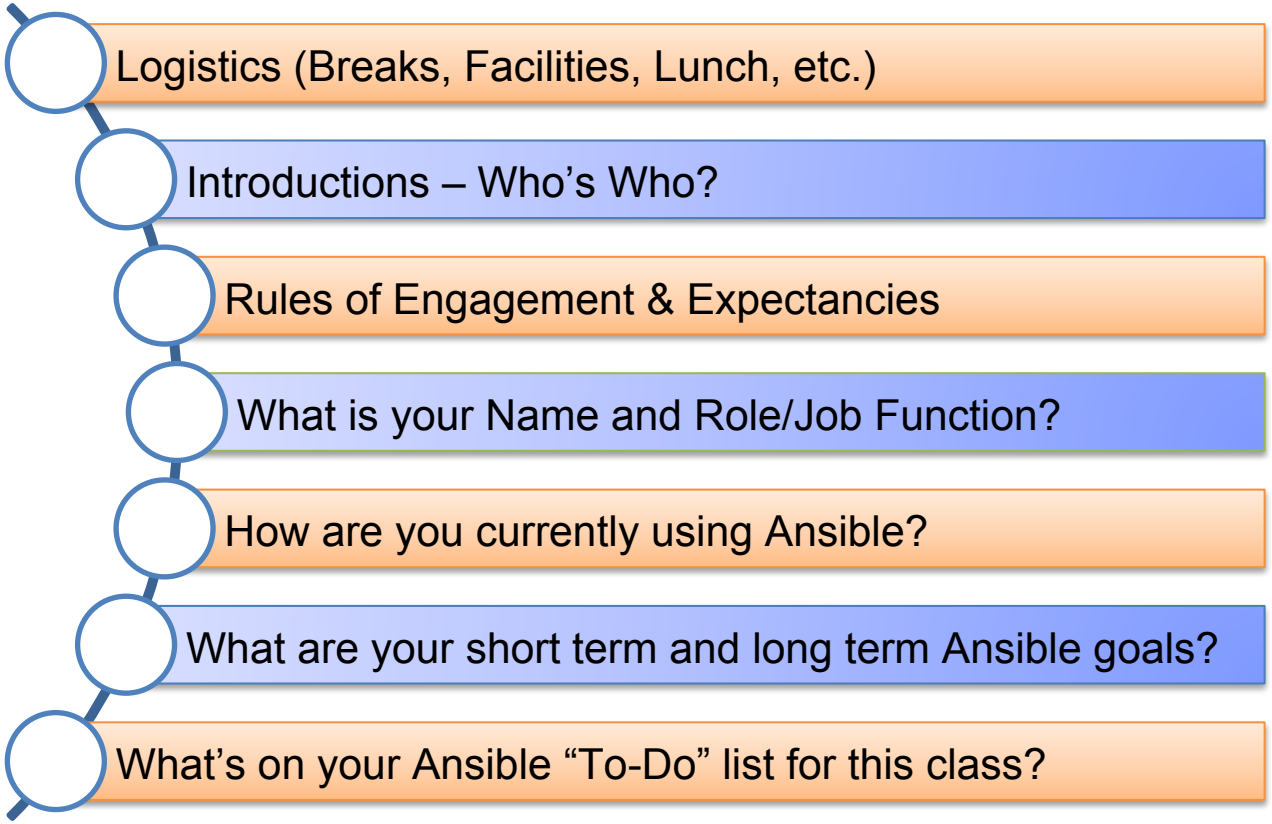
Ansible Configuration Management Boot Camp

Welcome!

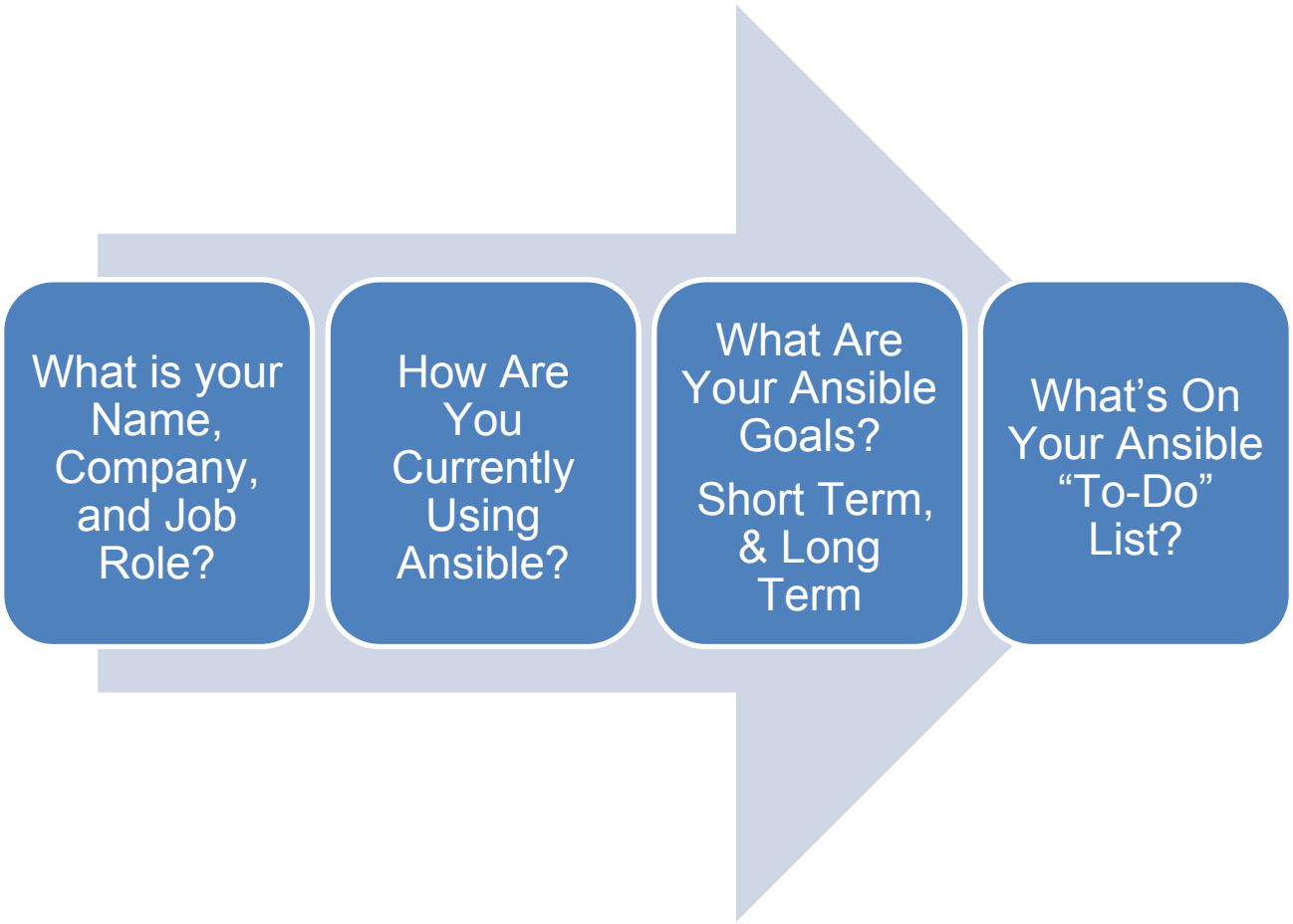

Logistics (Breaks, Facilities, Lunch, etc.)
Introductions – Who's Who?
Rules of Engagement & Expectancies
What is your Name and Role/Job Function?
How are you currently using Ansible?
What are your short term and long term Ansible goals?
What's on your Ansible "To-Do" list for this class?

Instructor Information

Who is your instructor?

A little about me…

Who's Who?


What is your Name, Company, and Job Role?
How Are You Currently Using Ansible?
What Are Your Ansible Goals?
Short Term, & Long Term
What's On Your Ansible "To-Do" List?

Who is this course for?

- **This workshop is well suited for:**
  - **IT Specialists who are moderately comfortable working in a Linux environment, and familiar with SSH use, commonly working as either:**
    - **Developers**
    - **Q/A or Testers**
    - **System Administrators/Engineers**
    - **DBAs**
    - **Or an IT in different job role who finds themselves in need of a simplified orchestration and configuration management program.**

What to expect from this class

- Hands-on experience from basic to real-world application.
- Fully participatory and Ansible centered.
- Conversations and coaching.
- An effort to bring what we learn here from the “real-world”, to “your world”.

About the lab environment…

- Due to the versatile nature and robust capabilities of Ansible, the lab environment used in this class can be built in a large number of configurations. Because of this, we have designed the labs and the necessary environments on which they will run, to have characteristics similar to what would be seen in the typical enterprise setting.

- Additionally, some steps taken, configurations made, or modules used in the exercises will function as designed, you will likely recognize slight differences between this environment, and the environment you are used to. Regardless of this, the core principles used here remain the same, and the learning objectives will ultimately be achieved.

Exercise Resource Prerequisites:

For the purposes of this class, in order to successfully complete the lab exercises, the only software requirement is a SSH client such as PuTTY, OR a modern web browser capable of supporting HTML5 and JavaScript.

If you do not have one of these, let your instructor know at once so a solution may be found.

First Step: Getting To Know Ansible:

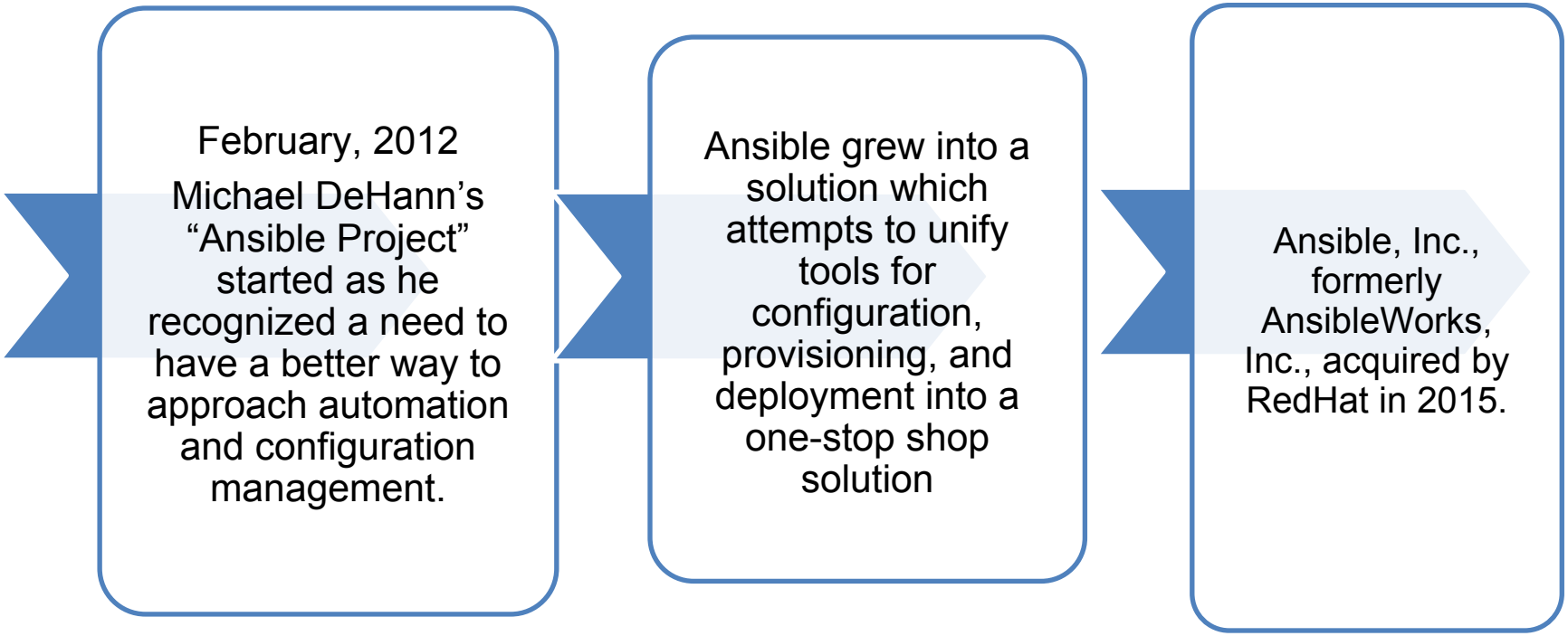
What we will cover in this section:

- ❑ The Origin of Ansible
- ❑ What *is* DevOps, and Why?
  - ❑ Continuous Integration/Continuous Delivery/Continuous Improvement
- ❑ Ansible's Place in the DevOps World
  - ❑ Simplified Automation With Ansible
  - ❑ Where is Ansible Used in the DevOps Team, and by Whom exactly?
  - ❑ How is Ansible Used by the *typical* DevOps Team?

