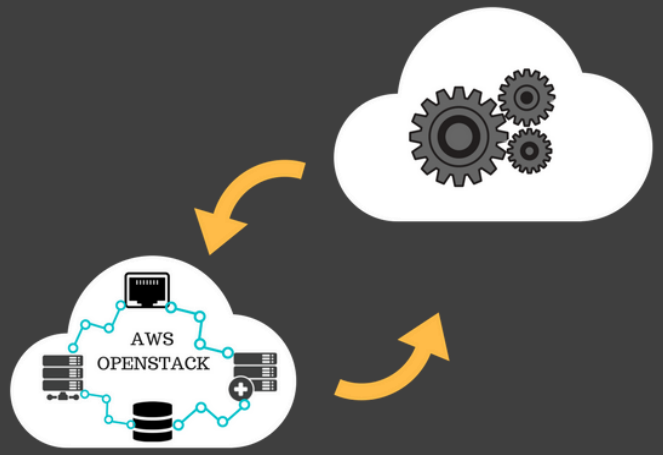
Course and Labs



AWS Design and Automation

Module 2: Automate in AWS

Topic 1: Cloud Formation Overview

Mohanraj Shanmugam

Ansible Overview

Infrastructure Automation

- *Infrastructure automation* is the process of scripting environments — from installing an operating system,

  to installing and configuring servers on instances,

  to configuring how the instances and

  software communicate with one another,

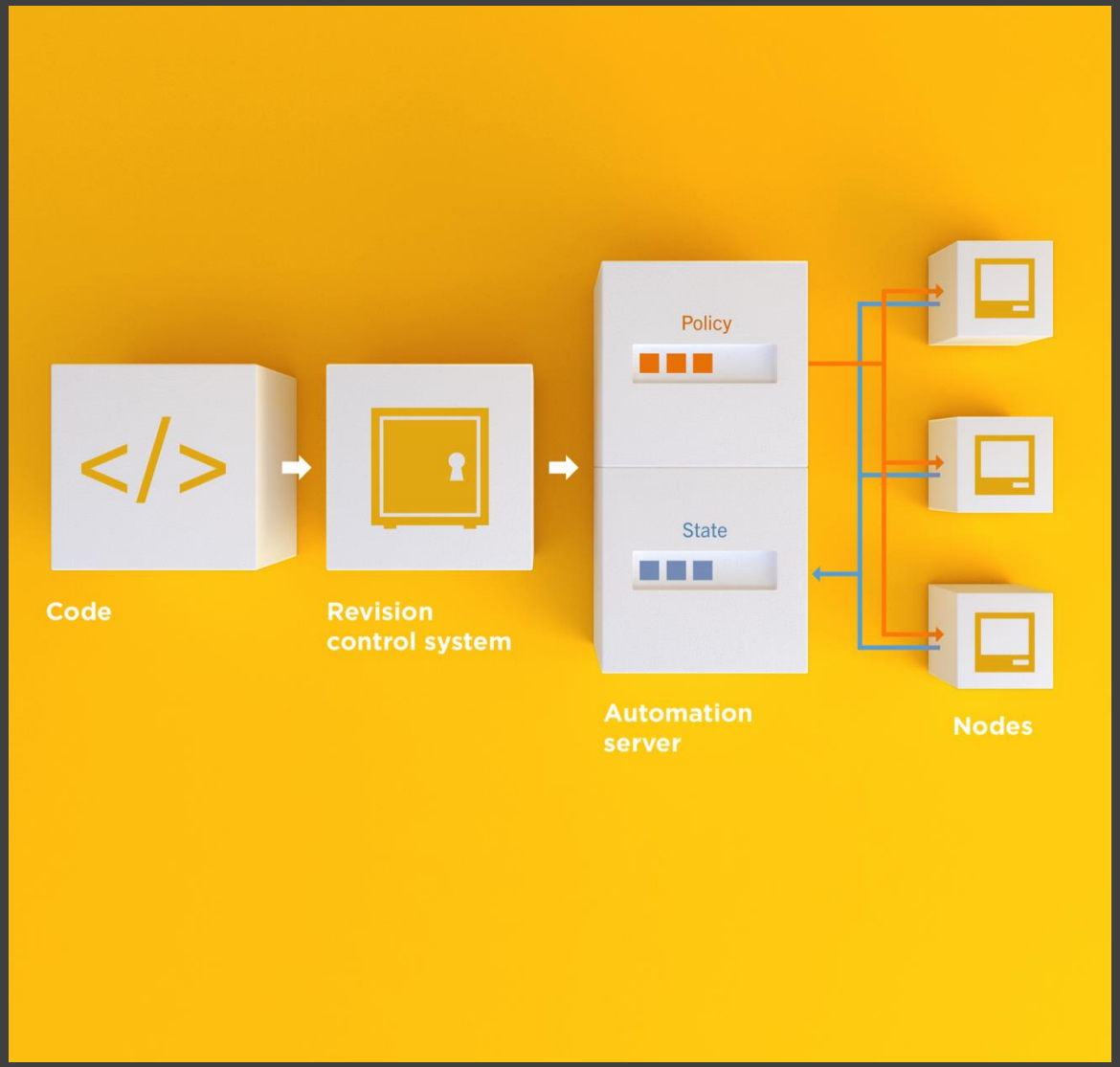
  and much more.

