl_cert: xnat_docker.crt

xnat_proxy_servers:
  - name: docker
    host: nrg-buildmaster.rc.fas.harvard.edu
    port: XNAT_DOCKER_HTTP_PORT
    ssl_port: XNAT_DOCKER_HTTPS_PORT
    ssl_cert: xnat_docker.crt
```

Role Handlers

- Same functionality as play handlers
- Role handlers become available to all tasks in the play, but only *after* the role runs
- Well designed roles support this pattern for cooperation with custom tasks:
  - Role defines all service handlers, even ones it doesn't use itself
  - Play includes role to perform basic configuration and make handlers available
  - In 'tasks:', modify a config file that impacts the service managed by that role
  - Notify handlers defined by the role

```yaml
- name: start nginx
  become: yes
  service:
    name: nginx
    state: started

- name: stop nginx
  become: yes
  service:
    name: nginx
    state: stopped

- name: restart nginx
  become: yes
  service:
    name: nginx
    state: restarted

- name: reload nginx
  become: yes
  service:
    name: nginx
    state: reloaded
```

Role Handler Execution Order

- Handlers notified from (not defined in!) roles execute *slightly* differently
  - Don't use both 'tasks:' and 'post_tasks:' in a playbook if you can avoid doing so!
- **Never** flush handlers in your role. It can lead to unpredictable and/or dangerous behaviors.

Handler order:

1. Each task defined in 'pre_tasks:'
2. Handlers notified in 'pre_tasks:'
3. Each task from each role defined in 'roles:'
4. Each task defined in 'tasks:
5. **Handlers notified in 'roles:'**
6. **Handlers defined in 'tasks:'**
7. Each task defined in 'post_tasks:'
8. Handlers notified in 'post_tasks:'

(Repeat this process for each play!)

1.8 Ansible Roles: Hands-On

(5 minute demo, 15 minute exercise)

- Demo the ansible-galaxy command to create a role skeleton.
- Switch your web application playbook to use roles

Demo:

ansible-galaxy init test -p roles

Exercise:

Create a role to deploy a Flask app with a templated configuration file and start/stop handlers, then modify *playbooks/1.6_webapp.yml* to use it.

(Stuck? Check out *1.8_webapp.yml!)*

5 Minute Break

1.9
Debugging Ansible

5 minute lecture/demo

- Demo key debugging tools and commands

Basic Debugging

- 'debug' can print a message with "msg=The value of foo is: {{ foo }}", or dump a variable with "var=foo"
  - Common errors: results.stdout vs. results.stderr, result list from with_items
- The 'fail' module can stop a play partway through
  - or you can set 'failed_when: true' on any task
- 'ignore_errors' on a task can get you past a failure while still performing subsequent steps

Intermediate Debugging

- Verbose mode
  - -v: print task output
  - -vv: print ssh connection info
- Open an interactive debugger:
  - Modify the play to include 'strategy: debug'
  - Step through task execution
  - Change task arguments and retry failing tasks