What we will cover in this section (Continued):

- Why Configuration Management Anyway?
- Comparison of Ansible and Other CM Tools
- Strengths and weaknesses of Ansible
- Understanding the Importance of Idempotence
- Additional Resources and Reference Materials

The Origin of Ansible:



Level-set: DevOps – What is it, and Why?

The answer to these questions, are simultaneously very simple, and entirely more complicated.

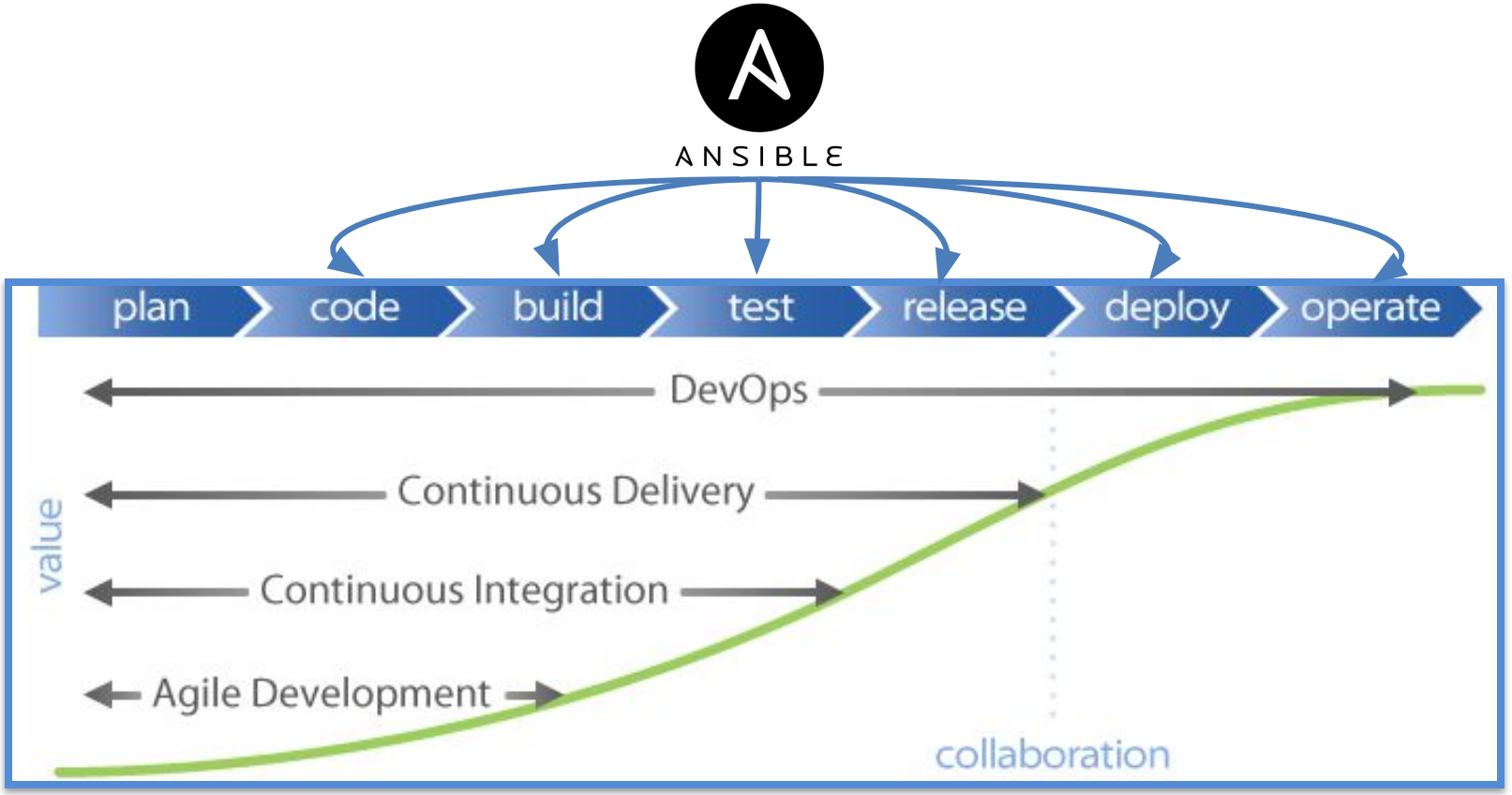
Simple Answer: DevOps can be considered the accumulation of industry best practices, processes, procedures, methods, and frameworks, all implemented in such a way as to ensure continuous integration of the organization's departments and teams, thereby empowering them to continuously deliver the organization's product (code, service, etc.), while also continuously improving every facet of this methodology at every stage possible.

(Not so simple though, is it?)

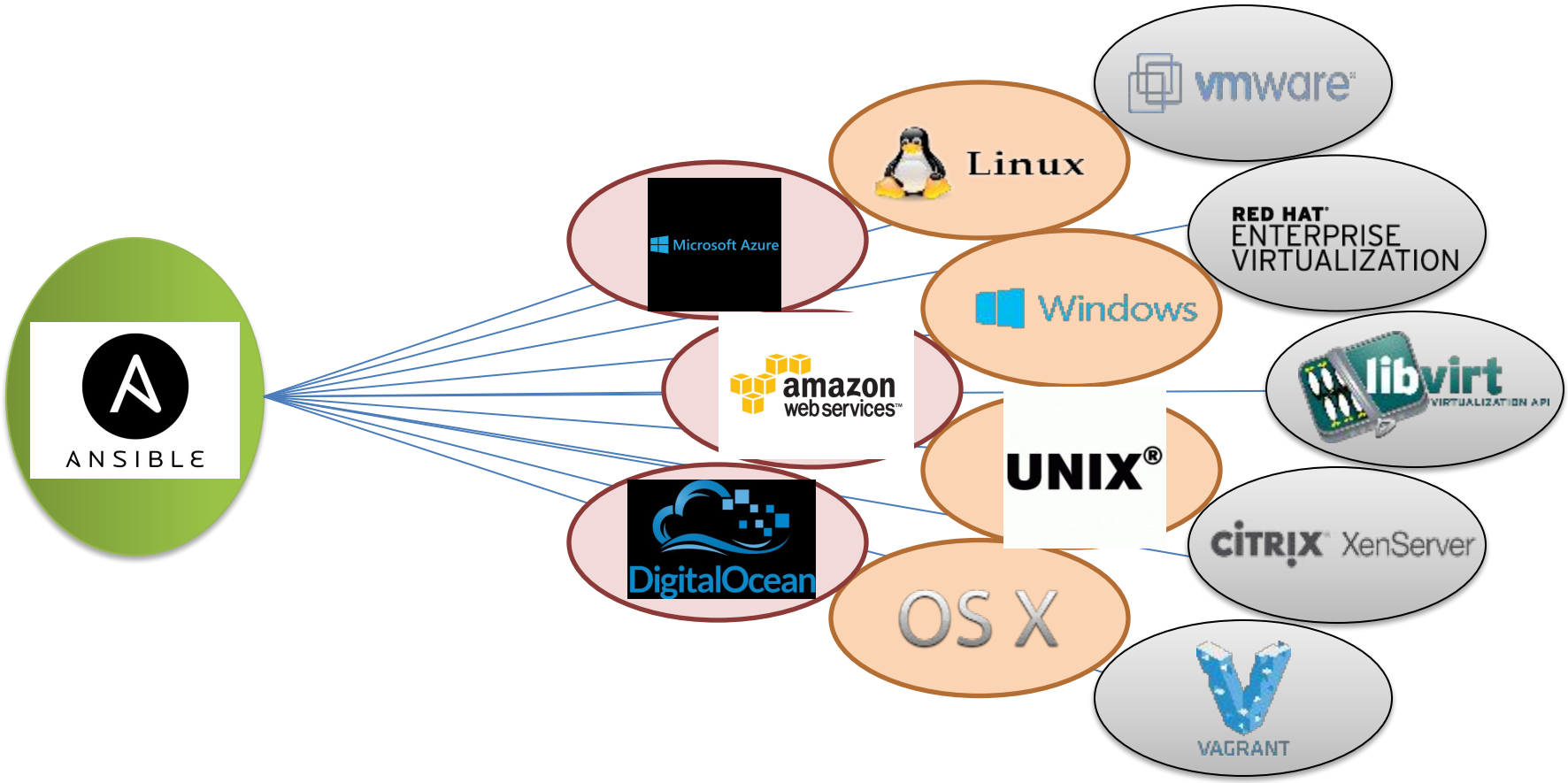
Ansible's Place In The DevOps World

- Simplified Automation (Deployment, Provisioning, Testing)
- Provisioning of Systems, Platforms, and Infrastructure
- Establishing Iterative Processes
- Configuration Management
- Managing Environments Consistently
- All in the efforts to achieve: Continuous Integration
  Continuous Delivery
  Continuous Improvement

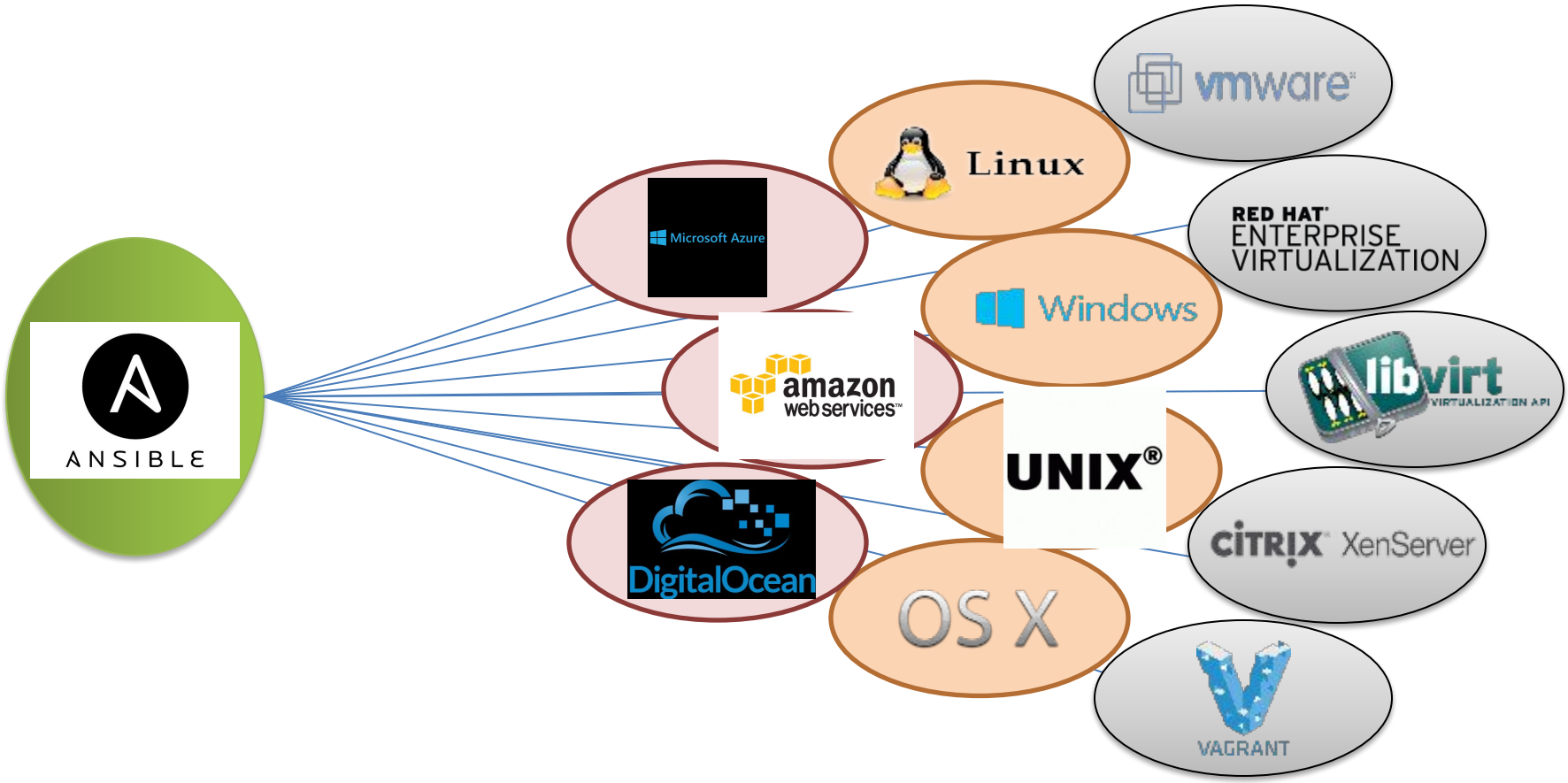
Continuous Integration
Continuous Delivery
Continuous Improvement