By scripting environments, you can apply the same configuration to a single node or to thousands.

Infrastructure Automation

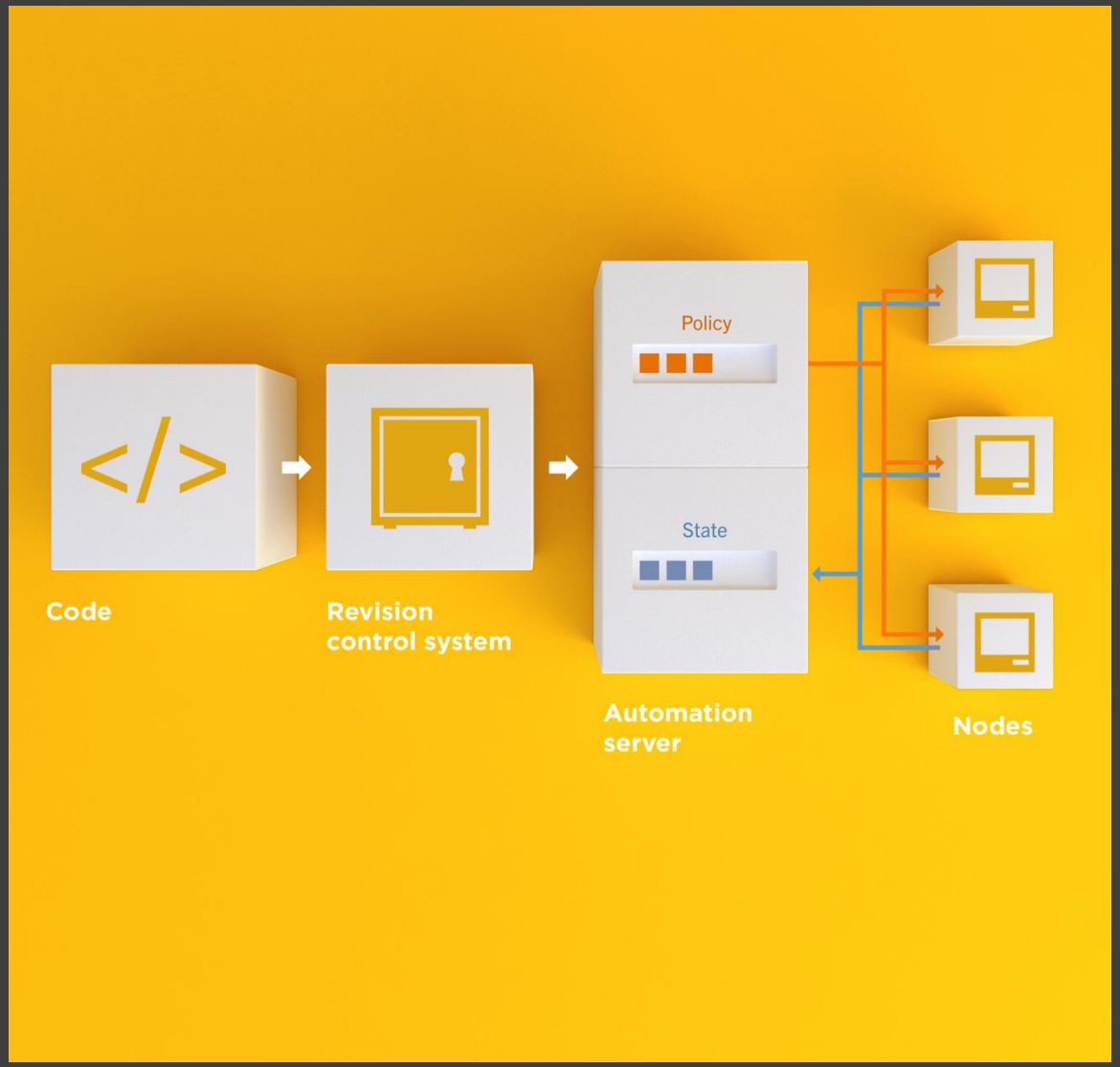
- Infrastructure automation also goes by other names:
  - configuration management,
  - IT management,
  - provisioning, scripted infrastructures,
  - system configuration management, and
  - many other overlapping terms.
- The point is the same: you are describing your infrastructure and its configuration as a script or set of scripts so that environments can be replicated in a much less error-prone manner.
- Infrastructure automation brings agility to both development and operations because any authorized team member can modify the scripts while applying good *development* practices — such as automated testing and versioning — to your infrastructure.