```
    docker:
    docker: TASK [Set up Laravel env] ******************
    docker:
    docker: <127.0.0.1> ESTABLISH LOCAL CONNECTION FOR
    docker: <127.0.0.1> EXEC /bin/sh -c '( umask 77 && 
le/ansible-tmp-1470237990.51-205183799058632 `" && echo
58632="` echo /tmp/`whoami`/.ansible/ansible-tmp-147023
 0'
    docker: <127.0.0.1> PUT /tmp/tmpU1iTol TO /tmp/whoa
    docker: <127.0.0.1> EXEC /bin/sh -c 'LANG=en_US.UTF
.harvard.edu:8888 LC_MESSAGES=en_US.UTF-8 no_proxy=loca
_ALL=en_US.UTF-8 http_proxy=http://rcsocks.rc.fas.harva
mi/command; rm -rf "/tmp/whoami/" > /dev/null 2>&1 && s
    docker: changed: [docker] => {"changed": true, "cmd
r/www/html/.env && cp /tmp/nrg_status/files/entrypoint.
od u+x /var/www/html/entrypoint.sh", "delta": "0:00:00.
176376", "invocation": {"module_args": {"_raw_params":
ww/html/.env && cp /tmp/nrg_status/files/entrypoint.sh
u+x /var/www/html/entrypoint.sh", "_uses_shell": true,
utable": null, "removes": null, "warn": true}, "module_
"2016-08-03 15:26:30.800811", "stderr": "", "stdout": "
}
    docker:
    docker: PLAY RECAP *********************************
    docker: docker                     : ok=15   change
    docker:
==> docker: Provisioning with shell script: /tmp/packer
==> docker: Committing the container
    docker: Image ID: 707bc57e0bb449bd4f765e24e00bd3ca8
==> docker: Killing the container: ec867edd96648c207a18
bfe634e
```

Advanced Techniques

- Use `ps` to look for Ansible processes and strace them: *(sudo) strace -p PID*
  - Stuck on an unexpected prompt?
  - Waiting to acquire a lock?
- *ANSIBLE_KEEP_REMOTE_FILES=1 ansible-playbook my-playbook.yml*
  - One remote file will be created per task
  - Combine with -vv mode and look for the file paths
  - Execute them as standalone scripts on the remote server (attach debugging tools!)
  - *Clean them up afterwards!*

1.10 Deploying Applications with Ansible

5 minute lecture
25 minute exercise

- Use a mix of downloaded and hand-written roles to build a basic web application on your AWS instance
- Test your use of:
  - Playbooks
  - Variables
  - Roles
  - Inventory
  - Ansible commands

Exercise:

Continue to improve *playbooks/1.6_webapp.yml* with new functionality:

- Install nginx, Redis, and supervisord via roles
- Write a custom role to manage Flask with supervisord
- Persist number of visits in Redis
- Increment and display visits to user

1.11
Ansible's Strengths & Weaknesses

10 minute discussion

- Summarize what we've learned so far
- Present some perceived strengths/weaknesses of Ansible
- Encourage learners to discuss and tie it to what they've learned so far in their hands-on work

How Does Ansible Compare?

- Chef & Puppet: expect complete control of your system, require more buy-in
- Terraform: all-encompassing system, no orchestration
- CloudFormation: declarative configuration language, can't perform tasks
- Fabric: older, just orchestration, much less support for config management
- Salt: most similar, but not as widely adopted

Strengths and Weaknesses

- About the simplest approach that could possibly work
- Requires no agent and minimal configuration
- Runs almost anywhere
- Trivially extensible

- High flexibility means it's not always obvious the best way to do something
- Not a complete system out of the box
- It's a huge toolkit with hundreds of modules you will never use

Ansible: a toolkit, not a system!

1.12 Conclusion

10 minute discussion

- Recap of next day's plans
- Open Q&A with instructor

Course Outline

Day One:

- Cover course goals
- Introduce Ansible and its history
- Connect to training environment
- Ansible concepts:
  - Modules
  - Playbooks
  - Roles
- Practice writing playbooks and roles
- Deploy a single-node web application
- Compare Ansible to other tools

Day Two:

- Advanced module and role tips
- Explore cloud-focused features
- Deploy a multi-tier, multi-instance application in AWS
- Discuss code quality for Ansible
- Develop a custom Ansible module
- Learn about the Ansible ecosystem
- Recap what we've learned and revisit course goals

Questions?

See you tomorrow!

2.1 Introduction

5 minute lecture

- Recap of yesterday's material
- Describe goals for today
- Q&A on today's goals and last night's readings

Course Outline

Day One:

- Cover course goals
- Introduce Ansible and its history
- Connect to training environment
- Ansible concepts:
  - Modules
  - Playbooks
  - Roles
- Practice writing playbooks and roles
- Deploy a single-node web application
- Compare Ansible to other tools

Day Two:

- Advanced module and role tips
- Explore cloud-focused features
- Deploy a multi-tier, multi-instance application in AWS
- Discuss code quality for Ansible
- Develop a custom Ansible module
