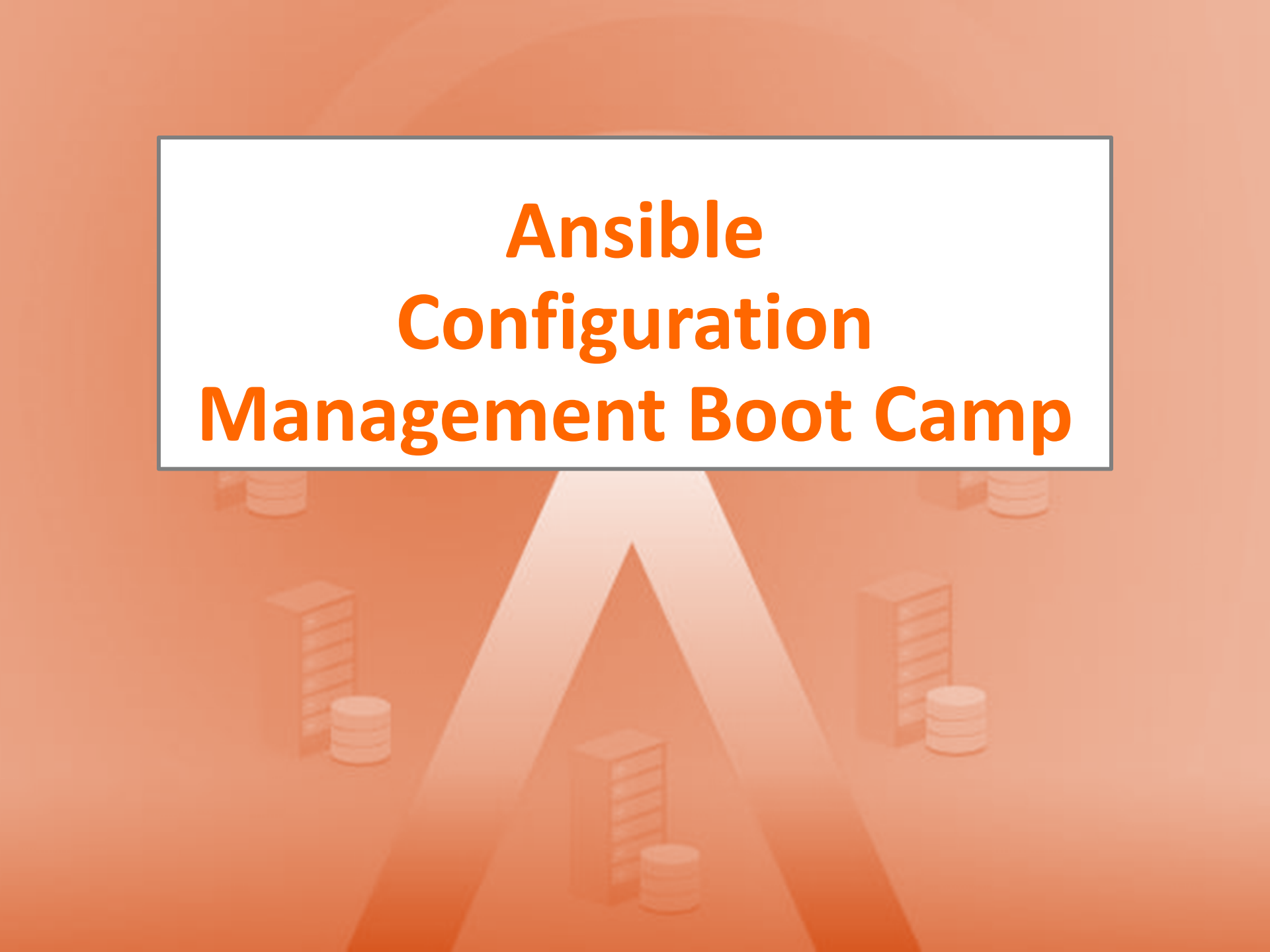
Ansible Configuration Management Boot Camp



Class Logistics

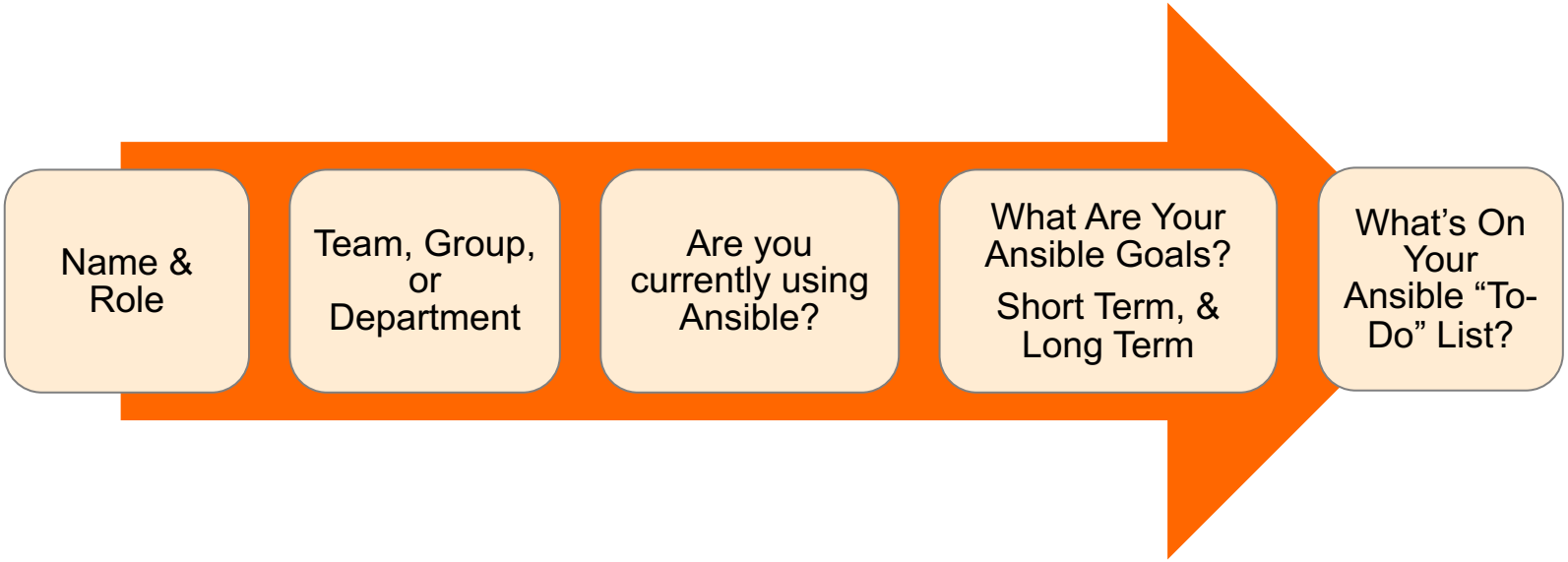
- Duration
- Start - Stop
- Lunch Break
- AM & PM Breaks

Instructor Information

Who is your instructor?

A little about me…

Introductions



Discussion

Who is this course for?

This workshop is well suited for:

- **IT Specialists who are moderately comfortable working in a Linux environment**
- **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.

Getting To Know Ansible

Module 1

This Section Will Cover:

- 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?

This Section Will Cover (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:

February, 2012

Michael DeHann's "Ansible Project" started as he recognized a need to have a better way to approach automation and configuration management.

Ansible grew into a solution which attempts to unify tools for configuration, provisioning, and deployment into a one-stop shop solution

Ansible, Inc., formerly AnsibleWorks, Inc., acquired by RedHat in 2015.

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

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

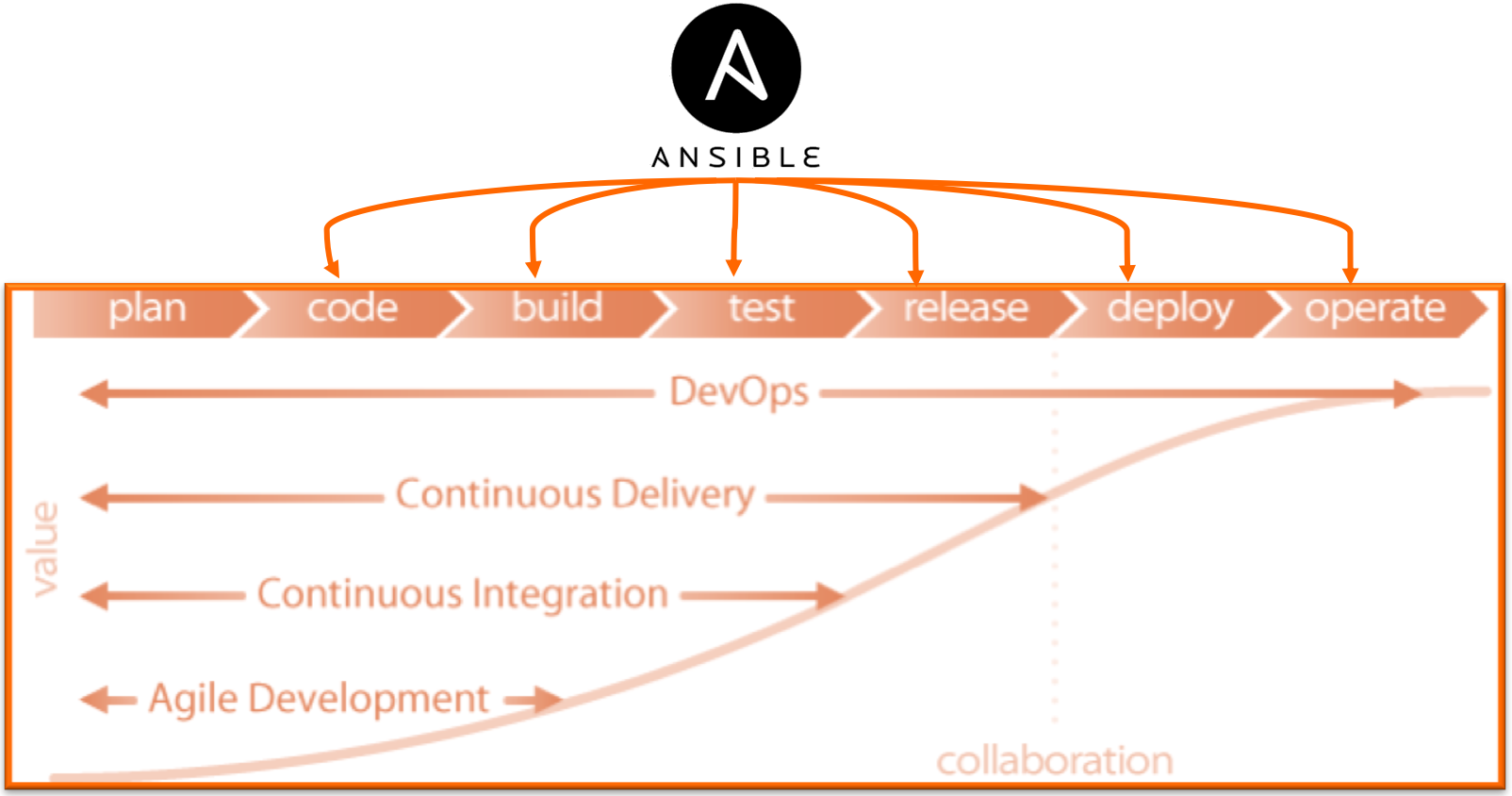
Simple Answer: DevOps can be considered the accumulation of industry best practices, processes, procedures, methods, and frameworks, all implemented in such a way as to ensure continuous integration of the organization's departments and teams, thereby empowering them to continuously deliver the organization's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

(Not so simple though, is it?)