Infrastructure as Code



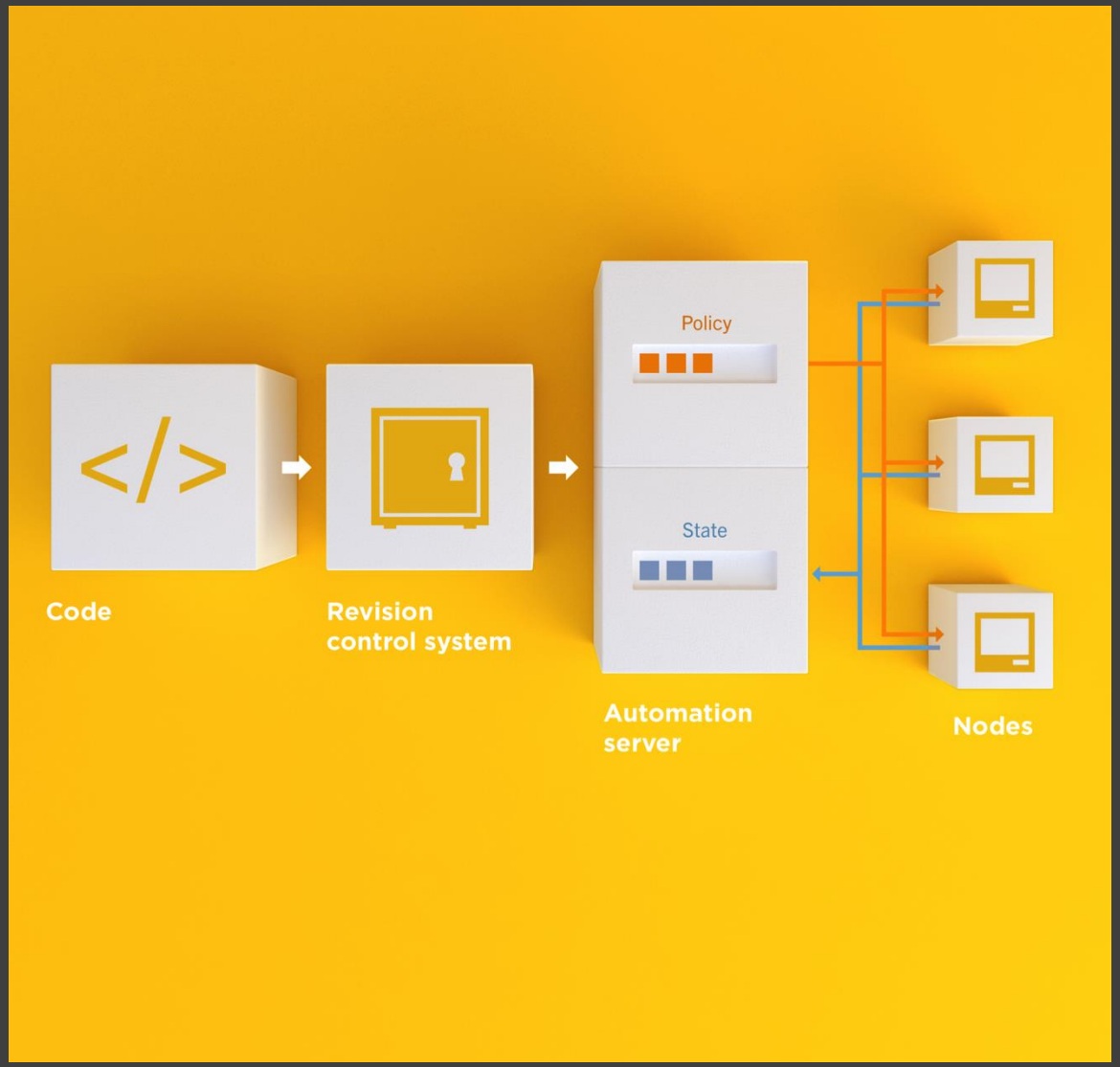
- Programmatically provision and configure components
- Treat like any other code base
- Reconstruct business from code repository, data backup, and compute resources