- Learn about the Ansible ecosystem
- Recap what we've learned and revisit course goals

2.2
Key Ansible Modules

15 minute lecture

- Cover the modules pointed to with individual documentation readings

- Point out some important quirks/gotchas for these common modules

- Module gotchas in general

file

- Name is misleading:
  - Delete files
  - Create directories or symlinks
  - 'touch' files
  - Change file permissions or owner
- You can 'become' a user and create files as them, or you can 'become' root and pass 'owner:' and 'group:'
  - Pro: Will respect existing permissions, can be run without root
  - Con: Can fail if the user doesn't exist

copy

- Transfers local files to the remote node
  - 'fetch' does the reverse
- For quick tasks, you can set 'content:' and provide a string (or variable) instead
- Look into 'synchronize' (rsync wrapper) if you have more complex needs

template

- Same execution environment as inside of Ansible playbooks (e.g. custom filters)
- But unlike in Ansible playbooks, Jinja macros can be used in template files
- Path will default to looking for templates relative to the current role, or otherwise the current playbook

lineinfile & replace

- Templating is not always desired; sometimes you want to tweak an existing file without replacing it
  - appending to .bashrc
  - modifying a non-Ansible-managed service
- 'replace' is simpler, but more limited
- 'lineinfile' (and 'blockinfile') are incredibly dangerous, but very powerful
  - Think 'chainsaw'

Safe Templating In Ansible

- This is a great way to render your machine broken!
  - c.f. visudo
- 'validate' argument to replace/lineinfile/template module that:
  - Templates the file to a temporary location
  - Runs an arbitrary command to validate it
  - Replaces the existing file if it validates
  - Optionally, backs up the original file
- Consider adding an 'Ansible Managed' header:
  - {{ansible_managed}}

command & shell

- 'command:' doesn't provide a shell, so you won't have: pipes, redirects, etc.
  - But in some contexts it can be safer than shell. Know what you're doing!
- 'shell' provides these
  - But it's /bin/sh by default, so e.g. 'source' won't work
  - Won't load a .bashrc even if you set the shell to bash
- Ansible can't tell whether command modules are idempotent, so make sure to include 'changed_when:'
- 'register:' saves the output, which includes the RC. If that isn't enough, you might need: *"changed_when: 'no changes' not in output.stderr"*
  - Remember, *output.stdout* and *output.stderr* are different streams!
- Remember to use nohup if you're launching persistent processes!
  - Or just deploy it as a service or with supervisord...

apt, yum, & pip

- There's no "general" package manager module, so to support multiple distros, you may need multiple package manager calls and conditionals
  - They usually have different package names anyways
- If you're using pip, make sure to account for virtualenvs
- Package manager modules "flatten" calls to "with_items:" so that they get condensed into a single package manager call

service & systemd

- Restart, reload, or otherwise manage services
- "service" will work on systemd systems, but it will be missing a few systemd-specific features
- These tasks are commonly used to implement handlers
- Remember that this will typically require 'become' on these tasks!

2.3
Ansible Roles: Logic

20 minute demo

- Explore Ansible Galaxy
- Examine popular public roles

Ansible Galaxy

- Comparable to Chef Supermarket or npm
- Much smaller
- Inconsistent quality
  - We'll talk about this later!
- However, still check it out first!
  - Even if you don't use a role, you might want to base your own code on it.

Most Downloaded

Role	Downloads
geerlingguy.redis	38130
nickhammond.logrotate	33169
geerlingguy.composer	31337
geerlingguy.mysql	28705
geerlingguy.java	28082
geerlingguy.daemonize	27094
geerlingguy.nginx	26801
geerlingguy.php	25554
geerlingguy.mailhog	25429
Stouts.mongodb	24809

Demo:

https://galaxy.ansible.com/geerlingguy/nginx/

Demo:

https://galaxy.ansible.com/tersmitten/supervisor/

5 Minute Break

2.4
Ansible at Scale

5 minute lecture
5 minute demo

- Inventories and groups
- Dynamic inventories
- Working with multiple hosts
  - Parallelism
  - Rolling updates
  - Delegation

Inventory

- Named list of remote nodes
- Groups can be defined with ':children'
- Groups can be nested in other groups
- Nodes and/or groups can be combined via 'pattern matching':
  - OR: webservers:dbservers
  - AND: webservers:&staging
  - NOT: webservers:!phoenix
- Host definitions can include:
  - SSH connection information
  - User to run commands as
  - Python executable path
  - Variables **(not recommended!)**

