Infrastructure as Code through Ansible

Matt Heckathorn

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Document Markings

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-1013

Infrastructure as Code

What is it?

IaC Defined

Per wikipedia:

Infrastructure as code (IaC) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

The definitions may be in a version control system. It can use either scripts or declarative definitions, rather than manual processes, but the term is more often used to promote declarative approaches.

Common IaC tools

If you search the Internet for “infrastructure-as-code”, it’s pretty easy to come up with a list of the most popular tools:

- Chef
- Puppet
- Ansible
- SaltStack
- CloudFormation
- Terraform

All these tools have different features and use cases that can make choosing difficult

How to choose a tool?

Here are some trade-offs to consider:

- Configuration Management vs Orchestration
- Mutable Infrastructure vs Immutable Infrastructure
- Procedural vs Declarative
- Client/Server Architecture vs Client-Only Architecture

Configuration Management vs Orchestration

Configuration management tools are designed to install and manage software on existing servers

- Puppet
- Chef
- Ansible
- SaltStack

Orchestration tools are designed to provision the servers themselves, leaving the job of configuring those servers to other tools

- Terraform
- Cloudformation

Mutable Infrastructure vs Immutable Infrastructure

Mutable server infrastructure means servers are continually updated and modified in place. In other words, these servers are mutable; they can be changed after they're created.

- Configuration management tools such as Chef, Puppet, Ansible, and SaltStack typically default to a mutable infrastructure paradigm.
- Can lead to configuration drift

Immutable infrastructure means servers are never modified after they're deployed. If something needs to be updated, fixed, or modified in any way, new servers built from a common image with the appropriate changes are provisioned to replace the old ones. After they're validated, they're put into use and the old ones are decommissioned.

Procedural vs Declarative

Procedural programming can be viewed as code that specifies, step-by-step, how to to achieve some desired end state

- Chef and Ansible

Declarative programming tries to blur the distinction between a program as a set of instructions and a program as an assertion about the desired end state

- Terraform, CloudFormation, SaltStack, and Puppet

Master/Agent Architecture vs Master-Only Architecture

Master/Agent Architecture:

A master is responsible for cataloguing your desired commands and nodes to be managed. To execute those commands, the master talks to agents, which must be running on every node you want to configure.

This has a number of downsides:

- You have to install and run extra software on every one of your servers.
- You have to deploy an extra server (or even a cluster of servers for high availability) just for configuration management.
- You not only have to install this extra software and hardware, but you also have to maintain it, upgrade it, make backups of it, monitor it, and restore it in case of outages.

Master/Agent Architecture vs Master-Only Architecture

Master-Only Architecture:

- No agents running on nodes
- A master server communicates to clients via standard access methods
  - Includes ssh, provider APIs
  - Often easier-to-use and more maintainabile
- CloudFormation, Ansible, and Terraform, use a master-only architecture

Trade-Off Summary

	Chef	Puppet	Ansible	SaltStack	CloudFormation	Terraform
Code	Open source	Open source	Open source	Open source	Closed source	Open source
Cloud	All	All	All	All	AWS only	All
Type	Config Mgmt	Config Mgmt	Config Mgmt	Config Mgmt	Orchestration	Orchestration
Infrastructure	Mutable	Mutable	Mutable	Mutable	Immutable	Immutable
Language	Procedural	Declarative	Procedural	Declarative	Declarative	Declarative
Architecture	Client/Server	Client/Server	Client-Only	Client/Server	Client-Only	Client-Only

Ansible

Getting Started

Playbook Organization

https://docs.ansible.com/ansible/2.9/user_guide/playbooks_best_practices.html#directory-layout

Inventory File

https://docs.ansible.com/ansible/2.8/user_guide/playbooks_best_practices.html#how-to-differentiate-staging-vs-production

- Groups are nice for organization, but that's not all groups are good for. You can also assign variables to them!
- If we had any default values, or values that were universally true, we would put them in a file called group_vars/all

Ad-hoc Commands

```
# as bruce, sudoing to root
$ ansible all -m ping -u bruce -b
# as bruce, sudoing to batman
$ ansible all -m ping -u bruce -b --become-user batman
```

https://docs.ansible.com/ansible/2.9/modules/modules_by_category.html

Roles

https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_roles.html

- Roles are ways of automatically loading certain vars_files, tasks, and handlers based on a known file structure. Grouping content by roles also allows easy sharing of roles with other users.
- Use molecule to develop and test your ansible roles: https://molecule.readthedocs.io/en/latest/

Roles Continued

Roles expect files to be in certain directory names. Roles must include at least one of these directories, however it is perfectly fine to exclude any which are not being used. When in use, each directory must contain a main.yml file, which contains the relevant content:

- tasks - contains the main list of tasks to be executed by the role.
- handlers - contains handlers, which may be used by this role or even anywhere outside this role.
- defaults - default variables for the role (see Variables for more information).
- vars - other variables for the role (see Variables for more information).
- files - contains files which can be deployed via this role.
- templates - contains templates which can be deployed via this role.
- meta - defines some meta data for this role. See below for more details.

Search for user shared roles at: https://galaxy.ansible.com/

Plays

https://docs.ansible.com/ansible/2.9/user_guide/playbooks.html#working-with-playbooks

- At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way.
- Playbooks are expressed in YAML format (see YAML Syntax) and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process.
- Each playbook is composed of one or more ‘plays’ in a list.
- The goal of a play is to map a group of hosts to some well defined roles, represented by things ansible calls tasks. At a basic level, a task is nothing more than a call to an ansible module

Top Level Playbooks are Separated By Role

- In site.yml, we import a playbook that defines our entire infrastructure. This is a very short example, because it's just importing some other playbooks:

```
---
# file: site.yml
- import_playbook: webservers.yml
- import_playbook: dbservers.yml
```

Top Level Playbooks Continued

- In a file like webservers.yml (also at the top level), we map the configuration of the webservers group to the roles performed by the webservers group:

```
---
# file: webservers.yml
- hosts: webservers
  roles:
    - common
    - webtier
```

Top Level Playbooks The Third

- The idea here is that we can choose to configure our whole infrastructure by “running” site.yml or we could just choose to run a subset by running webservers.yml. This is analogous to the “–limit” parameter to ansible but a little more explicit:

```
ansible-playbook site.yml --limit webservers

ansible-playbook webservers.yml
```

Plays Examples

https://github.com/ansible/ansible-examples

Ansible

What’s left

Ansible Tower/AWX

Red Hat® Ansible® Tower helps you scale IT automation, manage complex deployments and speed productivity. Centralize and control your IT infrastructure with a visual dashboard, role-based access control, job scheduling, integrated notifications and graphical inventory management.

https://www.ansible.com/products/tower

Dynamic Inventory

https://docs.ansible.com/ansible/2.9/user_guide/intro_dynamic_inventory.html

Often a user of a configuration management system will want to keep inventory in a different software system. Ansible provides a basic text-based system as described in Working with Inventory but what if you want to use something else?

Frequent examples include pulling inventory from a cloud provider, LDAP, Cobbler, or a piece of expensive enterprise CMDB software.

Contact Information

Presenter / Point of Contact
Matt Heckathorn
Integration engineer
Telephone: +1 412.268.6642
Email: maheckathorn@cert.org