Infrastructure as Code



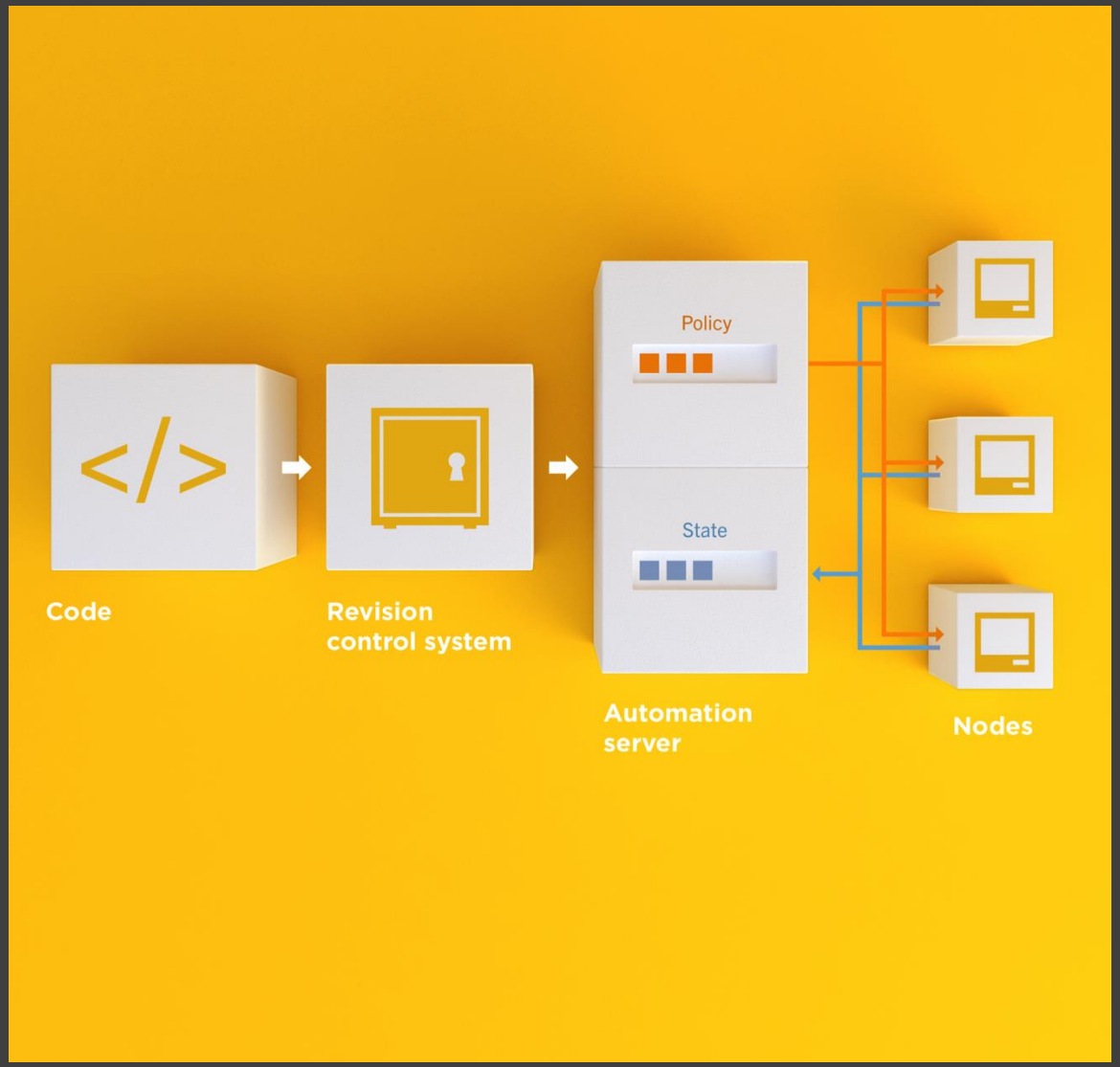
- Programmatically provision and configure components
- Use the Ansible programming language to define the components of your infrastructure.
- This allows you to capture and document the shape of your infrastructure in a consistent way.

Infrastructure as Code



- Treat like any other code base
- Treat this like any other code base:
  1. Store the code in a version control system.
  2. Add automated tests to the code.
  3. Refactor the code over time.
  4. Build and version software artifacts (e.g. packages)
  5. The code is executable documentation.

Infrastructure as Code



- Reconstruct business from code repository, data backup, and compute resources
- With your infrastructure now fully captured in code, you are now able to rebuild your entire business, well, at least all of your business's applications, with your code repository, a backup of your data, and compute resources, be they bare metal, virtual machines, or cloud instances.