```
[atlanta]
host1
host2

[raleigh]
host2
host3

[southeast:children]
atlanta
raleigh

[southeast:vars]
some_server=foo.southeast.example.c
halon_system_timeout=30
self_destruct_countdown=60
escape_pods=2

[usa:children]
southeast
northeast
southwest
northwest
```

```
# MGH infrastructure
[mgh]
embarc
nexuscentral
gspcentral
simons
gspstaging
launchpad

# RC-managed infrastructure

# Specific RC nodes
[rc]
cbscentral
cbsstaging
contecentral
nrgcentral

# Groups that fall under RC
[rc:children]
ncf
workstations
nrg
```

Dynamic Inventories

- Dynamic inventories replace a static list of hosts with a data-driven one
- Implemented as a script in the inventory directory that calls APIs, reads CSVs, ...
- There are existing dynamic inventories for EC2, Google Compute Engine, Linode, OpenStack, and others
- EC2 inventory: lists each host, but also creates groups for each EC2 tag

```
#!/usr/bin/env python

'''
EC2 external inventory script
=================================

Generates inventory that Ansible can understand by making API request to
AWS EC2 using the Boto library.

NOTE: This script assumes Ansible is being executed where the environment
variables needed for Boto have already been set:
    export AWS_ACCESS_KEY_ID='AK123'
    export AWS_SECRET_ACCESS_KEY='abc123'

optional region environement variable if region is 'auto'

This script also assumes there is an ec2.ini file alongside it.  To specify a
different path to ec2.ini, define the EC2_INI_PATH environment variable:

    export EC2_INI_PATH=/path/to/my_ec2.ini

If you're using eucalyptus you need to set the above variables and
you need to define:

    export EC2_URL=http://hostname_of_your_cc:port/services/Eucalyptus

If you're using boto profiles (requires boto>=2.24.0) you can choose a profile
using the --boto-profile command line argument (e.g. ec2.py --boto-profile prod) or using
the AWS_PROFILE variable:

    AWS_PROFILE=prod ansible-playbook -i ec2.py myplaybook.yml

For more details, see: http://docs.pythonboto.org/en/latest/boto_config_tut.html

When run against a specific host, this script returns the following variables:
 - ec2_ami_launch_index
 - ec2_architecture
 - ec2_association
 - ec2_attachTime
 - ec2_attachment
 - ec2_attachmentId
 - ec2_block_devices
 - ec2_client_token
 - ec2_deleteOnTermination
 - ec2_description
 - ec2_deviceIndex
 - ec2_dns_name
 - ec2_eventsSet
 - ec2_group_name
 - ec2_hypervisor
 - ec2_id
```

Parallelism

- Ansible connects to multiple nodes and runs tasks simultaneously
- All nodes must complete (or fail) a task before the next task starts
  - You can bypass this by using 'strategy: free' to allow hosts to complete each *play* as fast as they can
- Ansible defaults to 5 forks - very low! Can usually be increased to 50 or more

Demo:
ansible-playbook playbooks/2.4_parallel.yml

Rolling Updates

- You may not want to run a play on all hosts at the same time
  - You have a rate limit on an API
  - You don’t want to overload the database while it’s still spinning up
  - You want special treatment, like making every n-th node a leader
- Set to ‘serial: 1’ to finish the whole play on each host before starting the next
- Set to ‘serial: 20%’ to run the play in five batches
- Include ‘max_fail_percentage:’ to bail out of the play early

Demo:
ansible-playbook playbooks/2.4_rolling.yml

Delegation

- Tasks usually run:
  - once per host
  - with that host's variables
  - on that host
- 'delegate_to:' allows tasks to run:
  - once per host
  - with that host's variables
  - *on a different host*
- Not commonly used, but very convenient!
  - Get values from each host, delegate to localhost, and write each to disk
  - Delegate to a leader node and send an instruction to each follower

Demo:
ansible-playbook playbooks/2.4_delegate.yml

2.5
Ansible in the Cloud

5m lecture
10m exercise

- Introduce cloud management modules
- Provision additional instances with Ansible
- Use dynamic inventory to request data from instances

ec2

- Built in module to provision, manage, and terminate EC2 instances: http://docs.ansible.com/ansible/ec2_module.html
- Other relevant EC2 modules:
  - ec2_facts: Get info about EC2 instances
  - ec2_tag: Just tagging, not provisioning
  - ec2_vol_facts: Get volume information

```
# Basic provisioning example
- ec2:
    key_name: mykey
    instance_type: t2.micro
    image: ami-123456
    wait: yes
    group: webserver
    count: 3
    vpc_subnet_id: subnet-29e63245
    assign_public_ip: yes

# Advanced example with tagging and CloudWatch
- ec2:
    key_name: mykey
    group: databases
    instance_type: t2.micro
    image: ami-123456
    wait: yes
    wait_timeout: 500
    count: 5
    instance_tags:
       db: postgres
    monitoring: yes
    vpc_subnet_id: subnet-29e63245
    assign_public_ip: yes
```

Exercise:

Provision a new instance with *playbooks/2.5_provision.yml*

Connect to your new instance with *playbooks/2.5_hello.yml*

2.6: Cloud Orchestration

5 minute lecture
25 minute exercise

- Deploy a multi-tier, multi-instance application in AWS
- Successfully serve traffic
- Advanced goal: perform a rolling restart after an upgrade

Exercise:

Provision a multi-tier app, with services on both your control machine and the instance you provisioned from *2.5_provision.yml*. Start from this playbook:

ansible-playbook playbooks/2.6_cloud.yml

10 Minute Q&A/Break

2.7
Writing Maintainable Ansible Code

15 minute lecture

- Discuss tips and tricks for making Ansible code more maintainable

What Is Maintainable Code?

- Dependency managed
  - Well-known
  - Version pinned
  - In code
- Documented
  - Conceptual/"why" (high-level, code-light documents kept up to date)
  - Practical/"how" (focused, concise comments)
- Only as complex as necessary
  - Standards-oriented
  - Minimize "clever tricks"
  - Principle of least surprise!
- Tested *and* regularly used
  - unit tests
  - integration tests
  - continuous integration
  - continuous deployment
- Production-ready
  - Monitoring
  - Alerting
  - Logging
  - Well-understood deployment process
- Human friendly
  - Easily readable and modifiable
  - More than one person understands it
  - Can be run either automated or manually

Structure Your Repository

http://docs.ansible.com/ansible/playbooks_best_practices.html#content-organization