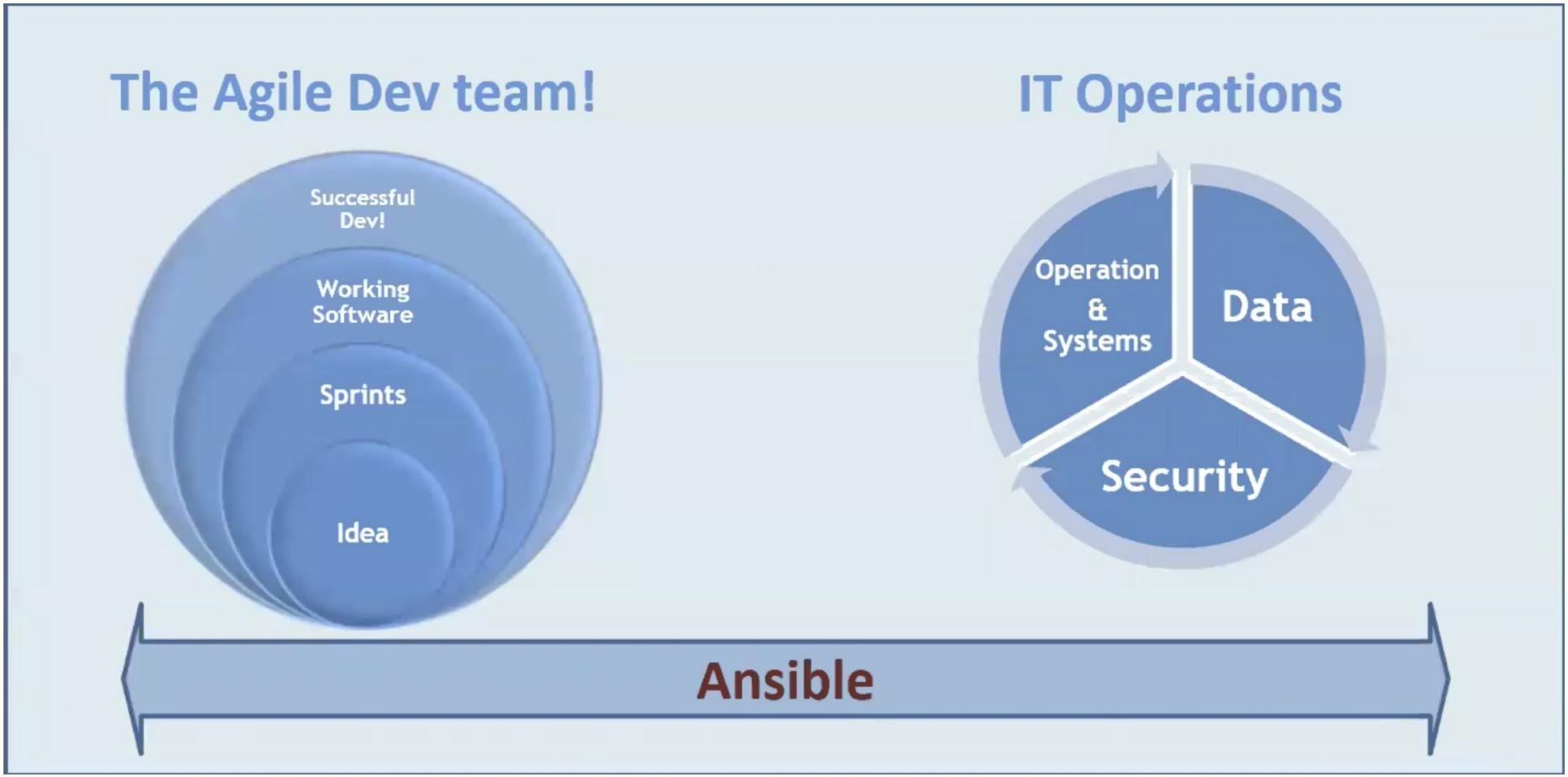
Ansible: Simplified Automation



Ansible: Simplified Automation



Ansible in The DevOps Team:



Key Term: Configuration Management

Ansible is often described as a configuration management tool, and is typically mentioned in the same breath as Chef, Puppet, and Salt.

Let's clarify the definition of "configuration management."

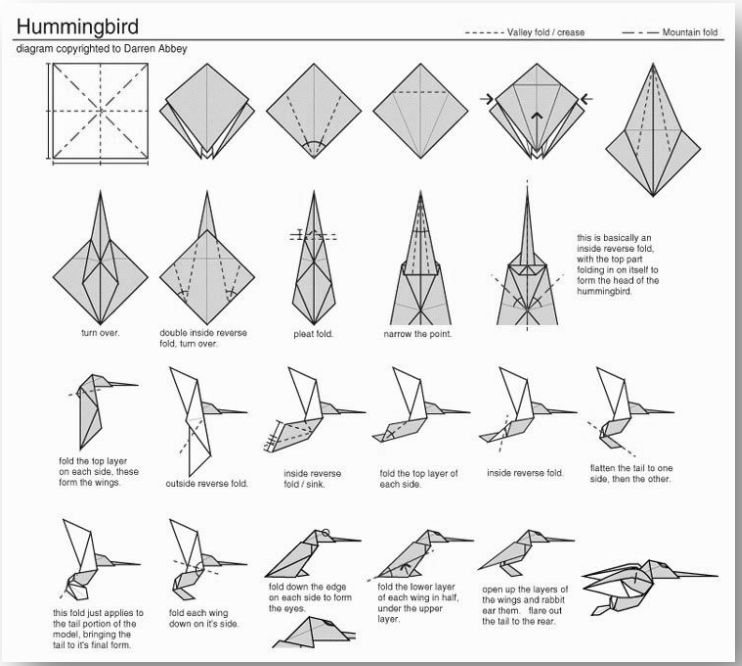
Key Term: Configuration Management

Ansible is often described as a configuration management tool, and is typically mentioned in the same breath as Chef, Puppet, and Salt.

Let's clarify the definition of "configuration management."

Why do we need configuration management?

Question:
How do you provision a new server?



Do you follow some documented instructions?

What is the current process you have in place?

Manual changes to servers, or any infrastructure, involves…

PEOPLE

Terrible at doing things repeatably...

Slow at doing things compared to machines...

Ansible Vs. Other CM Tools: Strengths

- Remarkably simple
- Agentless
- Secure (SSH)
- YAML (Playbooks)
- Written in Python
- Ansible Galaxy & Ansible Tower

Ansible Vs. Other CM Tools: Weaknesses

- No comprehensive GUI integrated by default. Alternative solutions are needed if a GUI is desired, such as Ansible Tower, Rundeck, or The Foreman.

- Currently, only minimal support/compatibility for Windows, as the Ansible Controller, however this is one area where Ansible developers are focusing their efforts.

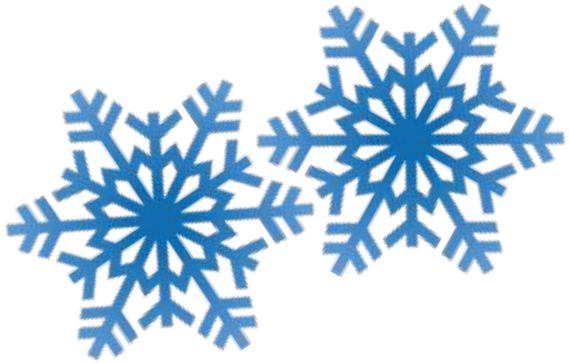
- Documentation supporting Ansible still leaves something to be desired. Especially for more complex configurations beyond novice to intermediate levels.

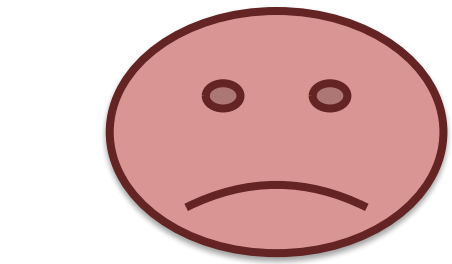
Key Term: Idempotence

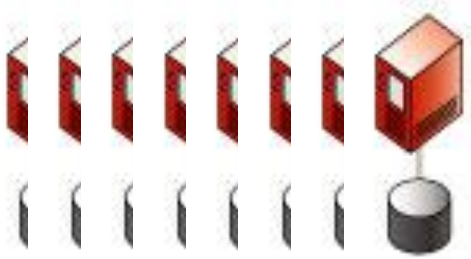
- When an action taken has the intended, and identical outcome, regardless of the number of times the action is taken, the result remains the same.

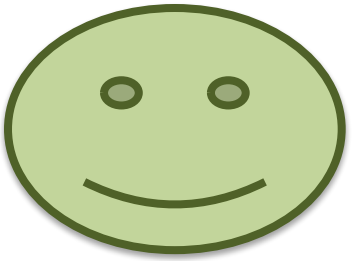
- Idempotence = Consistency + Predictability + Reliability

Managing environments consistently

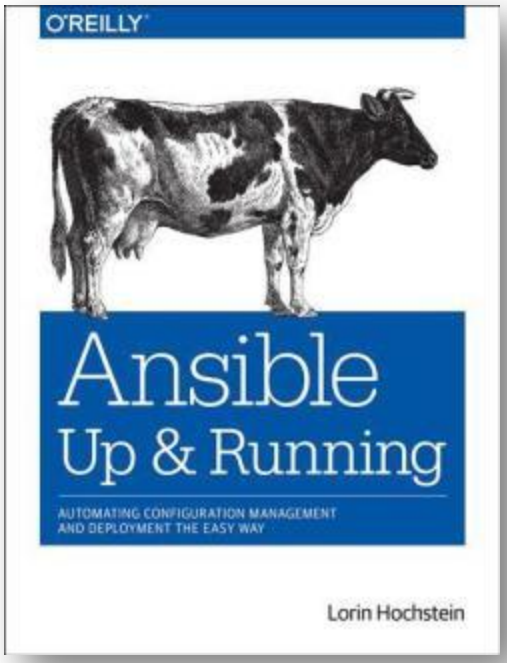








Additional Resources & Materials: Course Supplemental Text



The course includes the O'Reilly reference: *"Ansible: Up and Running."*

While we do not rely heavily on this book during the class, the text will provide you with a reference you can use after class as you begin to apply the practices you learn here in greater detail.