Introduction to Ansible

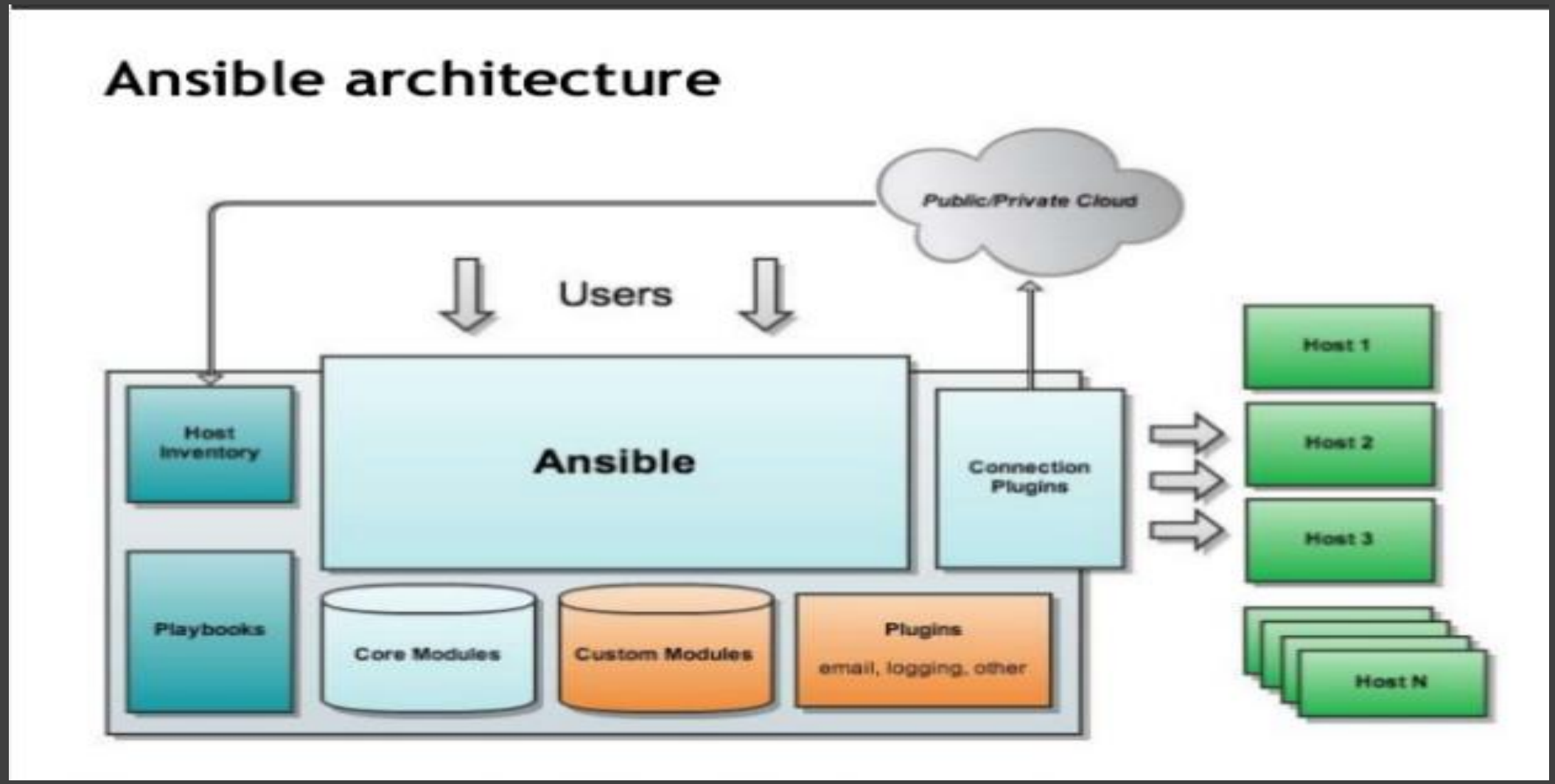
- Ansible is a radically simple IT automation platform that makes your applications and systems easier to deploy.

- It support configuration management with examples as below.
    - Configuration of servers
    - Application deployment
    - Continuous testing of already install application
    - Provisioning
    - Orchestration
    - Automation of tasks

Why Ansible

- It is a free open source application
- Agent-less – No need for agent installation and management
- Phython/yaml based
- Highly flexible and configuration management of systems.
- Large number of ready to use modules for system management
- Custom modules can be added if needed
- Configuration roll-back in case of error
- Simple and human readable
- Self documenting

Ansible Architecture



Inventory

Host 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.
- You can specify a different inventory file using the -i <path> option on the command line.
- Not only is this inventory configurable, but you can also use multiple inventory files at the same time

Host Inventory

- Hosts and Groups

mail.example.com

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

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

Host Inventory

- The headings in brackets are group names, which are used in classifying systems and deciding what systems you are controlling at what times and for what purpose.
- It is ok to put systems in more than one group, for instance a server could be both a webserver and a dbserver. If you do, note that variables will come from all of the groups they are a member of. Variable precedence is detailed in a later chapter.
- If you have hosts that run on non-standard SSH ports you can put the port number after the hostname with a colon. Ports listed in your SSH config file won't be used with the *paramiko* connection but will be used with the *openssh* connection.