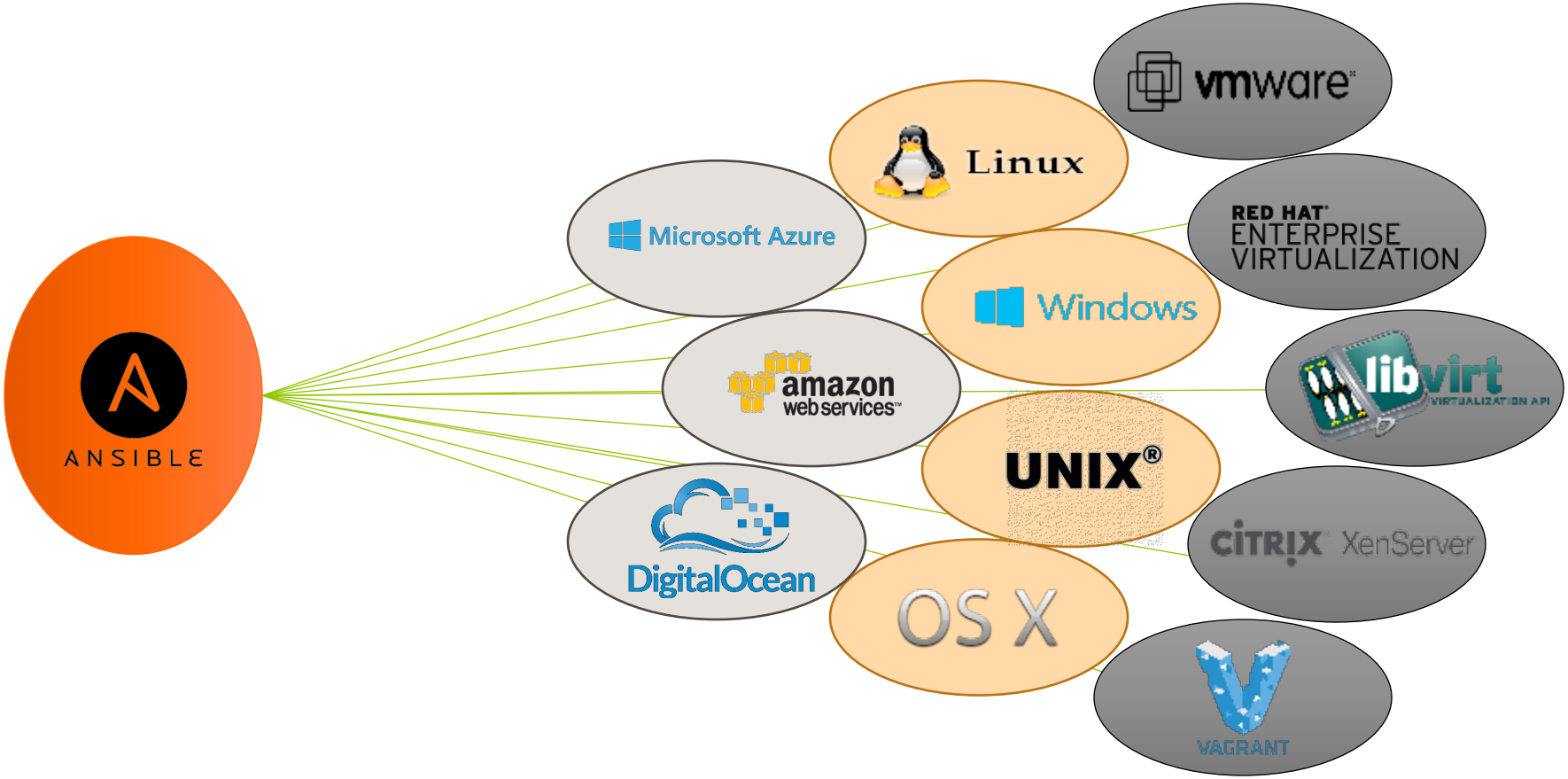
Ansible's Place In The DevOps World

- 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

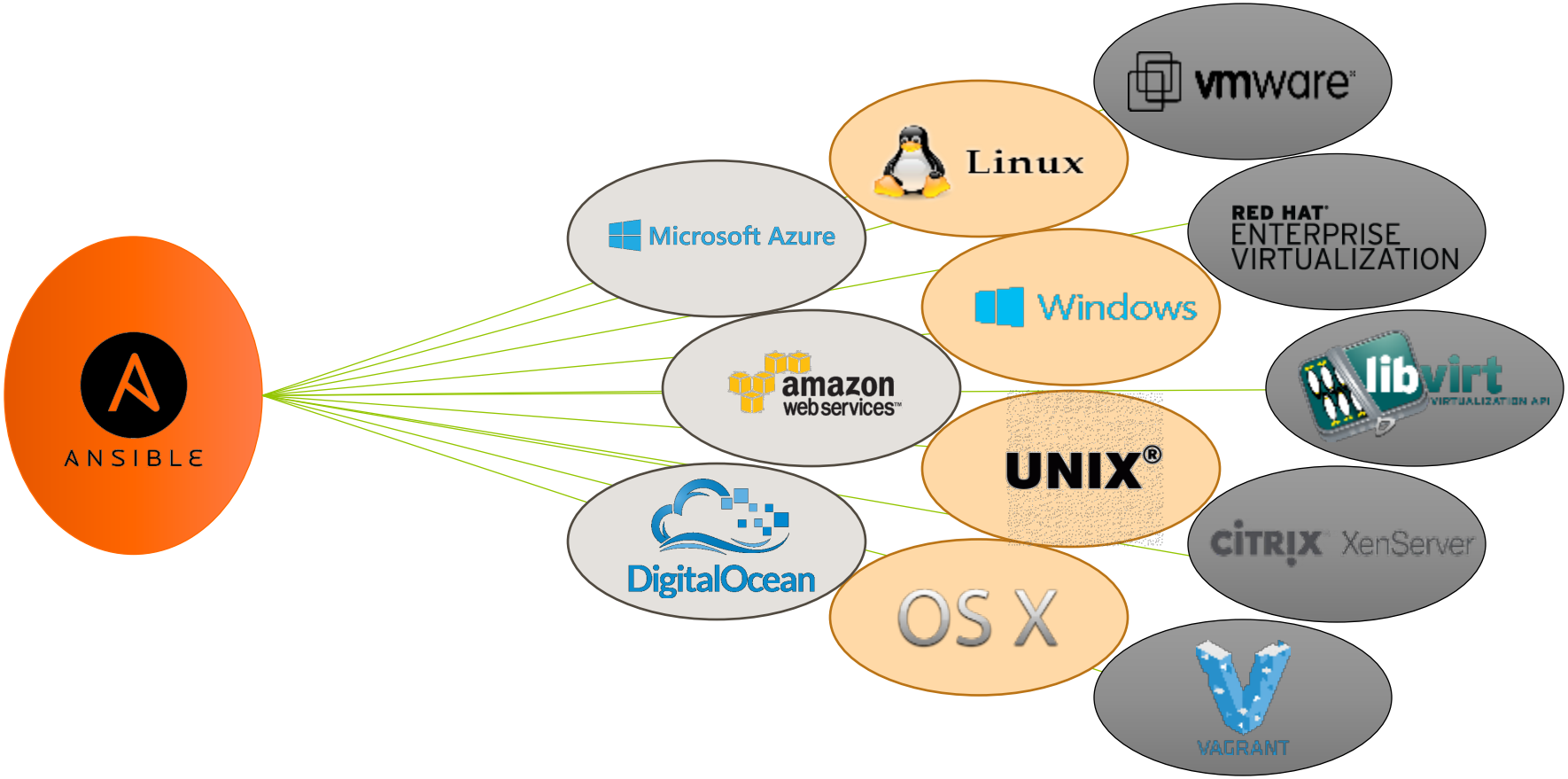
Continuous Integration
Continuous Delivery
Continuous Improvement



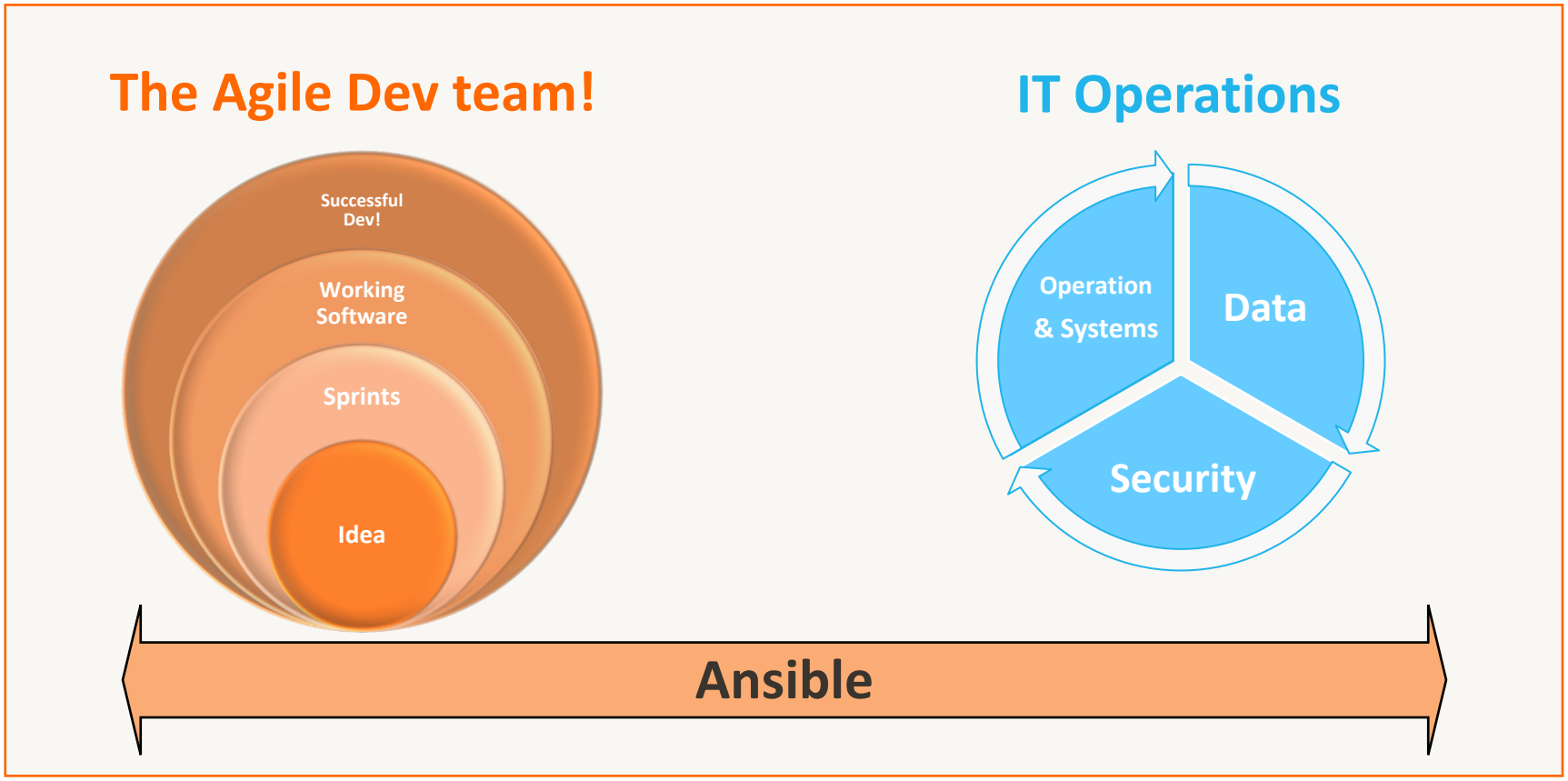
Ansible: Simplified Automation



Ansible: Simplified Automation



Ansible in The DevOps Team:



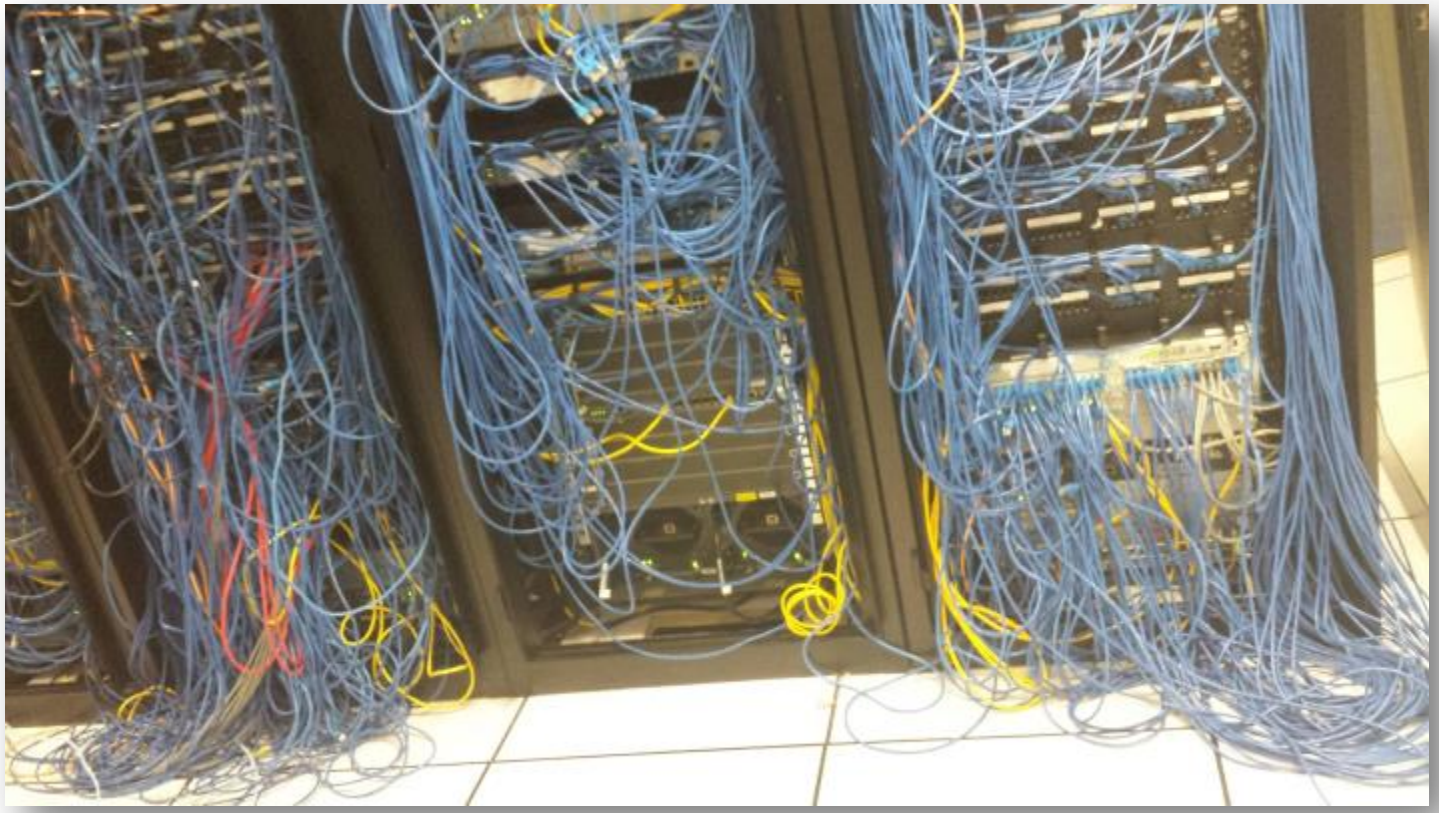
Key Term: Configuration Management

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

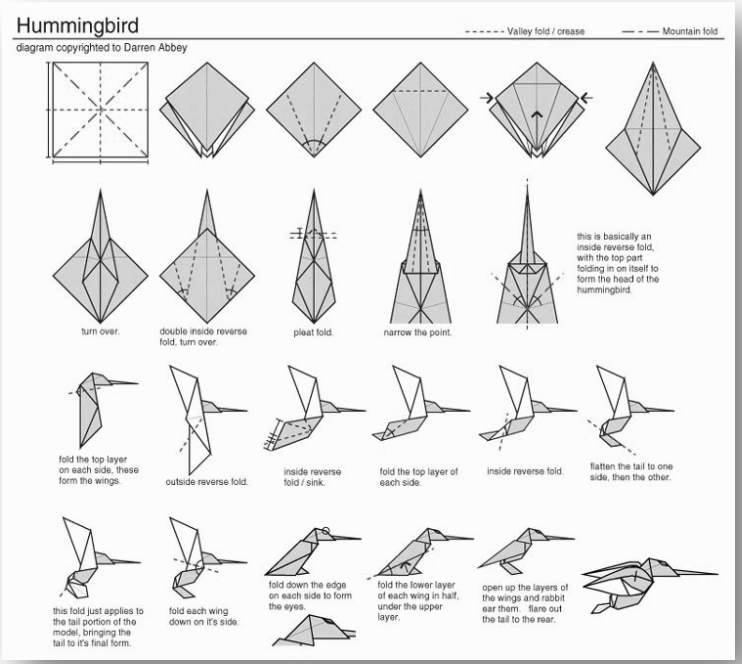
Let's clarify the definition of "configuration management."

Notes

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

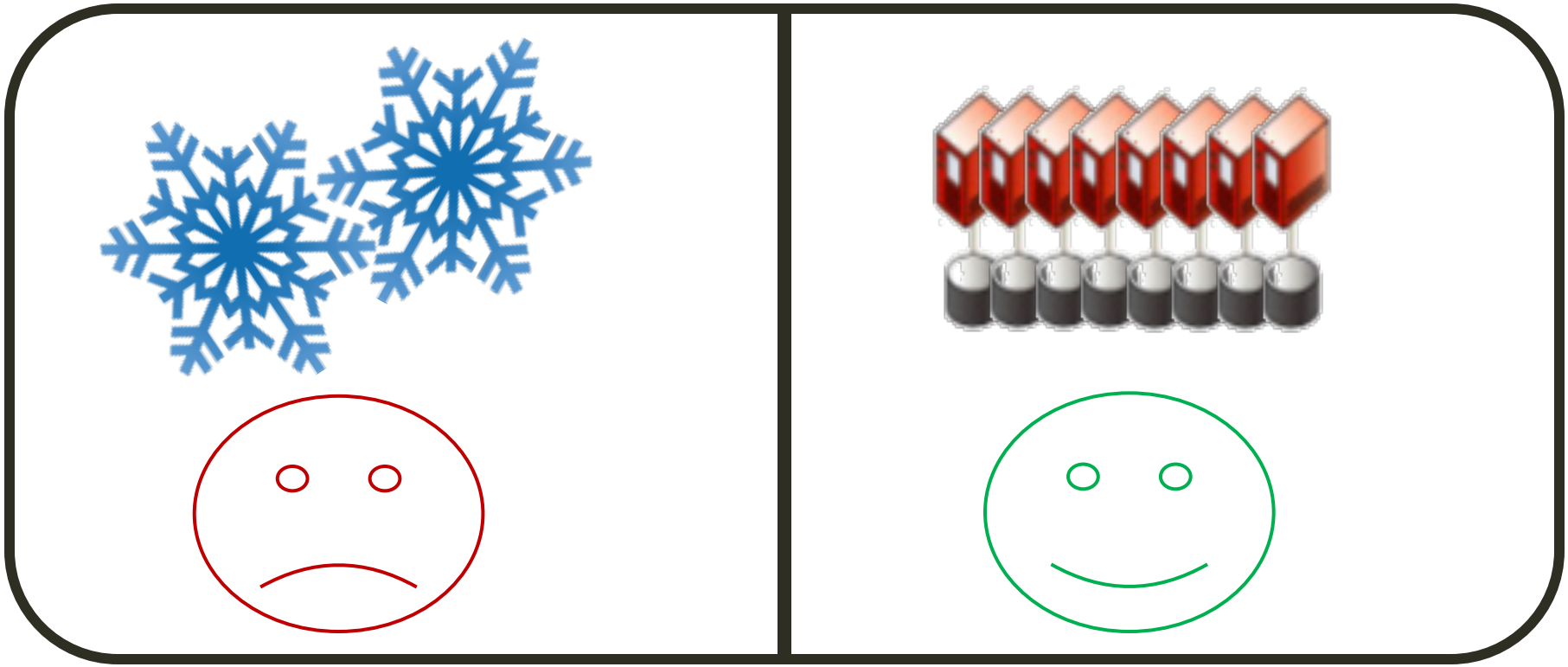
- No comprehensive GUI integrated by default. Alternative solutions are needed if a GUI is desired, such as Ansible Tower, Rundeck, or The Foreman.
- Currently, only minimal support/compatibility for Windows, as the Ansible Controller, however this is one area where Ansible developers are focusing their 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



Module Review: In this Module, you learned…

- The Origins of Ansible
- The definition of DevOps, what it is and why is it important?
- Ansible's place in the DevOps world
- How Ansible simplifies automation of multiple types of servers
- How automation using Ansible can remove slowness and defects from changes in infrastructure

Getting Started with Ansible

Module 2

This Section Will Cover:

- **Ansible Basics**
  - **Terminology**
  - **Dependencies/Constraints**
  - **Components**
  - **The typical Ansible Environment**
  - **Best Practices**
- **Authentication Basics**
  - **Authenticating with SSH Keys**
- **Vagrant Basics**

Ansible Basics:
Common Ansible Terminology

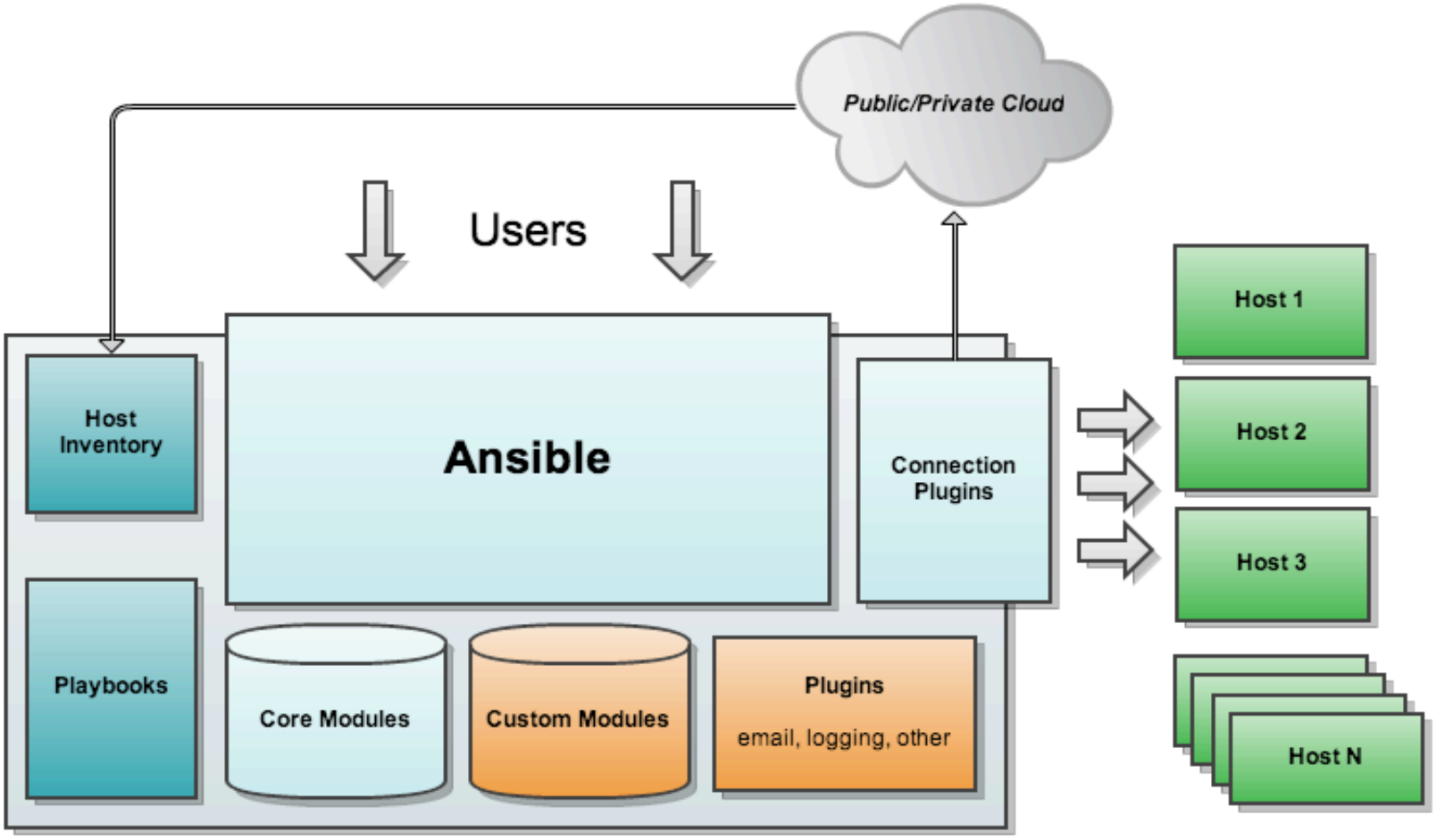
Ansible Controller	The host/device we execute our Ansible code from
Inventory	A list of servers/devices in our environment Ansible will connect to
Playbook	The state definition we want Ansible to apply to our servers
Roles	A package of tasks, config files, templates and variables we can include from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