```
production                # inventory file for production servers
staging                   # inventory file for staging environment

group_vars/
   group1                 # here we assign variables to particular groups
   group2                 # ""
host_vars/
   hostname1              # if systems need specific variables, put them here
   hostname2              # ""

library/                  # if any custom modules, put them here (optional)
filter_plugins/           # if any custom filter plugins, put them here (optic

site.yml                  # master playbook
webservers.yml            # playbook for webserver tier
dbservers.yml             # playbook for dbserver tier
```

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies

    webtier/              # same kind of structure as "common" was above, done for the webtier role
    monitoring/           # ""
    fooapp/               # ""
```

Structure Your Repository

- I prefer a slight variant on the Ansible recommendations
- Overall structure is a single repo - possible to share structure with others
- Each environment gets its own folder with independent inventories, group_vars, etc.:
  - environments/vagrant, environments/staging, environments/production
- Roles can be placed directly in the main repo (convenient), or split out into their own repo (better for open source releases)
- Playbooks folder can be placed in the main repo, or split into their own repo and versioned independently, depending on project complexity

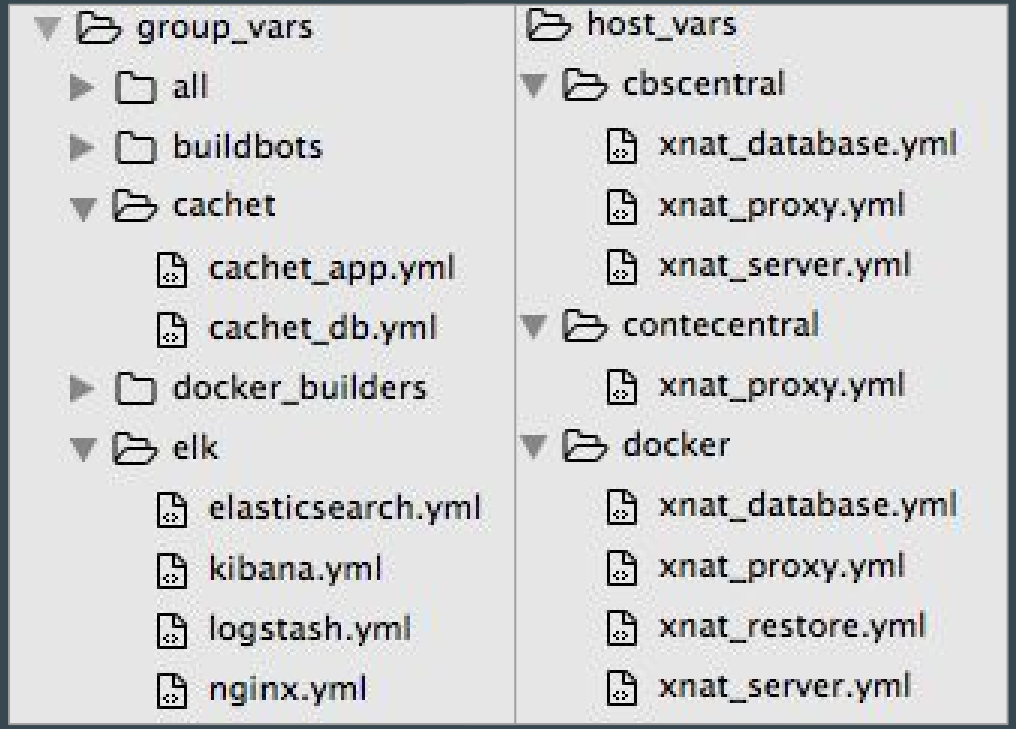
Structure Your Variables

- Probably the single most common source of Ansible maintainability problems
- Ansible variables can come from 16 different sources, and many of those sources have within-source precedence rules too
- There is no effective debugging tool for figuring out what source set which variable
- Minimize complexity as much as possible:
  - Decide which sources you'll use, and when
  - Develop standards to use cross-project
  - Be disciplined, even when it's inconvenient

role defaults (lowest!)
inventory vars
inventory group_vars
inventory host_vars
playbook group_vars
playbook host_vars
host facts
play vars
play vars_prompt
play vars_files
registered vars
set_facts
role and include vars
block vars (only for tasks in block)
task vars (only for the task)
extra vars (highest!)