Host patterns

- Adding a lot of hosts? If you have a lot of hosts following similar patterns you can do this rather than listing each hostname:
- [webservers]
- www[01:50].example.com
- For numeric patterns, leading zeros can be included or removed, as desired. Ranges are inclusive. You can also define alphabetic ranges:
- [databases]
- db-[a:f].example.com

Connection Type

- You can also select the connection type and user on a per host basis:
- Connection type to the host. This can be the name of any of ansible's connection plugins. SSH protocol types are ssh or paramiko. The default is Paramiko.

```
[targets]
localhost              ansible_connection=local
other1.example.com     ansible_connection=ssh       ansible_user=mpdehaan
other2.example.com     ansible_connection=ssh       ansible_user=mdehaan
```

Connection Plugin

- By default, Ansible talks to remote machines through pluggable libraries.
- Ansible supports native OpenSSH (SSH (Native)) or a Python implementation called paramiko.
- By default, Ansible manages machines over SSH.
- The library that Ansible uses by default to do this is a Python-powered library called paramiko.
- The paramiko library is generally fast and easy to manage,
- Though users desiring Kerberos or Jump Host support may wish to switch to a native SSH binary such as OpenSSH by specifying the connection type in their playbooks

Connection Plugin

- OpenSSH is preferred if you are using a recent version, and also enables some features like Kerberos and jump hosts.
- There are also other connection types like accelerate mode, which must be bootstrapped over one of the SSH-based connection types but is very fast, and local mode, which acts on the local system.
- Users can also write their own connection plugins.

Ansible variable in Inventory file

Course and Labs

ansible_host
The name of the host to connect to, if different from the alias you wish to give to it.
ansible_port
The ssh port number, if not 22
ansible_user
The default ssh user name to use.
ansible_ssh_pass
The ssh password to use (never store this variable in plain text)
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 variable in Inventory file

ansible_ssh_common_args
This setting is always appended to the default command line for sftp, scp, and ssh. Useful to configure a ProxyCommand for a certain host (or group).
ansible_sftp_extra_args
This setting is always appended to the default sftp command line.
ansible_scp_extra_args
This setting is always appended to the default scp command line.
ansible_ssh_extra_args
This setting is always appended to the default ssh command line.
ansible_ssh_pipelining
Determines whether or not to use SSH pipelining. This can override the pipelining setting in ansible.cfg.

Ansible variable in Inventory file

- ansible_become
- Equivalent to ansible_sudo or ansible_su, allows to force privilege escalation
- ansible_become_method
- Allows to set privilege escalation method
- ansible_become_user
- Equivalent to ansible_sudo_user or ansible_su_user, allows to set the user you become through privilege escalation
- ansible_become_pass
- Equivalent to ansible_sudo_pass or ansible_su_pass, allows you to set the privilege escalation password (never store this variable in plain text

Non-SSH connection types

- The following non-SSH based connectors are available:
- **local**
- **docker**
- This connector deploys the playbook directly into Docker containers using the local Docker client. The following parameters are processed by this connector:
- This connector can be used to deploy the playbook to the control machine itself.

ansible_host

The name of the Docker container to connect to.

ansible_user

The user name to operate within the container. The user must exist inside the container.

ansible_become

If set to true the become_user will be used to operate within the container.

ansible_docker_extra_args

Could be a string with any additional arguments understood by Docker, which are not command specific. This parameter is mainly used to configure a remote Docker daemon to use.

AWS EC2 External Inventory Script

- If you use Amazon Web Services EC2,
- maintaining an inventory file might not be the best approach, because hosts may come and go over time, be managed by external applications, or you might even be using AWS autoscaling.
- For this reason, you can use the EC2 external inventory script.

AWS EC2 External Inventory Script

- You can use this script in one of two ways.
- The easiest is to use Ansible's -i command line option and specify the path to the script after marking it executable:
- ansible -i ec2.py -u ubuntu us-east-1d -m ping
- The second option is to copy the script to /etc/ansible/hosts and chmod +x it.
- You will also need to copy the ec2.ini file to /etc/ansible/ec2.ini.
- Then you can run ansible as you would normally.