Notes

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



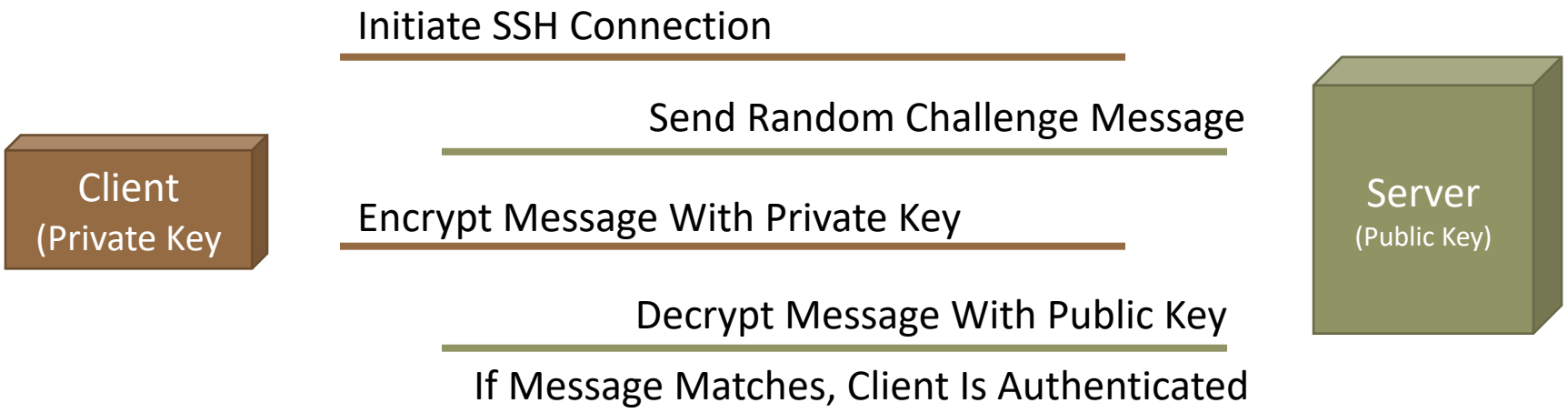
Ansible Basics: Best Practices

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**

Notes

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

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

```

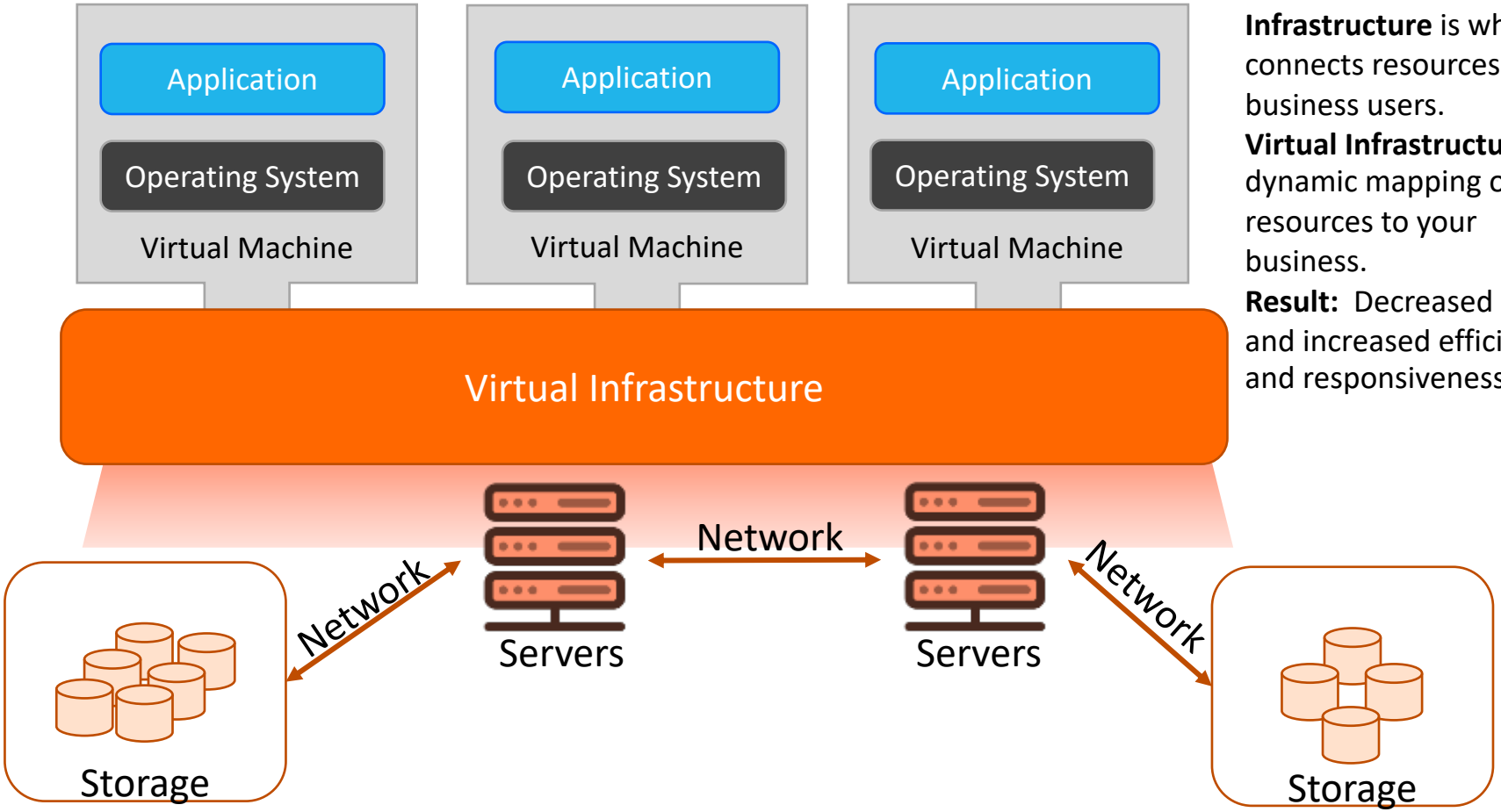
What is Our Classroom Lab Setup Like?

- Our lab environment is made up of a virtual machine which you with use SSH to connect to. On this machine there will be five Docker containers that your labs will take place in.
- When you first log to your student base machine using the credentials provided by your instructor, you are now in the virtual machine, to setup your Docker environment for the lab follow the “prerequisites” section of the lab guide.
- Your Ansible control machine will be a Docker container. Although this is not a course on Docker, be aware that some Docker commands will be needed.

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 users.

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

Result: Decreased costs and increased efficiencies and responsiveness.

Installation and Lab Environment Walkthrough

Exercise Objective A: Accessing the Lab Environment

Exercise

Module Review: In this Module, you learned…

Module Review

- Common Ansible Terminology.
- Ansible Dependencies and Constraights and how to work around them.
- What Ansible Best Practices look like.
- How to create SSH keys and utilize with your Ansible environment.
- What Vagrant is and how it can be used.
- A brief overview of Virtualization.
- How to access your lab environment.

Ansible Fundamentals

Module 3

This Section Will Cover:

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

This Section Will Cover (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.

Command Line Basics

- Exercise Objective A: Install Ansible
- Exercise Objective B: Post Installation Tasks
- Exercise Objective C: Building an Inventory
- Exercise Objective D: Ansible Command Line Basics

Exercise

Command Line Basics

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

Exercise

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

- 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/}
```

Notes

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
```

Building a Playbook

- **Exercise Objective A: Create a task using a playbook**

Exercise

Building a Playbook

- **Exercise Objective A: Exercise 5**

Exercise

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, i.e. 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_”.

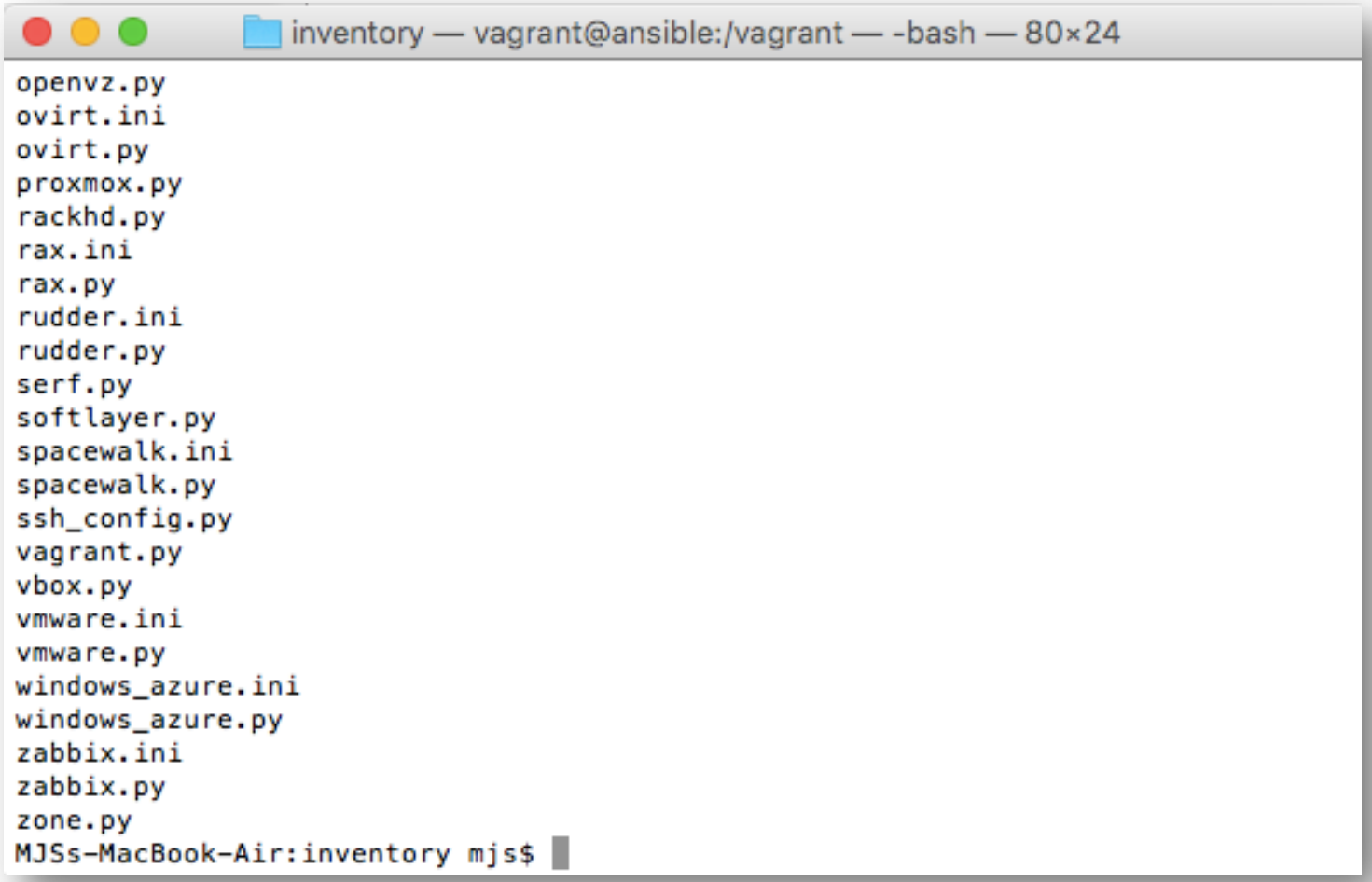
Finally, adjust the variables in the vars file to point to the matching vault_ variables and ensure that the vault file is vault encrypted.

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.

Advanced Inventory

- **Exercise Objective A: Advanced Inventory**

Exercise

Advanced Inventory

- **Exercise Objective A: Exercise 6**

Exercise

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`

Notes

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”**

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

Advanced Playbooks

- **Exercise Objective A: Advanced Playbooks**

Exercise

Advanced Playbooks

- **Exercise Objective A: Exercise 7**

Exercise

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?

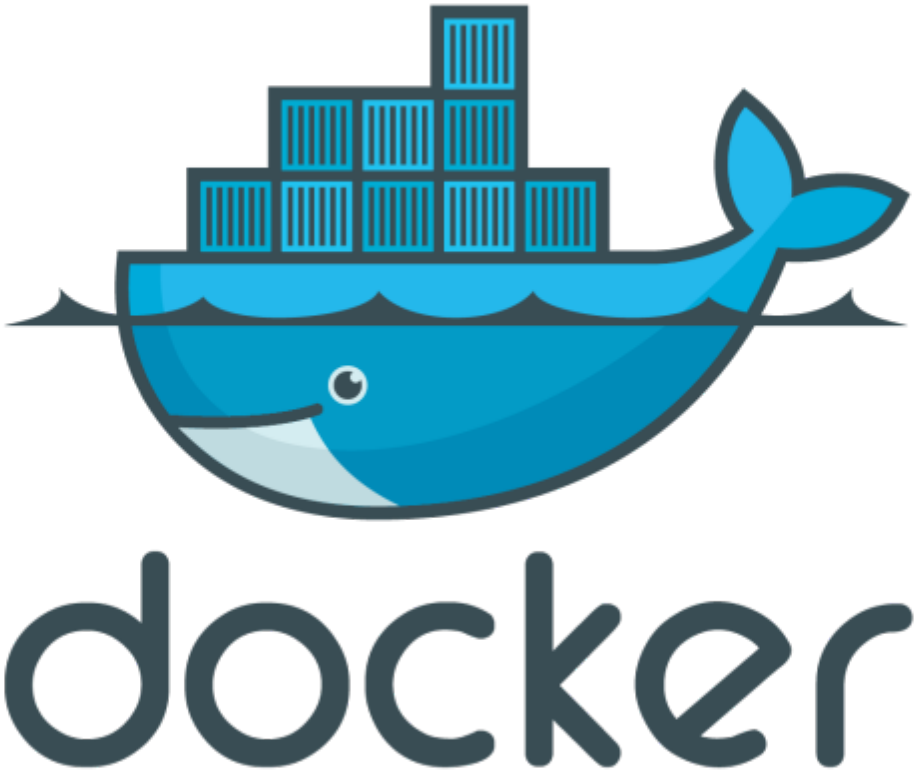
Ansible and Git

- 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.

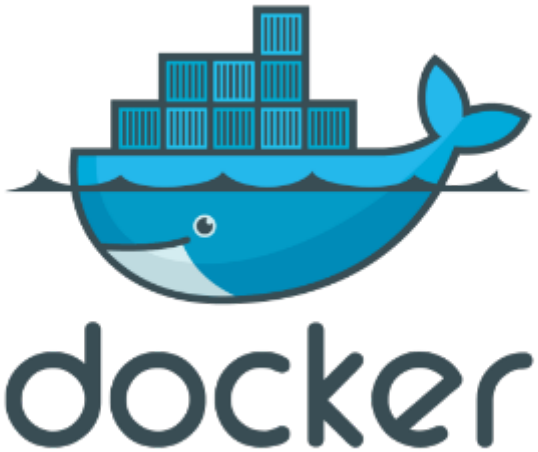
Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server.

Docker Application Life Cycle

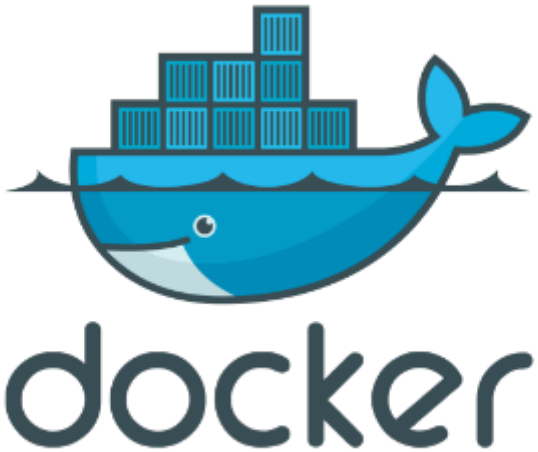
Here's what the typical lifecycle of a Docker-based application looks like:

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.**

A Brief Look at Ansible and Docker



A Brief Look at Ansible and Docker (Continued)



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
```