Structure Your Variables

- host_vars and group_vars are a good starting point for most organizations
- host_vars has precedence over group_vars on a per-variable basis
  - Keep most configuration general
  - Selectively override specific variables
- Instead of single configuration files, store host or group config as folders of logically separated concerns
  - All .yml files in a subfolder get included



Structure Your Playbooks

- Play settings should be variables, but with sensible defaults provided:
  - hosts: {{play_hosts|default('all')}}
- Avoid using pre_tasks for anything that doesn't *need* to be run before roles.
- Avoid using 'pre_tasks:', 'roles:', 'tasks:' *and* 'post_tasks:' in the same playbook. There are few cases where this level of complexity is necessary.
- If you think you'll want to run a play by itself, put it in its own playbook file. You can always include it from other playbooks.

Structure Your Roles

- Provide a sensible but general vars/main.yml.
- Consider providing contingent override files in in vars/ specific to environments, OSes, cloud providers, etc.
- Define a full set of handlers (start, stop, restart, reload) for services managed by your role. Even if you aren't using them, other developers might try to later, and their code won't fail until runtime.
- ***Roles should never flush handlers or restart services without providing a way to override this behavior.*** You could cause security or stability issues in an unrelated role!

Avoid Nested Variables In Tasks

- Ansible's control structures are very weak
- Writing more complex structures in Jinja is almost unreadable
- Loops are almost impossible after more than two levels of nesting
- Consider implementing complex loops inside of custom modules
- Jinja template files *can* still have arbitrary nesting! (Loops, subtemplates)

Write Idempotent Playbooks

- Most Ansible modules are idempotent:
  - "Modules are ‘idempotent’, meaning if you run them again, they will make only the changes they must in order to bring the system to the desired state. This makes it very safe to rerun the same playbook multiple times. They won’t change things unless they have to change things."
- “command” and “shell” modules are not inherently idempotent!
- A playbook of idempotent modules is idempotent
  - Important exception: using handlers breaks idempotency guarantee in the case of failures!
- Idempotency is hard to achieve in practice, but it’s a valuable goal.

Avoid Orphaned Files In Tasks

- Ansible has no knowledge of files generated on previous runs
- The naive approach will result in ‘orphaned’ configuration files, SSH keys, etc.
  - Define ‘app_name: blue’
  - Playbook templates out ‘blue.conf’
  - Change app_name to ‘green’
  - Playbook templates out ‘green.conf’, *ignoring* ‘blue.conf’
  - Webserver loads ‘blue.conf’ *and* ‘green.conf’
- Instead, remove and recreate files with variable-based names
- Can be challenging to implement pattern across roles or playbooks

2.8
Ansible Roles: Analyzing Code Quality

15 minute discussion

- Go over sample code from several public Ansible Galaxy roles
- Discuss code quality issues and how they might impact your Ansible deployment

Ansible Galaxy

- The tool shipped with Ansible: ansible-galaxy
- The website it connects to: https://galaxy.ansible.com/
- Great place to learn how other people build roles!
- But take everything with a grain of salt
  - Hugely varying quality
  - Some roles work, but won't compose with others well
  - Low number of ratings, so hard to judge

Case Study: AWS CLI

https://github.com/dstil/ansible-aws-cli/blob/master/tasks/main.yml

Case Study: Consul

- Consul Role #1: https://galaxy.ansible.com/andrewrothstein/consul/
- Consul Role #2: https://galaxy.ansible.com/brianshumate/consul/
- Consul Role #3: https://galaxy.ansible.com/alexanderjardim/consul/

Case Study: Elasticsearch

- Elasticsearch Role #1: https://galaxy.ansible.com/jpnewman/elasticsearch/
- Elasticsearch Role #2: https://github.com/geerlingguy/ansible-role-elasticsearch/

5 Minute Break

2.9
Customizing Ansible

5 minute lecture
15 minute exercise

- Discuss customization options
- Describe API for modules
- Write a module in the language of your choice

Custom Dynamic Inventories

- You can write your own dynamic inventory scripts for cloud providers, managed devices, etc.
- http://docs.ansible.com/ansible/dev_guide/developing_inventory.html

Custom Plugins

- Modifications to Ansible's functionality, rather than providing new tasks
- Written in Python, stored in global location, loaded with config file
- Usually not necessarily, though you might use them for implicit cross-playbook implementation instead of explicitly calling a role

Ansible supports many plugin types:

- Connection (alternatives to SSH)
- Callback (output formatting)
- Lookup (use data from a database, file, etc.)
- Vars (Provide arbitrary custom variables)
- Filters/Tests (extend Jinja2 with Python)
- ...and more!