AWS EC2 External Inventory Script

- To successfully make an API call to AWS, you will need to configure Boto (the Python interface to AWS).
- There are a variety of methods available, but the simplest is just to export two environment variables:
- export AWS_ACCESS_KEY_ID='AK123'
- export AWS_SECRET_ACCESS_KEY='abc123‘
- You can test the script by itself to make sure your config is correct:

- cd contrib/inventory
- ./ec2.py --lis

AWS EC2 External Inventory Script

- There are other config options in ec2.ini, including cache control and destination variables.
- By default, the ec2.ini file is configured for all Amazon cloud services, but you can comment out any features that aren't applicable.
- For example, if you don't have RDS or elasticache, you can set them to False

```
[ec2]
# To exclude RDS instances from the inventory, uncomment and set to False.
rds = False
# To exclude ElastiCache instances from the inventory, uncomment and set to False.
elasticache = False
```

AWS EC2 External Inventory Script

- Global
  - All instances are in group ec2.
- Instance ID
  - These are groups of one since instance IDs are unique. e.g. i-00112233 i-a1b1c1d1
- Region
  - A group of all instances in an AWS region. e.g. us-east-1 us-west-2
- Availability Zone
  - A group of all instances in an availability zone. e.g. us-east-1a us-east-1b

AWS EC2 External Inventory Script

- Security Group
  - Instances belong to one or more security groups.
- A group is created for each security group, with all characters except alphanumerics, converted to underscores (_).
- Each group is prefixed by security_group_.

- Currently, dashes (-) are also converted to underscores (_).
- You can change using the replace_dash_in_groups setting in ec2.ini (this has changed across several versions so check the ec2.ini for details). e.g. security_group_default security_group_webservers security_group_Pete_s_Fancy_Group

AWS EC2 External Inventory Script

- Tags
- Each instance can have a variety of key/value pairs associated with it called Tags.
- The most common tag key is ‘Name’, though anything is possible. Each key/value pair is its own group of instances, again with special characters converted to underscores, in the format tag_KEY_VALUE e.g. tag_Name_Web can be used as is tag_Name_redis-master-001 becomes tag_Name_redis_master_001 tag_aws_cloudformation_logical-id_WebServerGroup becomes tag_aws_cloudformation_logical_id_WebServerGroup

Playbooks

Playbooks

- Playbooks are Ansible's configuration, deployment, and orchestration language.
- They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process.
- If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material.

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 designed to be human-readable and are developed in a basic text language.

Playbooks

- Playbooks are a completely different way to use ansible than in adhoc task execution mode, and are particularly powerful.
- Simply put, playbooks are the basis for a really simple configuration management and multi-machine deployment system, unlike any that already exist, and one that is very well suited to deploying complex applications.

Playbooks

- Playbooks can declare configurations
- but they can also orchestrate steps of any manual ordered process, even as different steps must bounce back and forth between sets of machines in particular orders.
- They can launch tasks synchronously or asynchronously.
- Playbooks are more likely to be kept in source control and used to push out your configuration or assure the configurations of your remote systems are in spec.

Playbook Language

- Playbooks are expressed in YAML format
- 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

Playbook Language

- By composing a playbook of multiple ‘plays’,
- it is possible to orchestrate multi-machine deployments, running certain steps on all machines in the webservers group,
- then certain steps on the database server group, then more commands back on the webservers group, etc.
- “plays” are more or less a sports analogy.
- You can have quite a lot of plays that affect your systems to do different things.
- It’s not as if you were just defining one particular state or model, and you can run different plays at different times.

Web Server playbook

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

```
---
- hosts: webservers
  remote_user: root

  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf

- hosts: databases
  remote_user: root

  tasks:
  - name: ensure postgresql is at the latest version
    yum: name=postgresql state=latest
  - name: ensure that postgresql is started
    service: name=postgresql state=started
```

Hosts and Users

- For each play in a playbook, you get to choose which machines in your infrastructure to target and what remote user to complete the steps (called tasks) as.
- The hosts line is a list of one or more groups or host patterns, separated by colons
- The remote_user is just the name of the user account:

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
  become_user: postgres
```

```
---
- hosts: webservers
  remote_user: yourname
  become: yes
  become_method: su
```

Tasks list

- Each play contains a list of tasks.
- Tasks are executed in order, one at a time, against all machines matched by the host pattern, before moving on to the next task.
- It is important to understand that, within a play, all hosts are going to get the same task directives.
- It is the purpose of a play to map a selection of hosts to tasks.
- When running the playbook, which runs top to bottom, hosts with failed tasks are taken out of the rotation for the entire playbook.
- If things fail, simply correct the playbook file and rerun.

Modules

Modules

- Modules (also referred to as “task plugins” or “library plugins”) are the ones that do the actual work in ansible, they are what gets executed in each playbook task.
- But you can also run a single one using the ‘ansible’ command.

ansible webservers -m service -a "name=httpd state=started"

ansible webservers -m ping

ansible webservers -m command -a "/sbin/reboot -t now"

Core Modules

- These are modules that the core ansible team maintains and will always ship with ansible itself.
- They will also receive slightly higher priority for all requests than those in the “extras” repos.

Extras Modules

- These modules are currently shipped with Ansible, but might be shipped separately in the future.
- They are also mostly maintained by the community.
- Non-core modules are still fully usable, but may receive slightly lower response rates for issues and pull requests.
- Popular “extras” modules may be promoted to core modules over time.

Modules Overview

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

Handlers

- **Running Operations On Change of state**
- As we've mentioned, modules should be idempotent and can relay when they have made a change on the remote system.
- Playbooks recognize this and have a basic event system that can be used to respond to change.
- These 'notify' actions are triggered at the end of each block of tasks in a play, and will only be triggered once even if notified by multiple different tasks.
- For instance, multiple resources may indicate that apache needs to be restarted because they have changed a config file, but apache will only be bounced once to avoid unnecessary restarts.

Handlers