As we progress through the courseware, we will occasionally reference topics in this text, and other resources for either further reading, or to give you a place to refresh your memory after the course.

Second Step: Getting Started with Ansible:

What we will cover in this section:

- ❑ Ansible Basics
  - ❑ Terminology
  - ❑ Dependencies/Constraints
  - ❑ Components
  - ❑ The *typical* Ansible Environment
  - ❑ Best Practices
- ❑ YAML Fundamentals
  - ❑ Basic YAML Syntax
  - ❑ YAML in Use
- ❑ Authentication Basics
  - ❑ Authenticating with SSH Keys
- ❑ Vagrant Basics

Ansible Basics:
Common Ansible Terminology

Ansible Controller	**The host/device we execute our Ansible code from**
Inventory	A list of servers/devices in our environment Ansible will connect to
Playbook	The state definition we want Ansible to apply to our servers
Roles	A package of tasks, config files, templates and variables we can include from our playbooks

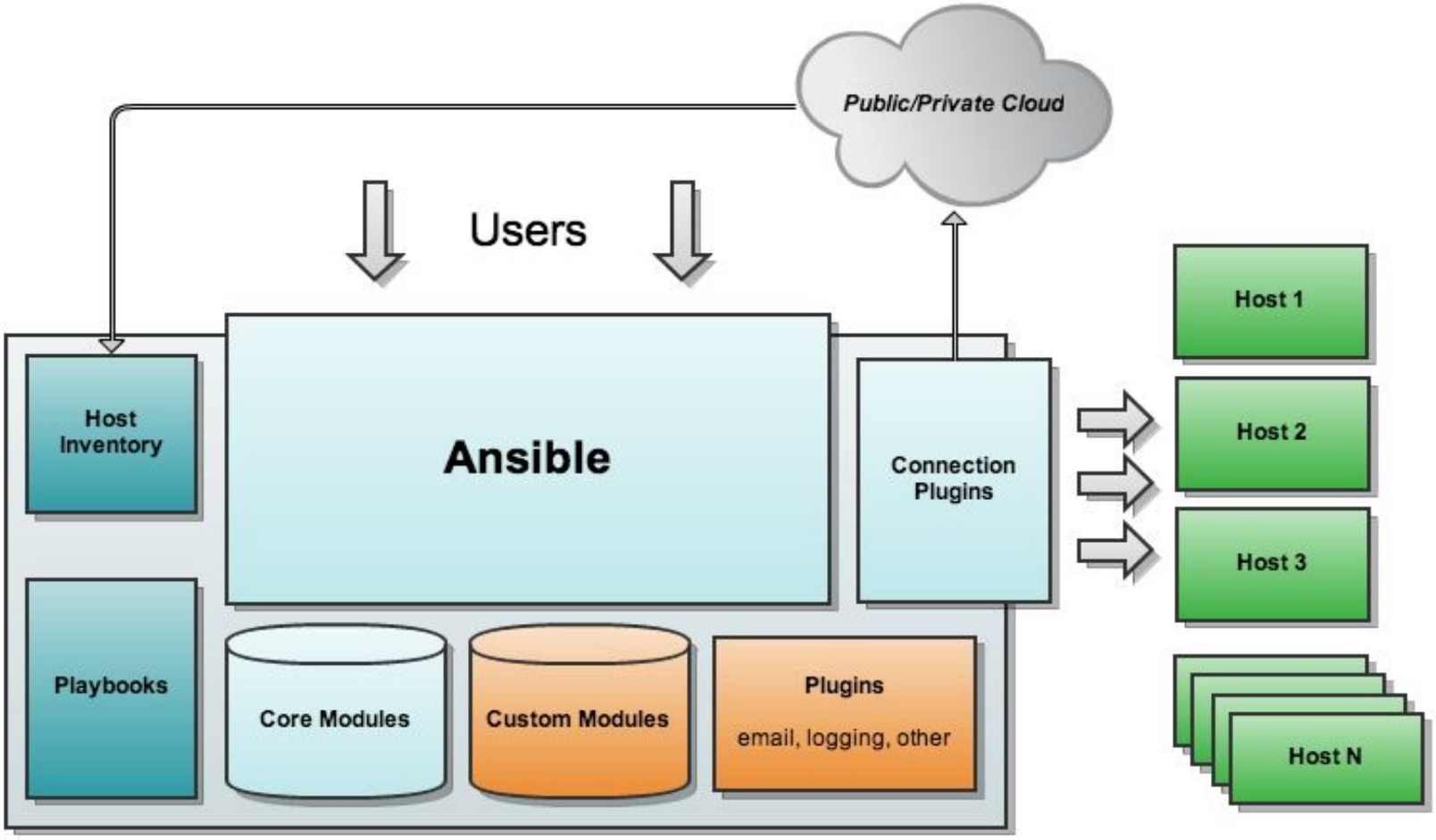
Ansible Basics:
Common Ansible Terminology

Ansible Controller	**The host/device we execute our Ansible code from**
Inventory	A list of servers/devices in our environment Ansible will connect to
Playbook	The state definition we want Ansible to apply to our servers
Roles	A package of tasks, config files, templates and variables we can include from our playbooks

Ansible Basics: Dependencies/Constraints

- Jumping between modules, it is important to know which module should be used in each case. For example, the “command” and “shell” modules are commonly used in places where the other should have been.
- Understanding the various input, output, configuration, and DSL formats seen and used in Ansible.
- The modules used and commands input using Ansible depend on thorough understanding of the target host’s own dependencies.
  - For example, depending on your target node’s operating system, one module may need to be used over another.
- Don’t forget about your network dependencies.

Ansible Basics: Components & Example of a *Typical* Ansible Environment



Ansible Basics:
Best Practices

When getting started, here are some best practices to keep in mind to ensure we have a smooth overall experience with Ansible.

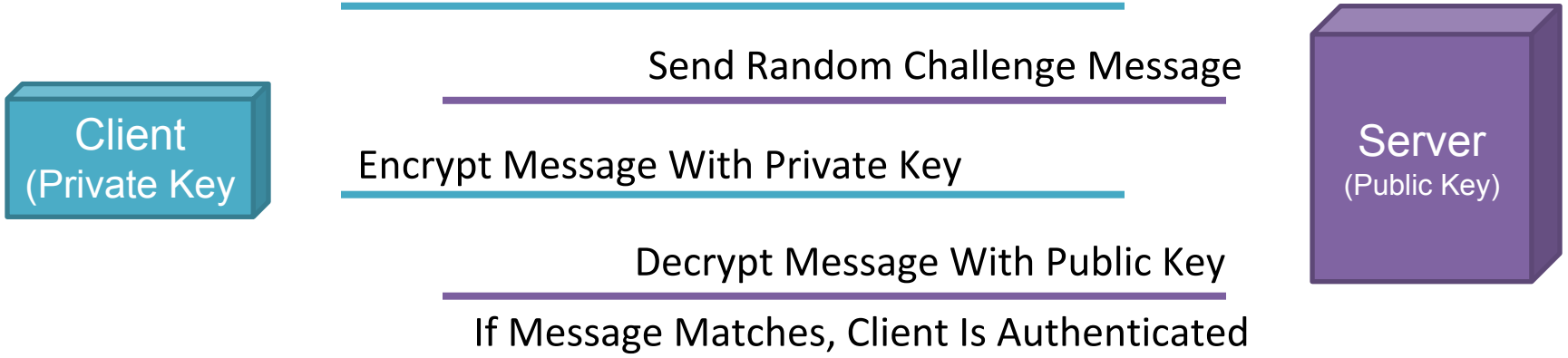
- ❑ Don’t underestimate the importance of properly configuring SSH.
- ❑ Assuming you are using Github, get Ansible to communicate with the Github server for you, and do so securely.
- ❑ Keep your vars separate from your secret vars.
- ❑ Separate your setup and deploy playbooks

Ansible Basics:
Best Practices

When getting started, here are some best practices to keep in mind to ensure we have a smooth overall experience with Ansible.

- ❑ Don’t underestimate the importance of properly configuring SSH.
- ❑ Assuming you are using Github, get Ansible to communicate with the Github server for you, and do so securely.
- ❑ Keep your vars separate from your secret vars.
- ❑ Separate your setup and deploy playbooks

Authentication Basics: SSH Keys



Creating SSH Keys:

On your local computer, generate a SSH key pair by typing:

```
ssh-keygen
```

Which should result in the following output:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/username/.ssh/id_rsa):
```

If you had previously generated an SSH key pair, you may see a prompt that looks like this:

```
/home/username/.ssh/id_rsa already exists.
Overwrite (y/n)?
```

Vagrant Basics

- **What is Vagrant?**
  - **A virtual machine environment managing and provisioning tool that can be used in conjunction with many other DevOps tools for creating, configuring, managing, and testing hosts, servers, and even entire infrastructures.**
- **Vagrant Features & Capabilities:**
  - Network interface management
  - Multi-machine management
  - Shared folder management:
  - Provisioning
- **Vagrant and Ansible:**
  - **Vagrant has easy integration with Ansible, and can work well together with Ansible as the provisioner. The processes to implement each work very similarly, and it is common to see them implemented together in a typical enterprise environment.**

Example of a Vagrantfile

```
1.# This is optimized for Vagrant 1.7 and above.
2.# Although versions 1.6.x should behave very similarly, it is recommended
3.# to upgrade instead of disabling the requirement below.
4.Vagrant.require_version ">= 1.7.0"
5.Vagrant.configure(2) do |config|
6.      config.vm.box = "ubuntu/trusty64"
7.# Disable the new default behavior introduced in Vagrant 1.7, to
8.# ensure that all Vagrant machines will use the same SSH key pair.
9.     config.ssh.insert_key = false
10.       config.vm.provision "ansible" do |ansible|
11.             ansible.verbose = "v“
12.             ansible.playbook = "playbook.yml"
13.  end
14.end
15.
```

Sub-section: Pre-exercise Level Set

Getting to Know Our Lab Environment

- ❑ What is Our Classroom Lab Setup Like?
- ❑ Virtualization Crash Course
- ❑ Overview of Jenkins
- ❑ Brief Example of Ansible in a Jenkins Environment
- ❑ Overview of LXC
- ❑ Example of Ansible provisioned resources and LXC

What is Our Classroom Lab Setup Like?

- Our lab environment is made up of a series of virtual machines which are unique to each student. Because of this, if something in your environment becomes corrupted or unusable, it won't impact anyone else in the class.

- When you first log to your student base machine using the credentials provided by your instructor, you are now in the Ansible Controller environment, this is where we will run most of our exercises from.

- We have provided the code for most of the exercises in the 'code' directory under your particular student account name. To see this, enter this on your Ansible controller after logging in:

```
$ls ./code
```

Virtualization Crash Course: 30,000 ft. view

- Virtualization leverages available software and hardware technologies to enable a single machine to operate as if it were multiple. How it does this can be very different depending on many factors including: The technologies or resources available, or even the overall goals to be met.

- A system can be virtualized at various levels, serving different purposes. Here are some of these levels and a few of their defining characteristics:

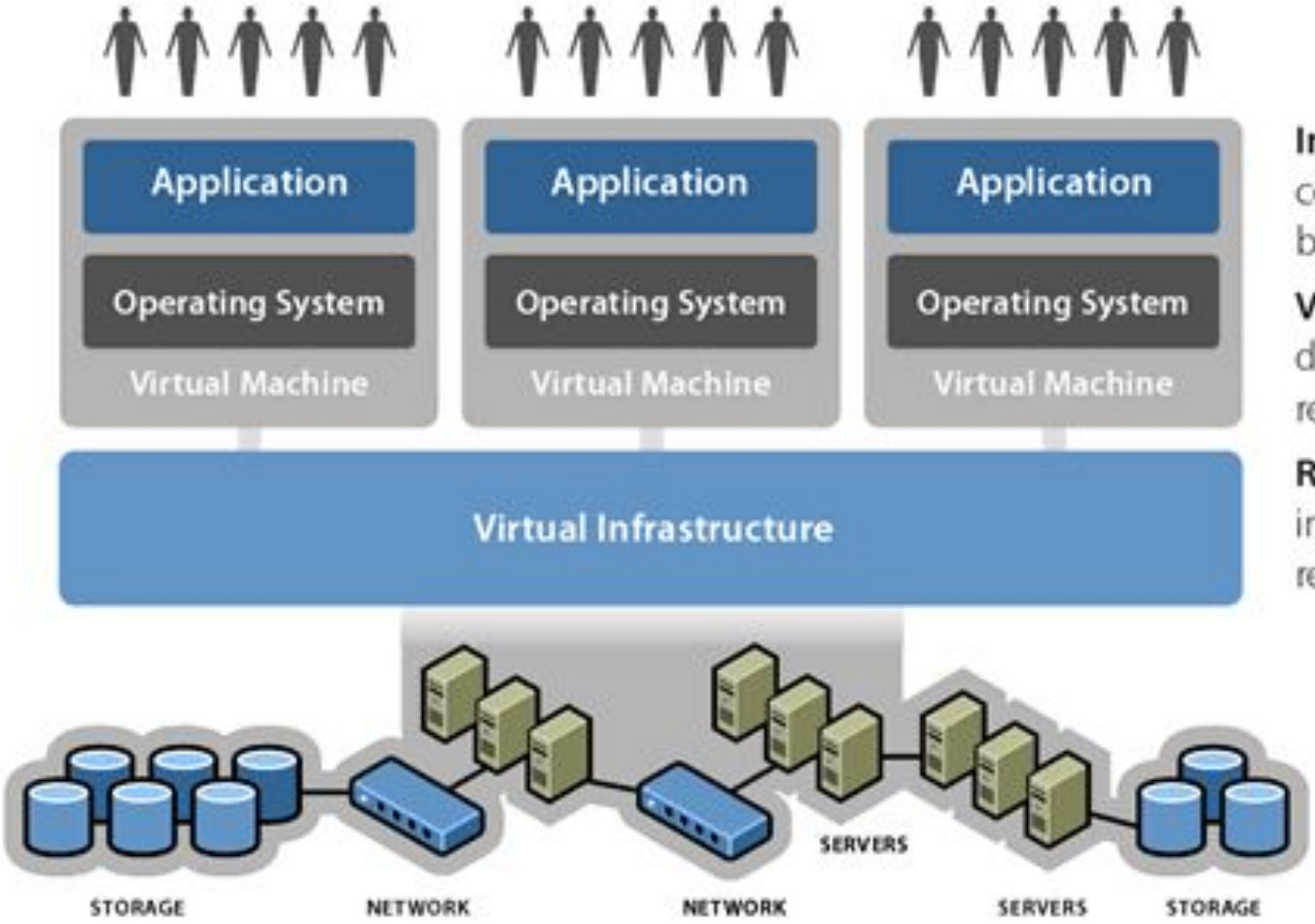
  - Hardware-level – Low performance, low elasticity, minimal tenancy

  - Platform/Application Level – High abstraction, high tenancy, poor security/containment.

  - OS-level – High tenancy, high performance, relatively efficient resource use, better security, better compartmentalization, potentially slow provisioning speed

  - Containerization – Similar in resource sharing to traditional virtualization, just at a much smaller scale. Does not require the entire OS install to be provisioned.

Infrastructure as a service



Infrastructure is what connects resources to your business.

Virtual Infrastructure is a dynamic mapping of your resources to your business.

Result: decreased costs and increased efficiencies and responsiveness

Overview of Jenkins

- Jenkins is a self-contained, Java-based Continuous Integration tool with an extensive library of plugins available in the Jenkins Update Center. This powerful open-source utility, when integrated with the right set of DevOps oriented tools, can be utilized to maximize the capabilities of nearly every aspect of your enterprise IT environment, both easily and efficiently.

- Ansible has an excellent relationship with Jenkins, and the tools are commonly deployed together.

- Combining Ansible with Jenkins, and containers, you can now provision from build to deploy, a fully functioning, testable, and repeatable Jenkins environment, all at the push of a button.

Well… perhaps just a few buttons need pushed.

Overview of LXC

- LXC, or Linux Containers, offers a unique approach to virtualization, where the resources allocated to the container, or virtual machine without the overhead of a fully virtualized system, maintain the same level of access to the kernel, file systems, and other host resources through kernel namespaces over OS level virtualization.
  - Key features include:
    - Isolated CPU, RAM, I/O, and networking resources through cgroups
    - Controlled network connection to both the virtual and physical networks
    - Solid Linux support community
    - Relatively easy to moderate learning curve.

Exercise:
Installation and Lab Environment Walkthrough

- **Exercise Objective A: Accessing the Lab Environment (Prerequisite)**

- **Exercise Objective C: Ansible Easy Install (Exercise 2)**

Ansible Core Fundamentals:

What we will cover in this section:

- ❑ Ansible Command Line
  - ❑ Basic Ansible Command Line Syntax
  - ❑ Installing Software/Packages
  - ❑ Managing Users, Permissions, and Services
- ❑ Ansible GUI
  - ❑ A Glance at GUIs for Ansible

Ansible Core Fundamentals:

What we will cover in this section (Continued):

- ❑Inventory Breakdown
- ❑Ansible Hosts
- ❑Ansible Tasks
- ❑Ansible Plays
- ❑Ansible Roles
- ❑Ansible Modules
- ❑Ansible Playbooks
- ❑Ansible Management,
- ❑Provisioning With Ansible
- ❑Best Practices

Introduction to:
The Ansible Command Line

Ansible comes with a number of command line tools which you will use to interact with the rest of your infrastructure. The most frequently used tools for most people are:

- ansible - a low level tool useful for running ad-hoc commands.
- ansible-playbook - the tool you use to kick off playbook runs across your environment.
- ansible-galaxy - the tool you use to search, download and update community roles as well as create your own.
- ansible-vault - used in the protection of sensitive information stored in your ansible repository.

Modules Overview

- Modules, which in many other settings may be more commonly referred to as library, or task plugins, are the core component on what is accomplishing the jobs we tell Ansible we want done.
- Modules take a `key=value` approach to arguments and are space delimited.

Ansible Command Line: Installing Software/Packages

There are Ansible modules available to manage packages for many platforms. For instance, there are modules available for yum and apt.

Examples of these in use:

#Ensure a package is installed, but don't update it:
$ ansible webservers -m yum -a "name=acme state=present"

#Ensure a package is installed to a specific version:
$ ansible webservers -m yum -a "name=acme-1.5 state=present"

#Ensure a package is at the latest version:
$ ansible webservers -m yum -a "name=acme state=latest"

#Ensure a package is not installed:
$ ansible webservers -m yum -a "name=acme state=absent"

Ansible Command Line:
Managing Users, Permissions, & Services

- The ‘user’ module allows easy creation and manipulation of existing user accounts, as well as removal of user accounts that may exist:

For Example:

$ ansible all -m user -a "name=foo password=<encrypted password here>"

$ ansible all -m user -a "name=foo state=absent“

Ensure a service is started on all webservers:

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

Alternatively, restart a service on all webservers:

$ ansible webservers -m service -a "name=httpd state=restarted"

Ensure a service is stopped:

$ ansible webservers -m service -a "name=httpd state=stopped"

Inventory Breakdown

We have talked a lot about performing actions on a target machine, but so far we have only been dealing with a single target.

- **Ansible uses an "inventory" to define all the target machines in your environment, weather they are physical servers, virtual machines, or instances running on a third party cloud provider such as Amazon AWS or DigitalOcean.**

- **An inventory in it's most basic form can just be a text file containing a list of hostnames, but we can be a little more descriptive in our inventory and split our machines in to logical groups.**

Inventory – Groups: Example

When performing configuration tasks, we typically want to perform actions on groups of hosts, rather than on an individual host.

Consider our example inventory which defines two web nodes, and one database node:

- **[vagrant@ansible vagrant]$ cat hosts**
- **node1 ansible_ssh_host=10.2.3.20**
- **node2 ansible_ssh_host=10.2.3.30**
- **node3 ansible_ssh_host=10.2.3.40**

- **[web]**
- **node1**
- **node2**

- **[database]**
- **node3**

– Ansible automatically defines a group called all which includes all of the hosts in the inventory.

What if I want, or *need* a GUI?

- For most operations, people working on the command line is part of the job. But if you have members of your organization that need the ability to execute tasks with Ansible, or you want to hide ansible behind a web interface, there are options to help with that.

Exercise:
Command Line Basics

- Exercise Objective A: Exercise 1 & 2
- Exercise Objective B: Exercise 3
- Exercise Objective C: Exercise 4

Recap From Yesterday:

- What is Config Management?
- What is Devops / CI - CD mean?
- Ansible Basics (Inventory, Playbook, Roles)
- Ansible Architecture
- Basics of ssh, yaml files and vagrant fundamentals
- Basics of Virtualization
- Exercises on basics of Ansible modules, and setup

Hosts, Tasks, Roles, and Plays

- Tasks – Playbooks exist to run tasks. Tasks combine an action (a module and its arguments) with a name and optionally some other keywords (like looping directives).
- Plays – A play is minimally a mapping between a set of hosts selected by a host specifier (usually chosen by groups but sometimes by hostname globs) and the tasks which run on those hosts to define the role that those systems will perform.
- Roles – Roles are units of organization in Ansible. Assigning a role to a group of hosts (or a set of groups, or host patterns, etc.) implies that they should implement a specific behavior.
- Hosts– A host is simply a remote machine that Ansible manages.

YAML Fundamentals: Basic Syntax

Since we have already constructed a few example playbooks, you may have noticed files with the .yml extension.

- **Ansible uses YAML as the default language in which we write our Playbooks**
- **"YAML is a human friendly data serialization standard for all programming languages."** (Source: www.yaml.org)
- **More simply put, and specifically for our use case here: YAML is a file format and is similar in intent to JSON, but generally easier for humans to read.**

YAML Fundamentals: Basic Syntax

– Start of file and comments

- A YAML file generally starts with three "-" characters on a line by itself
- "#" character = Comment

– Dictionaries and Lists

- A dictionary is represented in a simple key: value form (the colon must be followed by a space):
- Dictionaries and Lists can be in abbreviated form if desired, though usually done when more complex data structures are wanted.
- List members, or items will always begin with a "– "

– Strings

- Strings in Quotes = Optional
  - Name: this is a string
  - Name: "Or you can put it in quotes"

– Boolean

- Ansible is fairly flexible on Boolean values or yes / no, true / false.

– Values

- Values can span multiple lines using a " | " or a " > "
  - | includes new lines
  - > ignores new lines.

YAML Fundamentals: YAML in Use

- Lists (Delimited with hyphens):

 # A List of "Family Guy" Characters

family_guy:

  - Peter Griffin
  - Lois Griffin
  - Meg Griffin
  - Chris Griffin
  - Stewie Griffin
  - Brian Griffin

- Or inline:

family_guy: [Peter, Lois, Meg, Chris, Stewie, Brian]

- Dictionaries:

 # A Client Contact Record

analog:

  company: Analog Coffee

  address: 235 Summit Ave

  city: Seattle

  state: WA

  website: http://analogcoffee.com/

- Or Inline:

analog: {company: Analog Coffee, address: 235 Summit Ave, city: Seattle, state: WA, website: http://analogcoffee.com/}

YAML Fundamentals: YAML in Use

- Lists (Delimited with hyphens):

 # A List of "Family Guy" Characters

family_guy:

  - Peter Griffin
  - Lois Griffin
  - Meg Griffin
  - Chris Griffin
  - Stewie Griffin
  - Brian Griffin

- Or inline:

family_guy: [Peter, Lois, Meg, Chris, Stewie, Brian]

- Dictionaries:

 # A Client Contact Record

analog:

  company: Analog Coffee

  address: 235 Summit Ave

  city: Seattle

  state: WA

  website: http://analogcoffee.com/

- Or Inline:

analog: {company: Analog Coffee, address: 235 Summit Ave, city: Seattle, state: WA, website: http://analogcoffee.com/}

Introduction to Playbooks

- Playbooks are expressed in YAML format
- By composing a playbook of multiple ‘plays’, it is possible to orchestrate multi-machine deployments.
- 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.