Custom Modules

- Extend Ansible to perform new tasks
- You can implement your own modules by providing executable code that conforms to a JSON-based specification
- Core Ansible modules are in Python; Python modules are easier, but any language will work!
- Whatever executable you push to the remote node needs to be runnable there
    - But you can install dependencies in previous tasks, even in the same role

```
#!/usr/bin/ruby
# WANT_JSON

require 'rubygems'
require 'json'

# this is a bare minimum example of a 'facts' module that returns some variables into the ansible
# namespace.  It may not be sufficiently idiomatic and doesn't do a lot of error checking.

File.open(ARGV[0]) do |fh|

   data = JSON.parse(fh.read())

   begin
      a = data['a'].to_i()
      b = data['b'].to_i()
   rescue
      # to raise an error, return failed=True and a msg string.

      print JSON.dump({
          'failed' => true,
          'msg'    => 'failed to parse inputs x or y'
      })

      # the error code here is not so important, the JSON is!

      exit(1)
   end

   # we may also wish to return changed=True or changed=False
   # if we were modifying system resources to support handlers and change tracking
   # if the module decides to not run, it can also return skipped=True

   result = {
      'a'   => a,
      'b'   => b,
      'sum' => a + b,
   }
```

Exercise:

Write your own Ansible module

2.10
The Ansible Ecosystem

(10 minute lecture)

- Community resources
- Paid tools and support
- Online learning

Ansible Documentation

- http://docs.ansible.com/
- Very detailed module references - check there first!
- Ansible moves fast, so don't trust code samples from Google
  - *Make sure you're using the right documentation version!*
- Missing more in-depth use cases and real world examples

Official Example Playbooks

- https://github.com/ansible/ansible-examples
- More realistic than www.ansible.com snippets
- Still lacking in complexity - if you need better examples, consider looking at popular roles on Ansible Galaxy.

Ansible On Github

- Developed on GitHub: https://github.com/ansible/ansible
- “Releases are named after Led Zeppelin songs.”
  - “(Releases prior to 2.0 were named after Van Halen songs.)”
- It’s possible to run an Ansible git checkout, including local modifications
- Comes with a set of shell scripts to configure dependencies properly: https://github.com/ansible/ansible/blob/devel/hacking/README.md
- Great for custom module development!

Mailing Lists

- Announcements: https://groups.google.com/forum/#!forum/ansible-announce
- General discussion: https://groups.google.com/forum/#!forum/ansible-project
- Dev discussion: https://groups.google.com/forum/#!forum/ansible-devel

Ansible IRC

- All on irc.freenode.net:
  - #ansible: general discussion, support
  - #ansible-devel: developer discussion
  - #ansible-meeting: community meetings

Conferences and Meetups

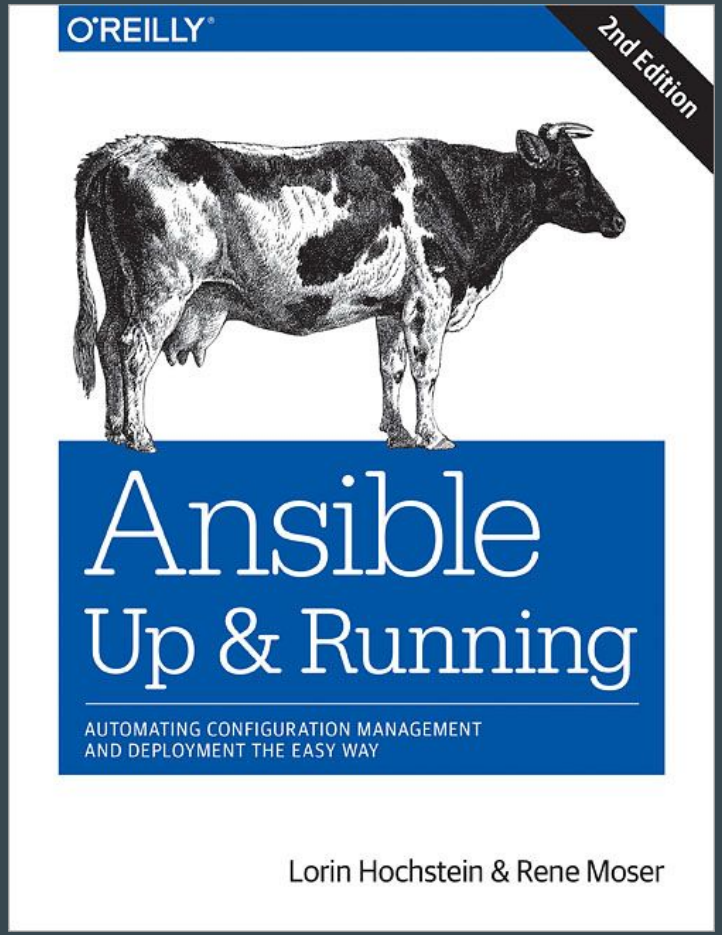
- AnsibleFest: https://www.ansible.com/ansiblefest
- Local meetups: https://www.meetup.com/topics/ansible/
  - Or check out your local DevOps, Infrastructure Coders, etc. meetup!