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.

Revisiting 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.

Module Review: In this Module, you learned…

Module Review

- Ansible command line.
- Ansible software plugins/modules/packages.
- How to manage users and inventory.
- How to install Ansible in a hands-on manner.
- What YAML is and how to use it.
- How to build a playbook in a hands-on manner.
- The benefit of Ansible roles and how they work.
- What Jinja2 basic syntax is and how to use it.
- How to handle errors in configurations.
- How to use Docker and git with your Ansible installation.
- How to handle Ansible updates and notifications.

Ansible Vault and Playbooks

Module 4

This Section Will Cover:

- **Managing Windows with Ansible**
- **Ansible Vault**
- **Ansible and PowerShell**
- **Ansible Playbooks**

Managing Windows with Ansible

- According to TechWorld, among the top ten Windows administration tasks are **Patching** and **User Administration**.
- Leveraging PowerShell and WinRM, Ansible takes the guesswork out of patching and user administration.
- Imagine a world where you can see, on one screen, all of the patches that were recently installed, which patches failed and on which servers, in your entire environment!
- What if you could see group membership for Domain Administrators and monitor any irregularities or changes across thousands of servers at once? With Ansible playbooks, *YOU CAN!*

Managing Windows with Ansible

- **Host Requirements**

  - Ansible's supported Windows versions generally match those under current and extended support from Microsoft. Supported desktop OSs include **Windows 7, 8.1, and 10**, and supported server OSs are **Windows Server 2008, 2008 R2, 2012, 2012 R2, and 2016**.

  - Ansible requires PowerShell 3.0 or newer and at least .NET 4.0 to be installed on the Windows host.

  - A WinRM listener should be created and activated. More details for this can be found here >>> https://docs.ansible.com/ansible/latest/user_guide/windows_setup.html

Managing Windows with Ansible

- Ansible requires PowerShell version 3.0 and .NET Framework 4.0 or newer to function on older operating systems like Server 2008 and Windows 7. The base image does not meet this requirement.

Managing Windows with Ansible

WinRM Configuration

- Once Powershell has been upgraded to at least version 3.0, the final step is for the WinRM service to be configured so that Ansible can connect to it.
- There are two main components of the WinRM service that governs how Ansible can interface with the Windows host: the **listener** and the **service** configuration settings.
- To see the current listeners that are running on the WinRM service, run the following command on your Windows server: winrm enumerate winrm/config/Listener

Managing Windows with Ansible

WinRM Configuration

- Output should look something like this:

```
Listener
    Address = *
    Transport = HTTP
    Port = 5985
    Hostname
    Enabled = true
    URLPrefix = wsman
    CertificateThumbprint
    ListeningOn = 10.0.2.15, 127.0.0.1, 192.168.56.155, ::1, fe80::5efe:10.0.2.15%6, fe80::5efe:192.168.56.155%8, fe80::
ffff:ffff:fffe%2, fe80::203d:7d97:c2ed:ec78%3, fe80::e8ea:d765:2c69:7756%7

Listener
    Address = *
    Transport = HTTPS
    Port = 5986
    Hostname = SERVER2016
    Enabled = true
    URLPrefix = wsman
    CertificateThumbprint = E6CDAA82EEAF2ECE8546E05DB7F3E01AA47D76CE
    ListeningOn = 10.0.2.15, 127.0.0.1, 192.168.56.155, ::1, fe80::5efe:10.0.2.15%6, fe80::5efe:192.168.56.155%8, fe80::
ffff:ffff:fffe%2, fe80::203d:7d97:c2ed:ec78%3, fe80::e8ea:d765:2c69:7756%7
```

Managing Windows with Ansible

Set up a WinRM listener

- **A great way for quickly and easily making the changes necessary for WinRM communication is simply running the following script.**
  ConfigureRemotingForAnsible.ps1

*psexec \\myserver -accepteula -nobanner -s -u DOMAIN\Administrator powershell -ExecutionPolicy Bypass -**Command** "iwr https://raw.githubusercontent.com/ansible/ansible/devel/examples/scripts/ConfigureRemotingForAnsible.ps1 -UseBasicParsing | iex"*

Managing Windows with Ansible

Set up a WinRM listener

- After running the script, you can test connectivity from Ansible to the Windows Server via WinRM by running the win_ping Ansible command. Below we can see both an unsuccessful WinRM test and a successful WinRM test from Ansible.

```
win2012ans.CLOUD.LOCAL | UNREACHABLE! => {
    "changed": false,
    "msg": "kerberos: authGSSClientStep() failed: (('Unspecified GSS failure.  M
inor code may provide more information', 851968), ('Ticket expired', -1765328352
))",
    "unreachable": true
}
```

Unsuccessful WinRM test from Ansible due to expired Kerberos credentials

```
win2012ans.CLOUD.LOCAL | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Successful test from the Ansible server to the Windows Server via WinRM

Managing Windows with Ansible

Automating Windows updates with Ansible

- Now that we have gone over the basics of setting up an Ansible server for Windows communication and seen how to ensure successful WinRM communication, let's look at how we can use this connection to automate Windows updates with Ansible.

Managing Windows with Ansible

Automating Windows updates with Ansible

- Ansible includes powerful modules "out of the box" that provide the mechanisms to install Windows updates on your Windows Servers.
- The win_updates module allows automating various aspects of installing Windows updates. Let's look at a few examples.

Managing Windows with Ansible

Automating Windows updates with Ansible

- If you want to install Windows updates by the category of the Windows updates, you can specify the category in the Ansible playbook directive of the win_updates module.

```
- name: Choose which Windows updates to install
  win_updates:
    category_names:
      - SecurityUpdates
      - CriticalUpdates
      - UpdateRollups
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- Note the difference between the GNU (Linux) patching playbook syntax vs the Windows playbook syntax:

```yaml
- name: Choose which Windows updates to install
  win_updates:
    category_names:
      - SecurityUpdates
      - CriticalUpdates
      - UpdateRollups
```

```yaml
- name: install hotfix without validating the KB and Identi
  win_hotfix:
    source: C:\temp\windows8.1-kb3172729-x64_e8003822a7ef470
    state: present
  register: hotfix_install

- win_reboot:
  when: hotfix_install.reboot_required
```

- This module requires GNU *patch* utility to be installed on the remote host.

Examples

```yaml
- name: Apply patch to one file
  patch:
    src: /tmp/index.html.patch
    dest: /var/www/index.html

- name: Apply patch to multiple files under basedir
  patch:
    src: /tmp/customize.patch
    basedir: /var/www
    strip: 1

- name: Revert patch to one file
  patch:
    src: /tmp/index.html.patch
    dest: /var/www/index.html
    state: absent
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- If you want to install only specific updates based on the KB numbers for a specific category, you can specify those updates via the whitelist directive in the YML code.

```
- name: Install specific updates based on the KBs for those updates
  win_updates:
    category_name:
    - SecurityUpdates
    whitelist:
    - KB4073819
    - KB4074228
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- If you simply want to check for missing updates, you can run the following. Note the difference between the syntax for Windows vs Linux:

```
1  # Check for missing updates
2    - name: Check for missing updates
3          win_updates: state=searched
4          register: update_count
```

Note

- This module actually calls the pertinent package modules for each system (apt, yum, etc).
- For Windows targets, use the win_package module instead.

Examples

```
- name: install ntpdate
  package:
    name: ntpdate
    state: present

# This uses a variable as this changes per distribution.
- name: remove the apache package
  package:
    name: "{{ apache }}"
    state: absent
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- What if you want to install the updates and reboot if the result records a reboot is necessary?

```
- name: Install all critical and security updates
  win_updates:
    category_names:
    - CriticalUpdates
    - SecurityUpdates
    - UpdateRollups
    state: installed
  register: update_result

        - name: reboot host if required
          win_reboot:
          when: update_result.reboot_required
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- Running an Ansible playbook is as simple as telling Ansible which inventory section you want to run the playbook against as well as which playbook you want to run. An example inventory file with domain controllers included will look something like this:

```
---
[DCs]
dc1.mydomain.local
dc2.mydomain.local
[AppServers]
app1.mydomain.local
app2.mydomain.local
[DBServers]
db1.mydomain.local
db2.mydomain.local
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- A sample windowsupdates.yml playbook to patch your domain controllers. Note how it directs the installation of security, critical, and update rollups:

```
---
- hosts: DCs
  tasks:
  - name: Choose which Windows updates to install
  win_updates:
    category_names:
      - SecurityUpdates
      - CriticalUpdates
      - UpdateRollups
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- A sample **windowsupdates.yml** playbook to patch your domain controllers. Note the difference between the Windows and Linux playbook syntax:

```
---
- hosts: DCs
  tasks:
  - name: Choose which Windows updates to install
  win_updates:
    category_names:
      - SecurityUpdates
      - CriticalUpdates
      - UpdateRollups
```

```
- name: install the latest version of Apache
  yum:
    name: httpd
    state: latest

- name: ensure a list of package s installed
  yum:
    name: "{{ package s }}"
  vars:
    package s:
    - httpd
    - httpd-tools

- name: remove the Apache package
  yum:
    name: httpd
    state: absent

- name: install the latest version of Apache from the testing repo
  yum:
    name: httpd
    enablerepo: testing
    state: present
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- You could also do something like the following: Configure the domain controllers (DCs) in this case with the settings in the windowsupdates.yml file.