```
- hosts: Kubernetes_minon     # Group name from inventory file
    User: centos              # Server authentication
    Sudo: true                # Server authentication
    Roles:
    - Java                    # Role installed on server
   - Memcached                # Role installed on server
```

Breaking down a task:

- Each task in our playbook starts with a name, and while the name value is optional, it helps others understand what your playbook is trying to do.
- Also, as we saw when we executed our playbook, the name gets displayed when the task is run, so it helps us know what Ansible is doing.
- These arguments tell the yum module that the "state" of the package named "epel-release" should be "present".

```
- name: Ensure the EPEL repository is present
  yum: name=epel-release state=present
```

Exercise:
Working with Playbooks

- **Exercise Objective A: Outlining and Creating A Simple Playbook (Exercise 5)**

How a role works

- All roles must have a tasks/main.yml file, this file gets executed first and describes how the rest of the role is defined.
- The files folder contains files that get copied to a target node, while the templates folder contains templates that get turned in to files on the target node.
- We will build our own role a bit later, but for now let's see what happens when we execute our role.

Proper Variable Use in Roles

Especially when security is a concern, ie. Ansible Vault is in play, we want to follow through with the official best practices from Ansible regarding Variables and Roles.

start with a **group_vars/** subdirectory named after the group.

In this directory, create two files named **vars** and **vault.**

In **vars,** define all of the variables needed secure, or otherwise.

Then, copy all of the sensitive variables over to the vault file and prefix these variables with ”vault_”.

Finally, adjust the variables in the vars file to point to the matching vault_ variables and ensure that the vault file is vault encrypted.

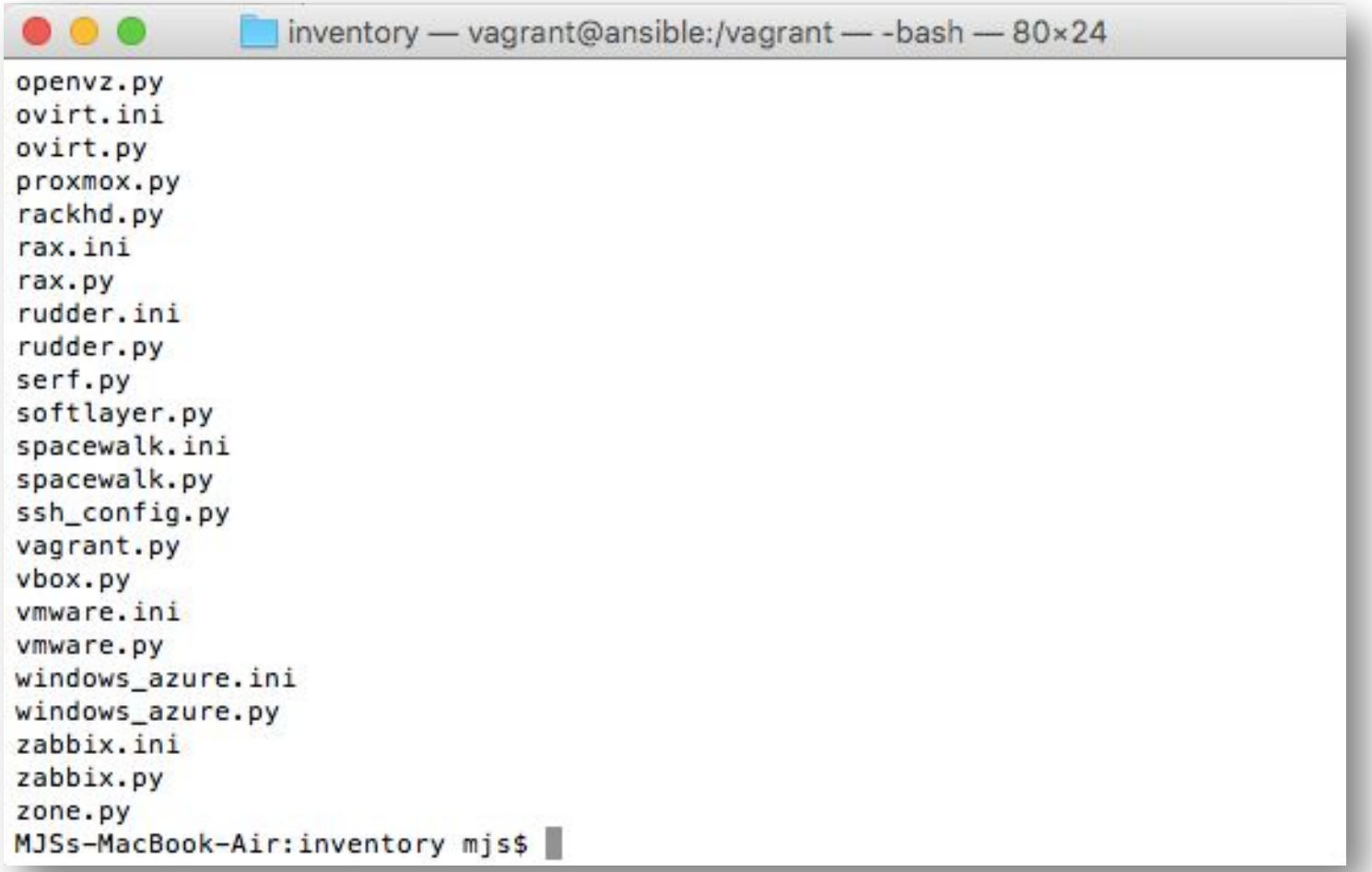
This best practice has no limit on the amount of variable and vault files or their names.

Setting some default variables

```
ntp_driftfile: /var/lib/ntp/drift
ntp_server: [0.centos.pool.ntp.org, 1.centos.pool.ntp.org, 2.centos.pool.ntp.org,
3.centos.pool.ntp.org]
ntp_restrict:
  - "restrict -4 default kod notrap nomodify nopeer noquery"
  - "restrict -6 default kod notrap nomodify nopeer noquery"
  - "restrict 127.0.0.1"

ntp_crypto: no
ntp_includefile: no
ntp_keys: no
ntp_trustedkey: no
ntp_requestkey: no
ntp_controlkey: no
ntp_statistics: no
ntp_broadcast: no
ntp_broadcastclient: no
ntp_multicastclient: no
```

Dynamic inventory



Final notes on dynamic inventory

Dynamic inventory scripts can save you a lot of time populating your inventory manually.

- **It is also possible to mix static and dynamic inventory.**
- **If you created a directory such as inventory/staging/ and put a static inventory file and a dynamic inventory script in that directory, when you give ansible the "-i inventory/staging" argument it will process both your static and dynamic inventory and combine the two together.**

Ansible Documentation

- The primary, and currently most detailed source of documentation on Ansible can be found in Ansible's official documentation site: **docs.ansible.com.**
  - Examples of content available include:
  - Installation and how to get started with Ansible
  - Playbooks
  - Inventory
  - A Module Index including Network Modules
  - A link to community provided information and how to contribute
  - Information regarding Ansible-Container
  - And much more.

Taking a Look at the Ansible Galaxy

- The website Ansible Galaxy, is a free site for finding, downloading, and sharing community developed Ansible roles. Downloading roles from Galaxy is a great way to jumpstart your automation projects.

- Access the Galaxy web site using GitHub OAuth, and to install roles use the 'ansible-galaxy' command line tool included in Ansible 1.4.2 and later.

  - Installing Roles with Ansible Galaxy:

    $ ansible-galaxy install username.rolename

  - Alternatively, specifying a specific directory where the downloaded role is placed:

    $ ansible-galaxy install username.role -p ~/Code/ansible_roles/

Best practices – directory layout

- As we identified previously, it is best to split up our Ansible tasks in to reusable components or roles.

- In order to keep our Ansible repository well organized, and so others we are collaborating with can find things easily, Ansible provides a suggested directory structure for your Ansible repository at:

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

- I have expanded a little on this directory layout based on using Ansible in real world environments and finding a structure that is comfortable for everyone to use.

Exercise:
Advanced Inventory and Variables

- **Exercise Objective A: Advanced Inventory & Variables (Exercise 6)**

Playbook Templates

Template Options:	
backup	serole
dest	setype
force	seuser
group	src
mode	unsafe_writes (added in 2.2)
owner	validate
selevel	

Jinja2 Basics and Syntax

- Ansible also uses Jinja2 to do variable substitution in playbooks. You will notice a similar {{ variable }} syntax while writing playbooks.

- You can use all of the Jinja2 features in your templates, however that level of granularity on a tool which is non-specific to Ansible is outside of the depth of this course, and therefore will not be covered in great detail.

Check out the Jinja2 Template Designer Documentation at http://jinja.pocoo.org/docs/dev/templates/ for more details.

Playbook Tags

- To Tag, or Not To Tag?
- There is a lot of healthy debate regarding the use of tags. Generally, though, industry best practices suggest to use them wisely, and not in excess.
  - Tags are good for troubleshooting/debugging as they allow you to run a portion of your playbook, without having to run it in its entirety.
  - Tags can be applied to playbooks, or roles.

Error and Exception Handling

- In some cases, you might find it necessary to have Ansible continue executing steps in a playbook after a failure occurs.

  To do this create a simple task like:

```
- name: do not see this as a failure
  command: /bin/false
  ignore_errors: yes
```

- We also may need to ensure that our handlers will continue to leave our hosts in the state in which we expect them to be, even when a task fails after handlers were notified to take an action. This can be done with in the following three ways:
  - In a play:
    ```
    force_handlers: True
    ```
  - Command-line option:
    ```
    -- force-handlers
    ```
  - In ansible.cfg:
    ```
    force_handlers = True
    ```

Task Ordering: Pre and post tasks

- A task is a single line item executed by the provisioning. If we look at our example below, we see a name:, and while the name is not required, it is highly recommended.