Red Hat

- Offers paid certifications in Ansible: https://www.ansible.com/training-certification
- Runs in person and online trainings: https://www.ansible.com/webinars-training
- Currently, no enterprise support offering
- But they do have paid Ansible consulting: https://www.ansible.com/consulting

O'Reilly Media

- Ansible: Up & Running is available for purchase, with several chapters free: http://www.ansiblebook.com/
  - Second edition is releasing soon!
- Network Automation With Ansible: http://www.oreilly.com/webops-perf/free/network-automation-with-ansible.csp

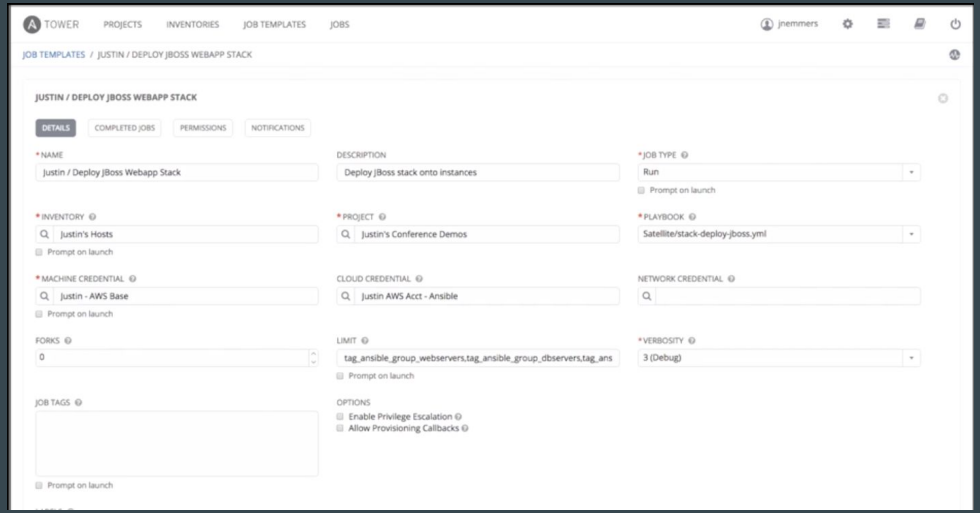


Ansible Tower

- Paid product offered by Red Hat
- Automatic updates to inventory and code to keep in sync with cloud and VCS
- ACLs on who can run jobs or view their output
- Provisioning callbacks for new cloud servers to request that Tower run Ansible playbooks on them
- On-run prompts for credentials and variables
- Paid support offerings!

Downsides of Ansible Tower

- User interface needs work
  - “Single page” application
  - Flaky backend
  - Confusing UX decisions
  - Maximum output size
- Deployment and maintenance are difficult
- Scalability issues
- Lack of control over per-job Ansible settings and versions
- Tower API callback issues are difficult to debug
- Overall, still a great project for its age



2.11 Conclusion

(5 minute lecture)

- Recap of course goals
- What we've learned
- Q&A with instructor

Course Goals

During this course, you'll learn...

- Why companies are adopting Ansible over other tools
- Key Ansible concepts: variables, modules, playbooks, roles, and more
- Best practices for writing reusable, maintainable code in Ansible

After the course, you'll be able to...

- Write Ansible roles and playbooks to automate routine operations tasks
- Orchestrate deployments of multi-tier applications
- Manage your infrastructure as code, whether in the cloud or the data center
- Drive adoption of Ansible within your team through incremental adoption

Adoption

- "Batteries included", with support for hundreds of common technologies
- Flexible and powerful
- Easy to bring into existing environments, whether legacy or cloud
- Open source
    - But with many support options from the community and from Red Hat

Concepts

- Discussed variables, modules, playbooks, roles, and more
- Implemented playbooks, roles, and even a custom module
- Used Ansible to deploy several applications on real architecture

Best Practices

- Emphasizing code quality for infrastructure code
- Writing easily composed roles and playbooks
- Understanding when not to use Ansible's flexibility
- Treating Ansible as a toolkit, not a system

Questions?

Thank you!

Feel free to connect!

Email: eronarn@gmail.com
LinkedIn: https://www.linkedin.com/in/eronarn/
Twitter: https://twitter.com/jmeickle
Web: https://permadeath.com

Special thanks to Brian and everyone else at O'Reilly Media for providing this platform. I'm also grateful to Quantopian for supporting Ansible education internally, to the Neuroinformatics Research Group at Harvard for letting me adopt Ansible, and to the DevOpsDays Boston audience for the initial test run of this course.

And to all of you for attending! <3