```
1 ansible-playbook -i inventory.yml -s windowsupdates.yml
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- If a target machine is offline, Ansible will simply display the error connecting to the target machine and move on to the next server in the list to update.
- In the example below, we are searching for all available updates and piping these out to c:\ansible_wu.txt. We can do this with the following code:

Managing Windows with Ansible

Automating Windows updates with Ansible

```
1 - name: Search-only, return list of updates found (if any), log to c:\ansible_wu.txt
2   win_updates:
3     category_names:
4      -SecurityUpdates
5      -CriticalUpdates
6      -UpdateRollups
7     state: searched
8     log_path: c:\ansible_wu.txt
```

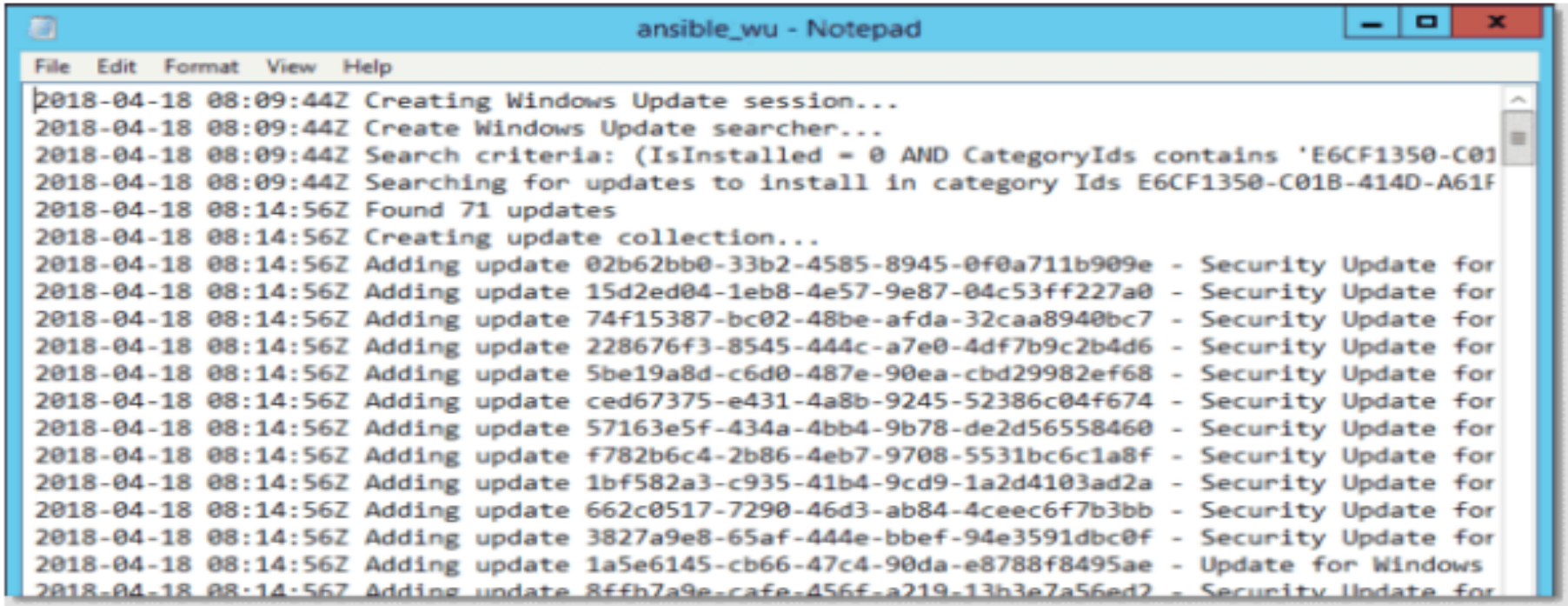
```
ok: [win2012ans.CLOUD.LOCAL]

TASK [test : Search-only, return list of found updates (if any), log to c:\ansib
le_wu.txt] ***


ok: [win2012ans.CLOUD.LOCAL]

PLAY RECAP *********************************************************************
win2012ans.CLOUD.LOCAL     : ok=2    changed=0    unreachable=0    failed=0
```

Running an Ansible playbook to search for available Windows updates

Managing Windows with Ansible

Automating Windows updates with Ansible

- The snippet below shows the contents of the c:\ansible_wu.txt file with the available Windows updates pulled from running the Ansible playbook.



Log file containing the missing Windows updates Ansible found

Managing Windows with Ansible

Automating Windows updates with Ansible

```
TASK [test : Search-only, return list of found updates (if any), log to c:\ansib
le_wu.txt] ***

ok: [win2012ans.CLOUD.LOCAL]

TASK [test : install all critical and security updates] ************************

changed: [win2012ans.CLOUD.LOCAL]

TASK [test : reboot host if required] ******************************************

changed: [win2012ans.CLOUD.LOCAL]

PLAY RECAP *********************************************************************
win2012ans.CLOUD.LOCAL     : ok=4    changed=2    unreachable=0    failed=0
```

Results after installing Windows updates and rebooting with Ansible

Managing Windows with Ansible

Automating Windows updates with Ansible

- Here we have a snippet of the Windowsupdate.log file showing Ansible indeed installed the updates.

```
Agent   ** START **  Agent: Installing updates [CallerId = <<PROCESS>>: powershell.exe]
Agent   *********
Agent     * Updates to install = 71
Agent     *   Title = Security Update for Microsoft .NET Framework 3.5 on Windows 8.1 and Windows Server 2012 R2 for x64-based Systems (KB2966826)
Agent     *   UpdateId = {02B62BB0-33B2-4585-8945-0F0A711B909E}.202
Agent     *     Bundles 1 updates:
Agent     *       {44FF7F80-399E-48B2-A90B-1B1FBBFEC334}.202
Agent     *   Title = Security Update for Microsoft .NET Framework 4.5.1 on Windows 8.1 and Windows Server 2012 R2 for x64-based Systems (KB2894856)
Agent     *   UpdateId = {15D2ED04-1EB8-4E57-9E87-04C53FF227A0}.203
Agent     *     Bundles 1 updates:
Agent     *       {3884B702-D936-4645-AA50-5EBE2AF73C45}.203
Agent     *   Title = Security Update for Microsoft .NET Framework 3.5 on Windows 8.1 and Windows Server 2012 R2 for x64-based Systems (KB2973114)
Agent     *   UpdateId = {74F15387-BC02-48BE-AFDA-32CAA8940BC7}.201
Agent     *     Bundles 1 updates:
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- Here we have a snippet of the Windowsupdate.log file showing Ansible indeed installed the updates.

```
Agent   ** START **  Agent: Installing updates [CallerId = <<PROCESS>>: powershell.exe]
Agent   *********
Agent     * Updates to install = 71
Agent     *   Title = Security Update for Microsoft .NET Framework 3.5 on Windows 8.1 and Windows Server 2012 R2 for x64-based Systems (KB2966826)
Agent     *   UpdateId = {02B62BB0-33B2-4585-8945-0F0A711B909E}.202
Agent     *     Bundles 1 updates:
Agent     *       {44FF7F80-399E-48B2-A90B-1B1FBBFEC334}.202
Agent     *   Title = Security Update for Microsoft .NET Framework 4.5.1 on Windows 8.1 and Windows Server 2012 R2 for x64-based Systems (KB2894856)
Agent     *   UpdateId = {15D2ED04-1EB8-4E57-9E87-04C53FF227A0}.203
Agent     *     Bundles 1 updates:
Agent     *       {3884B702-D936-4645-AA50-5EBE2AF73C45}.203
Agent     *   Title = Security Update for Microsoft .NET Framework 3.5 on Windows 8.1 and Windows Server 2012 R2 for x64-based Systems (KB2973114)
Agent     *   UpdateId = {74F15387-BC02-48BE-AFDA-32CAA8940BC7}.201
Agent     *     Bundles 1 updates:
```

Managing Windows with Ansible

Automating Windows updates with Ansible

- If you are interested, a Windows VM can be provided to you to test out the communication setup, configuration and management of a Windows server with Ansible. Please notify your instructor if you would like to work on this as part of your lab experience.

Managing Ansible Vault

What is Ansible Vault?

- Ansible Vault allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles.
- To enable this feature, a command line tool - ansible-vault - is used to edit files, and a command line flag (--ask-vault-pass or --vault-password-file) is used.

Creating Encrypted Files

To create a new encrypted data file, run the following command:

```
ansible-vault create foo.yml
```

First you will be prompted for a password. The password used with vault currently must be the same for all files you wish to use together at the same time.

Ansible Vault

What is Ansible Vault?

- Ansible Vault allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles.
- To enable this feature, a command line tool - ansible-vault - is used to edit files, and a command line flag (--ask-vault-pass or --vault-password-file) is used.

Creating Encrypted Files

To create a new encrypted data file, run the following command:

```
ansible-vault create foo.yml
```

First you will be prompted for a password. The password used with vault currently must be the same for all files you wish to use together at the same time.

Ansible Vault

What can be encrypted using Ansible Vault?

- Ansible Vault can encrypt any structured data file used by Ansible. This can include “group_vars/” or “host_vars/” inventory variables. Role variables and defaults are also included.
- Ansible Vault can also encrypt arbitrary files, even binary files.
- As of version 2.3, Ansible supports encrypting single values inside a YAML file, using the !vault tag to let YAML and Ansible know it uses special processing.

Ansible Vault

Single Encrypted Variable

As of version 2.3, Ansible can now use a vaulted variable that lives in an otherwise 'clear text' YAML file:

```
notsecret: myvalue
mysecret: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          663864396532363364626265666530633361646639663032313639346535613639643633833313662
          643162653630353037633634383265653730363231343336 0a626438346336353331386135323734
          626563616536303732316136626339623162336339363961653864396165333539653733396162 34
          343061353966633039 0a313736323265656432366236633330313963326365653937323833366536
          346237313766646231343834633162656434363434386232666239656363633261 36
other_plain_text: othervalue
```

Ansible Vault

To use a vault-id label for 'dev' vault-id:

```
ansible-vault encrypt_string --vault-id dev@password 'fooooodev' --name 'the_dev_secret'
```

Result:

```
the_dev_secret: !vault |
          $ANSIBLE_VAULT;1.2;AES256;dev
          30613233633461343837653833666333643061636561303338373661313838333565653635353162
          32633634346237333435386534626130643336343334646660a6636336239393934393166366338 63
          61636237636537333938306331383339353265363239643939666639386530626330633337633833
          66646563343731666630a3637363932626664656634326139326130363039633432636231373862 39
          6330
```

To encrypt a string read from stdin and name it 'db_password':

```
echo -n 'letmein' | ansible-vault encrypt_string --vault-id dev@password --stdin-name 'db_password'
```

Result:

```
Reading plaintext input from stdin. (ctrl-d to end input)
db_password: !vault |
          $ANSIBLE_VAULT;1.2;AES256;dev
          61323931353866666336306139373937316366366138656131323863373866376666353364373761
          35396332343138363464353237663061646261343765643 30a373530313635343535343133316133
          36643666306434616266376434363239346433643238336464643566386135356334303736353136
          65656331333663663 60a32656632336336393661366461636462343733613062313334353033 9739
          3039
```

To be prompted for a string to encrypt, encrypt it, and give it the name 'new_user_password':

```
ansible-vault encrypt_string --vault-id dev@./password --stdin-name 'new_user_password'
```

Ansible Vault

Most common use of Ansible Vault: Passwords

- Since Ansible 2.4 and later support using multiple vault passwords, --vault-id can be provided multiple times.
- As of version 2.3, Ansible supports encrypting single values inside a YAML file, using the !vault tag to let YAML and Ansible know it uses special processing.

Ansible Vault

Most common use of Ansible Vault: Passwords

For example, to use a 'dev' password read from a file and to be prompted for the 'prod' password:

```
ansible-playbook --vault-id dev@dev-password --vault-id prod@prompt site.yml
```

In the above case, the 'dev' password will be tried first, then the 'prod' password for cases where Ansible doesn't know which vault id is used to encrypt something.

There is also a config option (DEFAULT_VAULT_IDENTITY_LIST) to specify a default list of vault ids to use. For example, instead of requiring the cli option on every use, the (DEFAULT_VAULT_IDENTITY_LIST) config option can be used:

```
ansible-playbook --vault-id dev@dev-password --vault-id prod@prompt site.yml
```

The `--vault-id` can be used in lieu of the `--vault-password-file` or `--ask-vault-pass` options, or it can be used in combination with them.

When using ansible-vault commands that encrypt content (ansible-vault encrypt, ansible-vault encrypt_string, etc) only one vault-id can be used.

Ansible Vault – Lab Exercise

Let's use what we've learned to encrypt and decrypt a file using Vault:

To create a new file encrypted with Vault, use the `ansible-vault create` command. Pass in the name of the file you wish to create. For example, to create an encrypted YAML file called `vault.yml` to store sensitive variables, you could type:

```
$ ansible-vault create vault.yml
```

You will be prompted to enter and confirm a password:

```
Output
New Vault password:
Confirm New Vault password:
```

Ansible Vault – Lab Exercise

When you have confirmed your password, Ansible will immediately open an editing window where you can enter your desired contents.

To test the encryption function, enter some test text:

vault.yml

```
Secret information
```

Ansible will encrypt the contents when you close the file. If you check the file, instead of seeing the words you typed, you will see an encrypted block:

```
$ cat vault.yml
```

Output

```
$ANSIBLE_VAULT;1.1;AES256
65316332393532313030636134643235316439336133363531303838376235376635373430336333
39633536303731613566383763616463383537633634360a363138376163666265336433633664
30336233323664306434626363643731626536643833336638356661396364313666366231616261
37646563653132636 20a383666383233626665376364323062393462373266663066366536306163
31643731343666353761633563633634326139396230313734333034653238303166
```

Ansible Vault – Lab Exercise

Viewing Encrypted Files

Sometimes, you may need to reference the contents of a vault-encrypted file without needing to edit it or write it to the filesystem unencrypted. The `ansible-vault view` command feeds the contents of a file to standard out. By default, this means that the contents are displayed in the terminal.

Pass the vault encrypted file to the command:

```
$ ansible-vault view vault.yml
```

You will be asked for the file's password. After entering it successfully, the contents will be displayed:

```
Output
Vault password:
Secret information
```

Ansible Vault – Lab Exercise

Editing Encrypted Files

When you need to edit an encrypted file, use the `ansible-vault edit` command:

```
$ ansible-vault edit vault.yml
```

You will be prompted for the file's password. After entering it, Ansible will open the file an editing window, where you can make any necessary changes.

Upon saving, the new contents will be encrypted using the file's encryption password again and written to disk.

Ansible Vault – Lab Exercise

Manually Decrypting Encrypted Files

To decrypt a vault encrypted file, use the `ansible-vault decrypt` command.

> **Note:** Because of the increased likelihood of accidentally committing sensitive data to your project repository, the `ansible-vault decrypt` command is only suggested for when you wish to remove encryption from a file permanently. If you need to view or edit a vault encrypted file, it is usually better to use the `ansible-vault view` or `ansible-vault edit` commands, respectively.

Pass in the name of the encrypted file:

```
$ ansible-vault decrypt vault.yml
```

You will be prompted for the encryption password for the file. Once you enter the correct password, the file will be decrypted:

```
Output
Vault password:
Decryption successful
```

Ansible Vault – Lab Exercise

Manually Decrypting Encrypted Files

To decrypt a vault encrypted file, use the `ansible-vault decrypt` command.

> **Note:** Because of the increased likelihood of accidentally committing sensitive data to your project repository, the `ansible-vault decrypt` command is only suggested for when you wish to remove encryption from a file permanently. If you need to view or edit a vault encrypted file, it is usually better to use the `ansible-vault view` or `ansible-vault edit` commands, respectively.

Pass in the name of the encrypted file:

```
$ ansible-vault decrypt vault.yml
```

You will be prompted for the encryption password for the file. Once you enter the correct password, the file will be decrypted:

```
Output
Vault password:
Decryption successful
```

Ansible Vault – Lab Exercise

If you view the file again, instead of the vault encryption, you should see the actual contents of the file:

```
$ cat vault.yml
```

```
Output
Secret information
```

Your file is now unencrypted on disk. Be sure to remove any sensitive information or re-encrypt the file when you are finished.

Ansible Vault

Speeding up Vault Operations

- By default, Ansible uses “PyCrypto” to encrypt and decrypt vault files. If you have many encrypted files, decrypting them at startup may cause a perceptible delay. To speed this up, install the cryptography package:

```
pip install cryptography
```

Ansible and PowerShell

What we will cover in this module:

- ❑ What is PowerShell?
- ❑ Advantages of using PowerShell with Ansible
- ❑ Ansible basics for managing Windows Servers using PowerShell
- ❑ Examples of executing remote PowerShell commands on Windows servers using the win_command, win_shell and win_feature modules.

Ansible and PowerShell

What is PowerShell?

- Microsoft PowerShell is arguably the most powerful scripting language for automating configuration, management, and other Microsoft workstation and server functions.
- PowerShell combines the speed of the command line with the flexibility of a scripting language, making it a valuable Windows administration tool.
- By using Windows PowerShell with Ansible, we benefit from Ansible for automation and the power of PowerShell control for Windows resources.

Ansible and PowerShell

Advantages of using PowerShell with Ansible

- Ansible is a configuration management and orchestration platform that allows organizing resources intelligently, defining roles, as well as writing "playbooks" that build what the server role should look like.
- Additionally, the win_modules for Ansible typically aim to interact through the Ansible Python interface in the backend when communicating with the Windows Server.
- Ansible gives Windows administrators the ability to interact directly with PowerShell Windows modules and playbooks.

Ansible and PowerShell

Ansible basics for managing Windows Servers

- The modules you will primarily use when working with PowerShell using Ansible are the **win_command** module and the **win_shell** module.

- The win_command module can run Windows commands including PowerShell scripts by calling the PowerShell executable. This is generally the preferred method of running scripts such as PowerShell.
- The win_shell module is a less secure means of running PowerShell scripts since the user's environment affects it. It is similar to the win_command module; however, it runs the command via a shell on the target Windows Server.

Ansible and PowerShell

Examples of Windows Modules in Playbooks

```
- name: Run remote PowerShell Script
  win_command: powershell.exe -ExecutionPolicy ByPass -File C:/temp/powershellscript.ps1
```

```
- name: Run remote PowerShell script
  win_shell: C:\somescript.ps1 >> c:\somelog.txt
```

Sample playbook:

```
---
- hosts: web
  tasks:
  - name: Ensure IIS webserver is installed
    win_feature:
      name: Web-Server
      state: present
```

Ansible and PowerShell

Examples of Windows Modules in Playbooks

Sample Inventory.yml file:

```
---

[app]
app.domain.com

[web]
web.domain.com

[db]
db.domain.com
```

Sample playbook Ansible run:

Using the templates shown above, you can easily run a playbook against your web servers using the following Ansible command:

```
ansible-playbook -i inventory.yml -s web.yml
```

Ansible and PowerShell

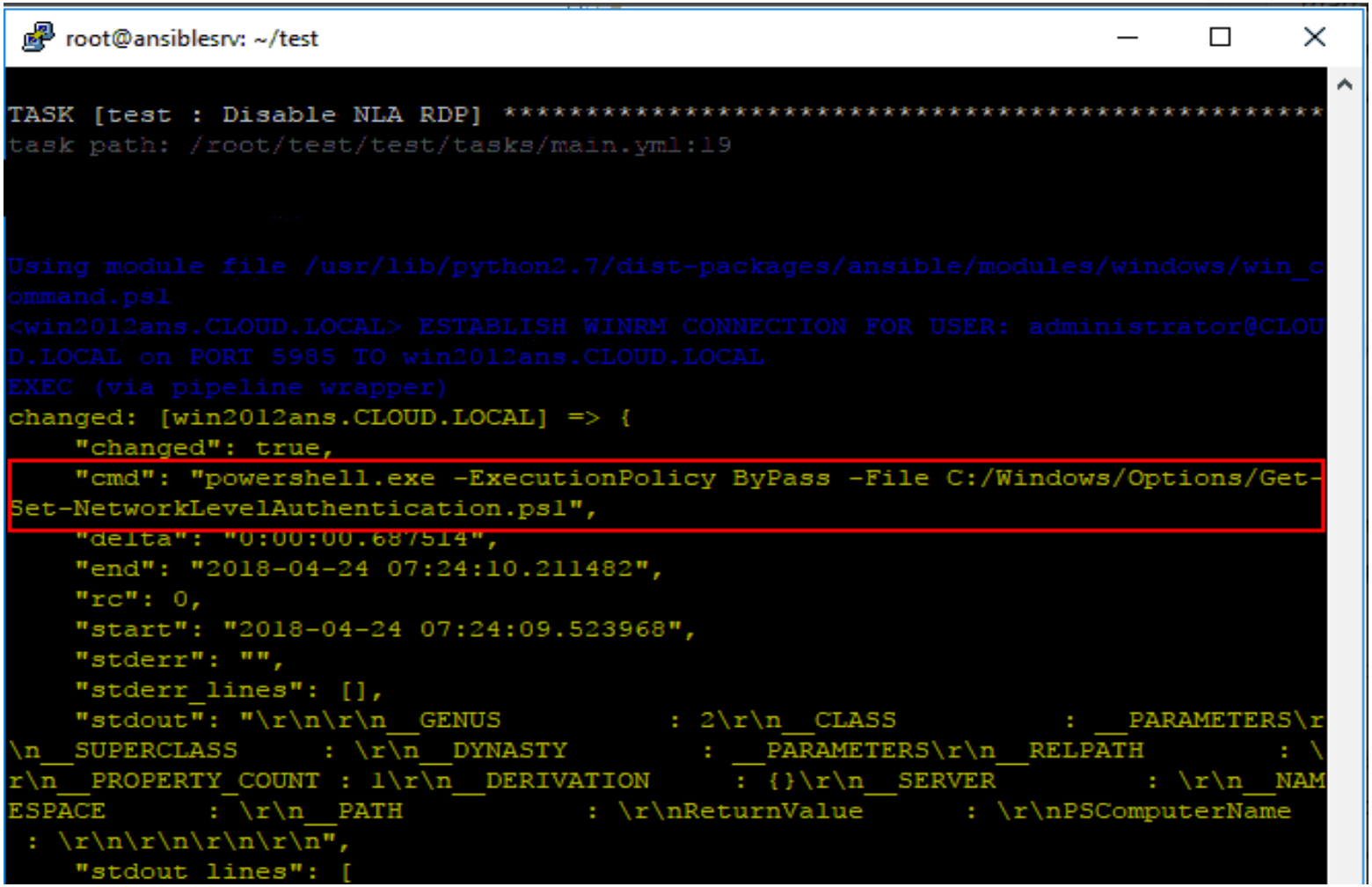
Using win_command module

- Using a Windows Server playbook to run PowerShell scripts via the win_command module, we can easily run scripts on remote Windows Servers. For example, look at the output on the next slide that shows us using the **win_command** module to set network level authentication on RDP.

- Ansible notifies us at the end of running a playbook how many changes it implemented (designated in yellow).

Ansible and PowerShell

Using win_command module



Ansible and PowerShell

Using win_shell module

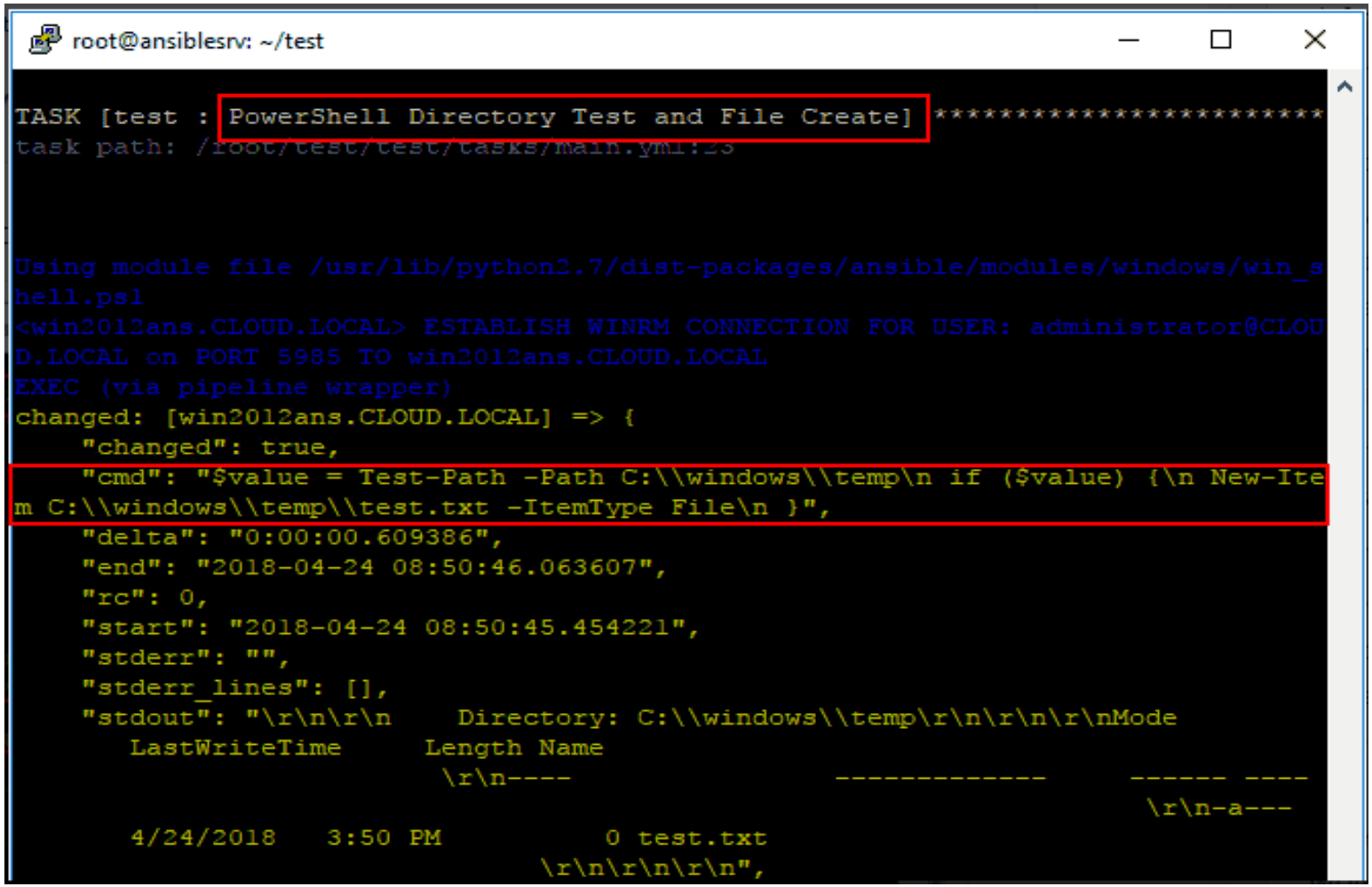
- With the **win_shell** module you can run PowerShell scripts with the .PS1 extension, or you can literally run PowerShell cmdlets from the Ansible playbook itself.

- Again, it's worth mentioning **win_shell** is a less secure means of running PowerShell code on the target Windows Server hosts.