```
- name: template configuration file
  template: src=template.j2 dest=/etc/foo.conf
  notify:
     - restart memcached
     - restart apache
handlers:
    - name: restart memcached
      service: name=memcached state=restarted
    - name: restart apache
      service: name=apache state=restarted
```

Handlers

```
handlers:
    - name: restart memcached
      service: name=memcached state=restarted
      listen: "restart web services"
    - name: restart apache
      service: name=apache state=restarted
      listen: "restart web services"

tasks:
    - name: restart everything
      command: echo "this task will restart the web services"
      notify: "restart web services"
```

Variables

What Makes A Valid Variable Name

- Variable names should be letters, numbers, and underscores.
- Variables should always start with a letter.
- foo_port is a great variable.
- foo5 is fine too.
- foo-port, foo port, foo.port and 12 are not valid variable names.

Inventory Variable

- **Host Variables**

[atlanta]

host1 http_port=80 maxRequestsPerChild=808

host2 http_port=303 maxRequestsPerChild=909

Inventory Variable

- **Group Variables**

[atlanta]

host1

host2

[atlanta:vars]

ntp_server=ntp.atlanta.example.com

proxy=proxy.atlanta.example.com

Groups of Groups, and Group Variables

```
[atlanta]
host1
host2
[raleigh]
host2
host3
[southeast:children]
atlanta
raleigh
```

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

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

Splitting Out Host and Group Specific Data

- The preferred practice in Ansible is actually not to store variables in the main inventory file.
- If the host is named ‘foosball’, and in groups ‘raleigh’ and ‘webservers’, variables in YAML files at the following locations will be made available to the host:

/etc/ansible/group_vars/raleigh # can optionally end in '.yml', '.yaml', or '.json'

/etc/ansible/group_vars/webservers

/etc/ansible/host_vars/foosball

---

ntp_server: acme.example.org

database_server: storage.example.org

Information discovered from systems: Facts

- There are other places where variables can come from, but these are a type of variable that are discovered, not set by the user.
- Facts are information derived from speaking with your remote systems.
- An example of this might be the ip address of the remote host, or what the operating system is.
- To see what information is available, try the following:
- ansible hostname -m setup
- This will return a ginormous amount of variable data, which may look like this, as taken from Ansible 1.4 on a Ubuntu 12.04 system

Information discovered from systems: Facts

- In the above the model of the first harddrive may be referenced in a template or playbook as:

{{ ansible_devices.sda.model }}

Similarly, the hostname as the system reports it is:

{{ ansible_nodename }}

and the unqualified hostname shows the string before the first period(.):

{{ ansible_hostname }}

Facts are frequently used in conditionals (see Conditionals) and also in templates.

Registered Variables

- Another major use of variables is running a command and using the result of that command to save the result into a variable. Results will vary from module to module. Use of -v when executing playbooks will show possible values for the results.
- The value of a task being executed in ansible can be saved in a variable and used later. See some examples of this in the Conditionals chapter.
- While it's mentioned elsewhere in that document too, here's a quick syntax example:

```
- hosts: web_servers

  tasks:

    - shell: /usr/bin/foo

      register: foo_result

      ignore_errors: True

    - shell: /usr/bin/bar

      when: foo_result.rc == 5
```

- Registered variables are valid on the host the remainder of the playbook run, which is the same as the lifetime of "facts" in Ansible. Effectively registered variables are just like facts.

Passing Variables On The Command Line

- In addition to vars_prompt and vars_files, it is possible to send variables over the Ansible command line. This is particularly useful when writing a generic release playbook where you may want to pass in the version of the application to deploy:
- ansible-playbook release.yml --extra-vars "version=1.23.45 other_variable=foo"
- This is useful, for, among other things, setting the hosts group or the user for the playbook.
- Example:

```
---
- hosts: '{{ hosts }}'
  remote_user: '{{ user }}'
  tasks:
    - ...
```

- ansible-playbook release.yml --extra-vars "hosts=vipers user=starbuck"

Variable precedence

- role defaults
- 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 (always win precedence)