```
- name: This is my task
  apt: pkg=nano state=latest
```

- This is what will be displayed when the task is executed. And apt: is a built-in module in Ansible which relates to, and performs the tasks of the Debian-based distribution package manager: “apt”

Pre and post tasks Example

```
---
- hosts: appservers
  become: yes
  serial: 1

  pre_tasks:
   - name: Disable the backend server in
HAProxy.
      haproxy:
        state: disabled
       host: '{{ inventory_hostname }}'
       socket: /var/lib/haproxy/stats
       backend: habackend
     delegate_to: "{{ item }}"
     with_items: groups.lb

 tasks:
   - debug: msg="Deployment would be done
here."
```

```
post_tasks:
  - name: Wait for backend to come back up.
    wait_for:
      host: '{{ inventory_hostname }}'
      port: 8080
       state: started
      timeout: 60

 - name: Enable the backend server in HAProxy.
    haproxy:
      state: enabled
      host: '{{ inventory_hostname }}'
      socket: /var/lib/haproxy/stats
      backend: habackend
    delegate_to: "{{ item }}"
    with_items: groups.lb
```

Only run once

run_once and delegate_to are extremely helpful in scenarios like updating a database schema or clearing an application's cache, where you need a particular task to only run one time, on a particular server.

For example:

```
- command: /opt/webapps/app/scripts/upgrade-database-schema
  run_once: true
  delegate_to: db1.techtown.com
```

Handlers

- Handlers are one of the conditional forms that Ansible supports.
- A handler is similar to a task, but it only runs if it has been notified by a task.
- A task will fire the notification if Ansible recognizes that the task has changed the state of the target machine.
- A task notifies a handler by passing the handler's name as the argument.

Triggering our handler

- We can change our roles/ntp/defaults/main.yml file to use a different set of NTP servers.
- ntp_server: [0.ubuntu.pool.ntp.org, 1.ubuntu.pool.ntp.org, 2.ubuntu.pool.ntp.org, 3.ubuntu.pool.ntp.org, ntp.ubuntu.com]
- Run the playbook again and see if the handler runs when the config file changes.

CLASSROOM WORK

Exercise:
Advanced PLaybooks

- Exercise Objective A: Advanced Playbooks (Exercise 7)

Creating Your Playbook

- How are you currently planning your playbooks?
- What do your current playbooks look like?
- What problems do you currently have with your playbooks?

Ansible and Git

- If the opportunity to use Git is present, every single artifact should be located there in proper version control.
- Regardless of how amazing Ansible is at configuration management, orchestration, and automation, everything falls by the way-side if we do not use version control appropriately.

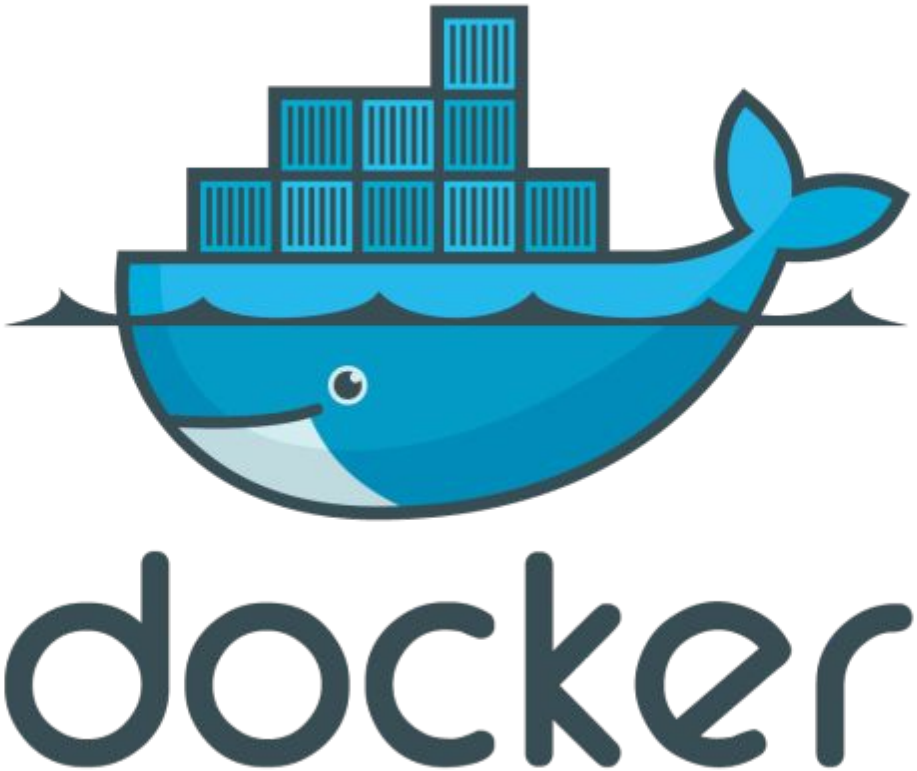
A Brief Look at Ansible and AWS

- Amazon provides three methods of accessing the AWS platform: web interface, command line interface and a REST API.

- If you don't currently have an Amazon AWS account, you should create one at https://aws.amazon.com/free/

- A large amount of support exists to aid you if you wish to learn more about how Ansible works with Amazon

Docker



What is Docker exactly?

The philosophy behind Docker is to be able to build, ship, and run applications anywhere. What this translates to in real world terms is that Docker allows you to package an application with all of its dependencies into a standardized unit for distribution.

Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server.

Docker Basics

Docker Image

- The basis of a Docker container

Docker Container

- The standard unit in which the application service resides

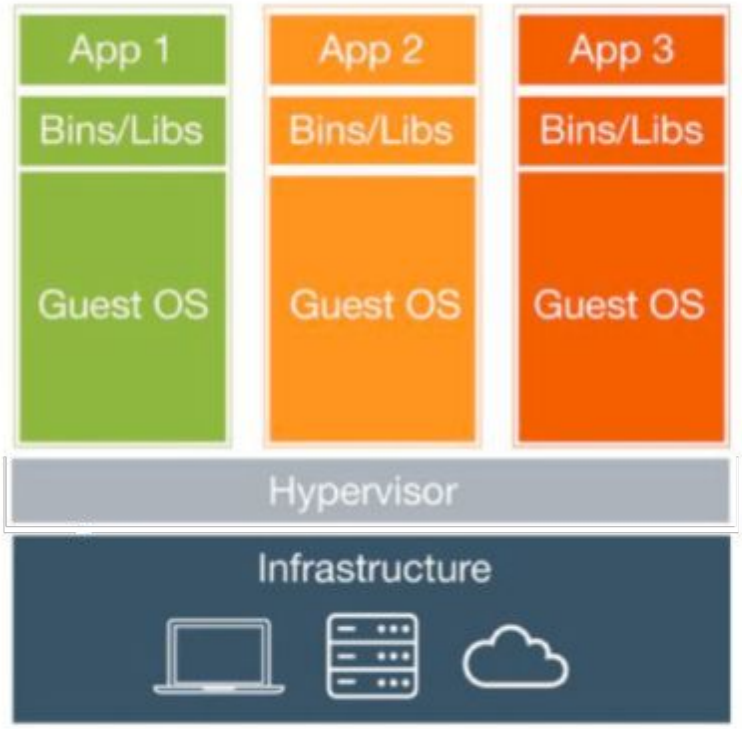
Docker Engine

- Creates, ships and runs Docker containers deployable on physical or virtual host locally, in a datacenter or cloud service provider

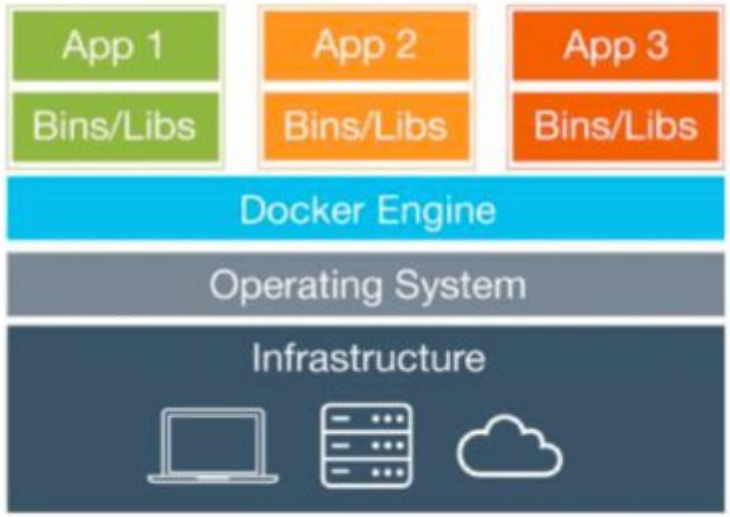
Docker Registry

- On-premises registry for image storing and collaboration

VMs vs. Containers



Virtual Machines



Containers

VMs vs. Containers

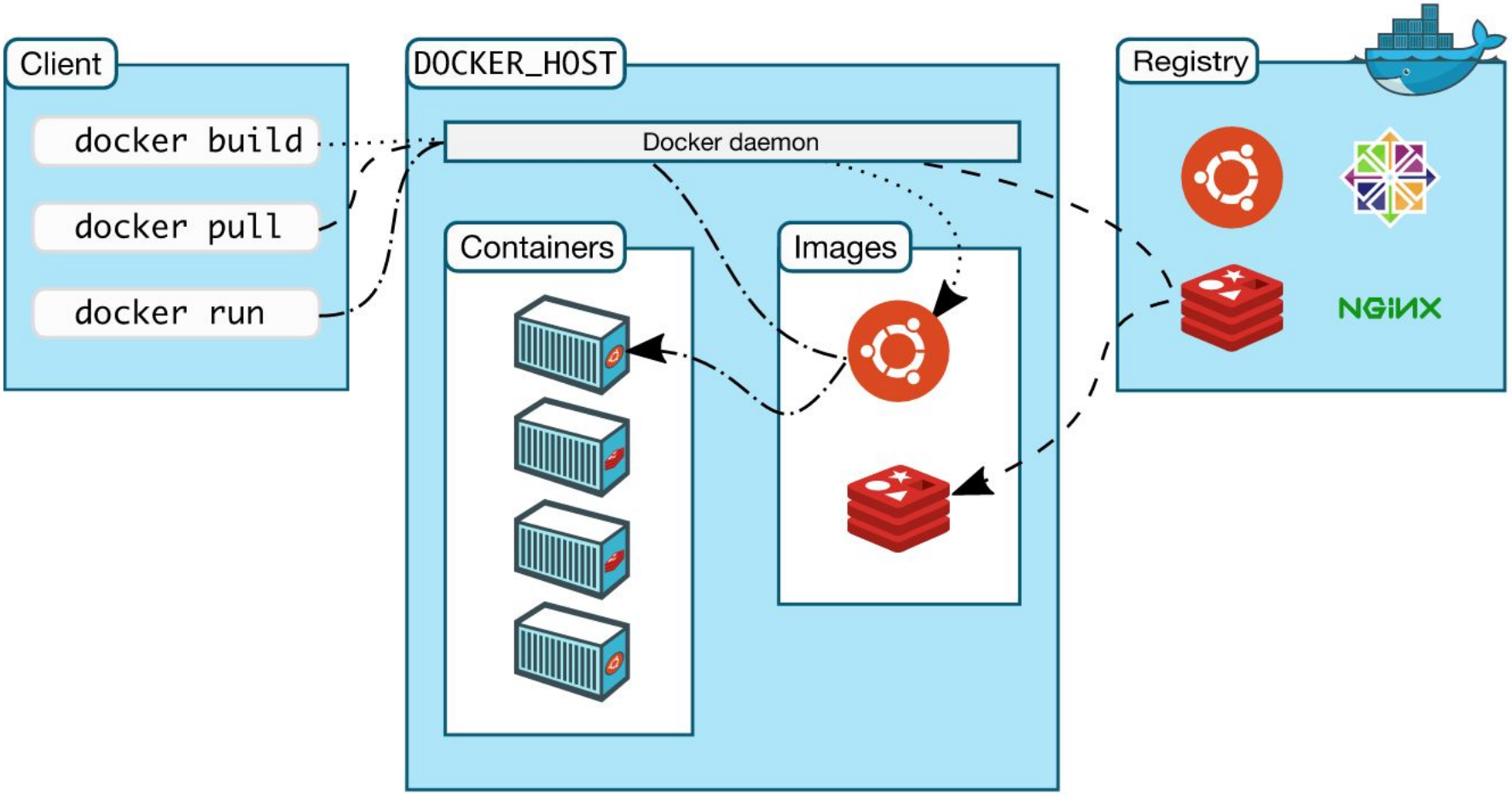
Virtual Machines (VMs)	Containers
Represents hardware-level virtualization	Represents operating system virtualization
Heavyweight	Lightweight
Slow provisioning	Real-time provisioning and scalability
Limited performance	Native performance
Fully isolated and therefore more secure (maybe)	Process-level isolation and therefore less secure (maybe)

Docker Application Life Cycle

Here's what the typical lifecycle of a Docker-based application looks like:

1. **Create Docker images on your local machine.**
2. **Push Docker images up from your local machine to the registry.**
3. **Pull Docker images down to your remote hosts from the registry.**
4. **Start up Docker containers on the remote hosts, passing in any configuration information to the containers on startup.**

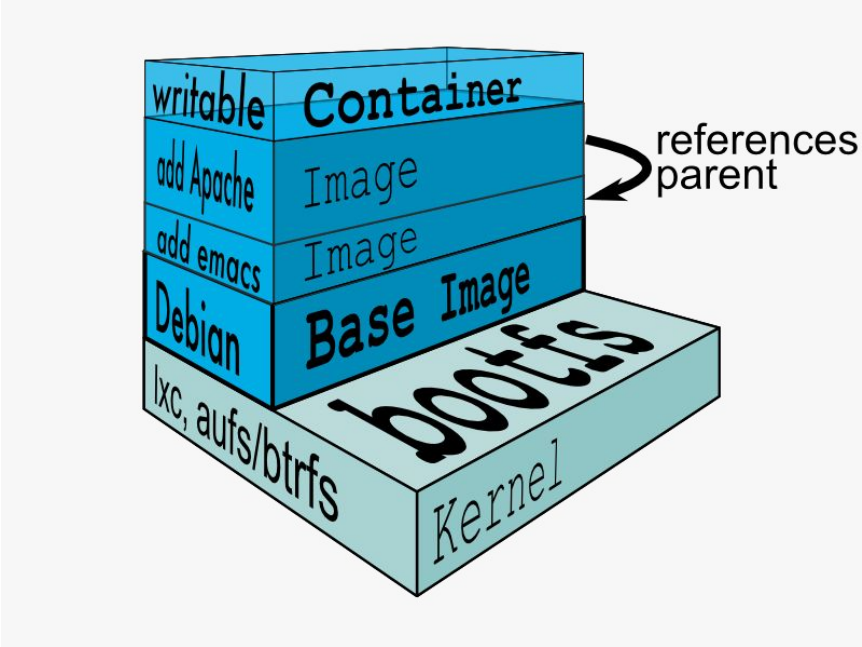
How is Docker Architected?



Docker Union File System

AuFS (AnotherUnionFS) is a multi-layered filesystem that implements union mount

- allows several filesystems or directories to be simultaneously mounted and visible through a single mount point
- appears to be one filesystem to the end user



Example Docker Layers

Docker Layer – Each Docker image is composed of a series of layers. Docker uses Union File Systems to combine these into a single image. The combination of these layers gives the illusion of a traditional file system. Only the top layer is writeable.

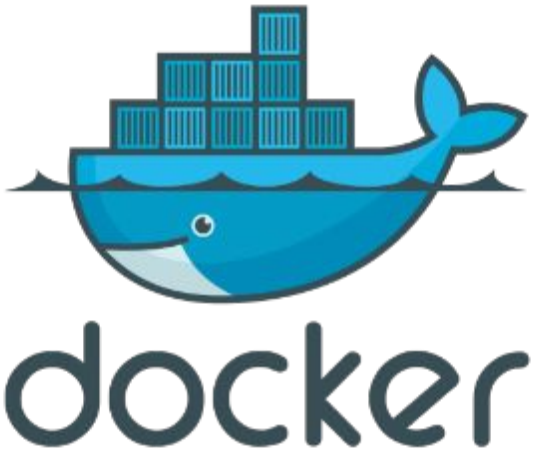
Where Containers are used?

- DevOps (Mostly in all types of Automation)
- Automated testing (Unit, acceptance and stress testing)
- CICD (if not tested on production-like environment, CICD will be risky)
- Microservices
- Less risky deployment architectures
- Chef kitchen test
- Not only used for testing but is used in production (along with container orchestration) in B2C and B2B organizations

Advantage of using Containers

- Developers testing their code in Prod environments
- Lesser hand-offs and drastically lesser waiting time (to get new environment setup)
  - –Reduce technical debt
- Overall system resilience - Service failure does not kill the application & may be invisible to users
- Scalability
  - – Seamless replication
- High Availability
- Less risky patterns for rolling updates available
- Prevent server sprawl and Jenga infrastructure

A Brief Look at Ansible and Docker



Deploying with Ansible

Deploying from git

Rolling updates

- Ansible by default will fork five SSH connections at a time, which means it will execute the deploy tasks on five of your servers at the same time, for zero downtime deployments this might not be ideal. We can tell Ansible to execute the deployment one server at a time by putting this at the top of our deploy.yml playbook:

```
---
- hosts: appservers
  gather_facts: no
  become: yes
  serial: 1
```

Deployment Strategies

- Recreate
- Ramped (Similar to Rolling Updates)
- Blue - Green Deployment
- Canary Deployment
- A/B Testing
- Shadow Deployments

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps
https://container-solutions.com/kubernetes-deployment-strategies/
https://github.com/ContainerSolutions/k8s-deployment-strategies
https://container-solutions.com/deployment-strategies/

Notifications

- Another important aspect of a successful deployment is communication, you can use one of the many built-in Ansible notification modules to share the deployment's progress via chat or email.

Re-visiting our Directory layout

- As we identified previously, it is best to split up our Ansible tasks in to reusable components or roles.

- In order to keep our Ansible repository well organized, and so others we are collaborating with can find things easily, Ansible provides a suggested directory structure for your Ansible repository at:

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

- I have expanded a little on this directory layout based on using Ansible in real world environments and finding a structure that is comfortable for everyone to use.

Templating with Jinja2

Jinja2 is a python library which allows templates to be expanded in various ways, such as substituting a variable expression into parts of that template.

Key Features :

- Sandboxed execution mode.
- powerful automatic HTML escaping system for cross site scripting prevention.
- Template inheritance makes it possible to use the same or a similar layout for all templates.
- High performance with just in time compilation to Python bytecode.
- Optional ahead-of-time compilation
- Easy to debug
- Configurable syntax.
- Template designer helpers.

Tracking State

- When you run ansible-playbook, Ansible outputs status information for each task it executes.
- You will notice that the status for some of the tasks that run is changed, and the status for some others is ok.
  - Changed relates to whether the state that Ansible detected the target to be in, prior to executing the tasks, was modified to meet the parameters we set out in our task.
  - Ok status relates to the same verification which Ansible performs, however the initial state detected, met the desired state requirements.

Testing, CI/CD/CI, and Final Steps

We will cover:

- ❑ Testing, Validating, & Debugging
- ❑ Continuous Integration, Continuous Delivery & Continuous Improvement
- ❑ Ansible in “The Wild”
- ❑ Final Step: Revisiting Your Ansible Needs

Testing, Validating, & Debugging

- ❑ Common Issues and Finding the Root Cause
- ❑ Basic unit, integration, and functional testing with Ansible
- ❑ Is My Sin-Tax Correct?
- ❑ Test “All The Things”
- ❑ Debugging & Best Practices
- ❑ Introduction to ServerSpec and RoleSpec for testing

Common Issues & Finding the Root Cause

- The most common problems seen with Ansible usually surround:
  - SSH and Authentication
    - “Your key is not known”
    - Your key is missing from authorized_keys
    - SSH agent not running
    - known_hosts conflicts
    - Sshdconfig
  - Sudo Failures

While the above are quite common, the most common issue brought up usually isn’t a result of a problem with Ansible, rather a problem of other tools not playing nicely that we have Ansible rely upon.

Test All The Things:
Basic Testing with Ansible

- ❑ Unit Testing
- ❑ Integration Testing
- ❑ Functional Testing

```
tasks:
 - service: name=foo state=started enabled=yes
```

Other Useful Modules for Testing:

Check Mode (**- check**) – Example: Drift Testing

- wait_for: host={{ inventory_hostname }} port=22 - Example: Port status check.
delegate_to: localhost

- script - Just like scripting in any language.

https://blog.theodo.fr/2015/10/best-practices-to-build-great-ansible-playbooks/

Production considerations - Is My Sin-Tax Correct?

ServerSpec

- ServerSpec is a testing tool written in Ruby, it is very popular with teams that use Chef and Puppet, but it works equally as well with Ansible.

RoleSpec

- Before we wrap up with testing, another testing framework worth bringing to your attention is called RoleSpec.
- It is relatively new but was built specifically for Ansible.

Continuous Integration → Continuous Delivery → Continuous Improvement

- ❑ Continuous Integration & Ansible:
- ❑ Continuous Delivery & Ansible:
- ❑ Continuous Improvement & Ansible:

Orchestrating with Ansible

- Ultimately, we want to take our workflows from where they are right now, to hopefully something like this (if not already):

- Use the same playbook all the time with embedded tests in development
- Use the playbook to deploy to a staging environment (with the same playbooks) that simulates production
- Run an integration test battery written by your QA team against staging
- Deploy to production, with the same integrated tests.

- Then eventually, to a workflow that gets us to a level of Continuous Deployment, which might look like this:

- Write and use automation to deploy local development VMs
- Have a CI system like Jenkins deploy to a staging environment on every code change
- The deploy job calls testing scripts to pass/fail a build on every deploy
- If the deploy job succeeds, it runs the same deploy playbook against production inventory

Automate All the Things

- What can be automated with Ansible?
  - System & Network Configuration Management
  - Testing/Auditing
  - Integration
  - Monitoring
  - Deployments small and/or large scale
  - Provisioning
  - Maintenance/Patching
  - Itself… yes, Ansible can automate Ansible!

Obtaining Zero Downtime

The more we achieve automation nirvana, the closer we can get to reaching zero downtime.

Until 2016, **80%** of all mission-critical IT service outages were due to **PEOPLE** and process errors, with more than **50%** of those outages caused by **change/ configuration/release integration and handoff** issues.

– source: Gartner RAS Core research, 2010

Ansible in The Wild

- Let's discuss how Ansible is being used in the real world:
  - Ansible at Cogapp
  - Ansible at HootSuite
  - Ansible at NASA
- Additional Ansible References & Resources

Ansible in The Wild: Cogapp

Cogapp – An Ansible Case Study:

Provisioning and Content Development:

https://www.ansible.com/hubfs/Whitepapers__Case_Studies/Cogapp_Case_Study.pdf?t=1472592588864

Ansible in The Wild: HootSuite

HootSuite – An Ansible Case Study:

Orchestration in a SaaS infrastructure:

https://cdn2.hubspot.net/hub/330046/file-480366621-pdf/pdf_content/Hoot_Suite_Case_Study.pdf?t=1472592588864

Ansible in The Wild: NASA

NASA – An Ansible Case Study:

https://www.ansible.com/hs-fs/hub/330046/file-1649288715-pdf/Whitepapers__Case_Studies/nasa_ansible_case_study.pdf?t=1472592588864

Additional Info: Capistrano-style deployments

The role can be found at
https://galaxy.ansible.com/f500/project_deploy/

There is a very comprehensive presentation explaining its use at
http://www.slideshare.net/ramondelafuente/ansible-projectdeploy

Additional Info: Blue-Green deployments

Ansible has a great blog post that goes into further details about this type of deployment:

https://www.ansible.com/blog/immutable-systems

Preparing for Ansible back at work

If time permits, let's revisit some questions we posed at the beginning of this workshop:

How are you using Ansible?

What are your Ansible Goals?

Short Term?

Long Term?

What is on your "To Do" list, now that we have seen what Ansible can do?

Course Evaluation

Please take a moment to complete your course evaluation

- Go to http://www.metricsthatmatter.com/ASPE
- Choose the class name listed with the date of the class and your instructor's name
- Fill out and submit the form
- It is available for 10 calendar days

Thank You!