```
1  - name: PowerShell Directory Test and File Create
2    win_shell: |
3      $value = Test-Path -Path C:\windows\temp
4      if ($value) {
5          New-Item C:\windows\temp\test.txt -ItemType File
6      }
```

Ansible and PowerShell

Using win_shell module



Ansible and PowerShell

Using win_feature module

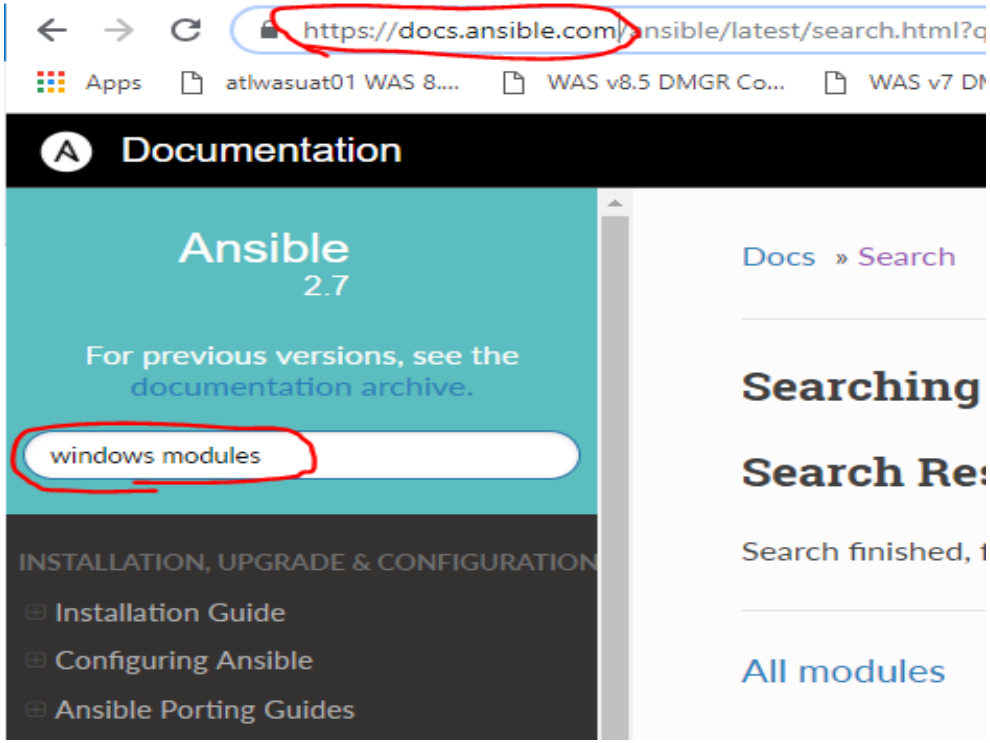
- With the **win_feature** module you can install or uninstall Windows Roles or Features on Windows Servers.

```
- name: Install IIS (Web-Server only)
  win_feature:
    name: Web-Server
    state: present

- name: Install IIS (Web-Server and Web-Common-Http)
  win_feature:
    name:
    - Web-Server
    - Web-Common-Http
    state: present
```

Ansible and PowerShell

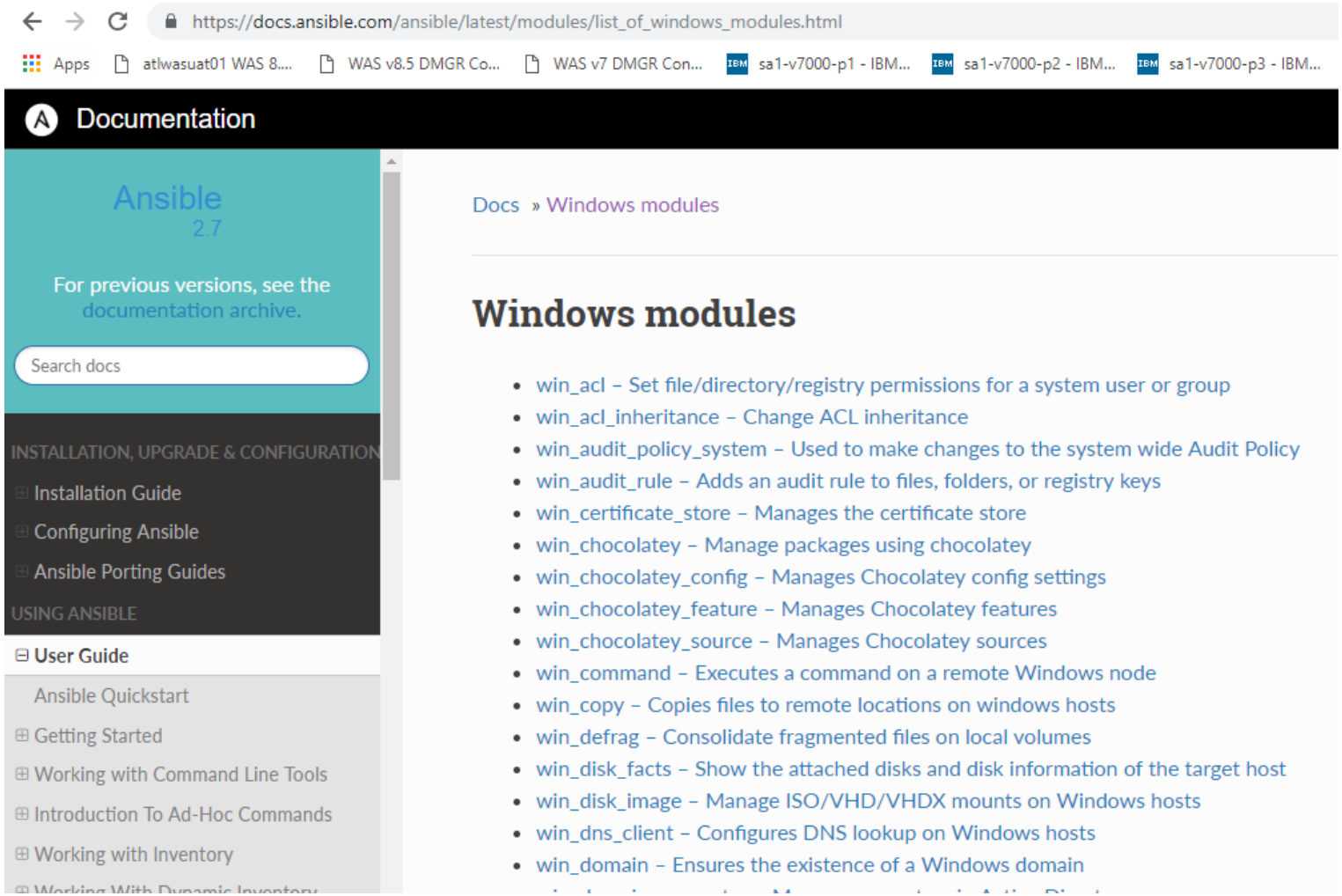
Windows Modules

- For a full list of all Windows modules available through Ansible, please go to https://docs.ansible.com and do a search for "Windows modules".



Ansible and PowerShell

Windows Modules



Ansible and PowerShell

Configuring Windows for Ansible

❑ Getting up and running with Ansible is as simple as:

1. Installing an Ansible control Linux server along with the Windows WinRM prerequisites.

2. Enabling WinRM connectivity on the target Windows Servers (possible via PsExec using the .ps1 script).

3. Setting up an inventory file to define the Windows Servers you want to control.

4. Creating a playbook.

5. Running the “ansible-playbook” command telling Ansible which servers in inventory you want to control and which playbook to run against those servers.

Creating Ansible Playbooks

Lab Exercise – Managing Windows

- Let's use what we've learned about managing Windows with PowerShell in a lab exercise. In this exercise, we'll create an Ansible playbook that will use an encrypted password to install the IIS Windows package and start it.
- Prerequisites – The following will need to be configured in order for you to complete this lab exercise:
  - WinRM configured on remote Windows server
  - Access to remote Windows server to execute PowerShell
  - Administrative user ID password to remote Windows server
  - Windows server added to your "inventory" file in a "windows" group
- **NOTE: If you do not have the prerequisites completed or you are having issues, please let your instructor know so that they can contact customer support ASAP.**

Creating Ansible Playbooks

Lab Exercise

- First, let's create a directory under our /ansible directory called "win_playbooks". You can do this by running the following command:
  mkdir win_playbooks
- Change directory (cd) into your win_playbooks directory and use VI to create a new playbook called **install_iis.yml**
- Remember that you have to have the three dashes at the top of your YAML file. We'll take it line by line here:

```
1  ---
2  # This playbook installs and enables IIS on Windows hosts
3
4  - name: Install IIS
```

Creating Ansible Playbooks

Lab Exercise

- Next, let's identify which group of servers we want to install IIS on. We've created our "windows" group in our inventory file with our remote Windows server. Add the "windows" group as the "hosts" entry in your playbook.

```
1  ---
2  # This playbook installs and enables IIS on Windows hosts
3
4  - name: Install IIS
5    hosts  windows
```

Creating Ansible Playbooks

Lab Exercise

- Next, we're going tell Ansible to not "gather facts" on our Windows server by setting that value to "false".

```
1  ---
2  # This playbook installs and enables IIS on Windows hosts
3
4  - name: Install IIS
5    hosts  windows
6    gather_facts: false
```

Creating Ansible Playbooks

Lab Exercise

- Next, we're going start building out our tasks. Note the syntax here, if the alignment is off, the playbook will not function properly.
- Note that we're using the "win_feature" module to install our IIS package. We're also passing it the "present" value as the final "state", which means it shoud be installed by the time Ansible is done running the playbook.

```
1  ---
2  # This playbook installs and enables IIS on Windows hosts
3
4  - name: Install IIS
5    hosts  windows
6    gather_facts: false
7    tasks:
8      - name: Install IIS
9        win_feature:
10         name: "Web-Server"
11         state: present
12         restart: yes
13         include_sub_features: yes
14         include_management_tools: yes
```

Creating Ansible Playbooks

Lab Exercise

- Next, we're going start building out our tasks. Note the syntax here, if the alignment is off, the playbook will not function properly.
- Note that we're using the "win_feature" module to install our IIS package. We're also passing it the "present" value as the final "state", which means it shoud be installed by the time Ansible is done running the playbook.

```
1  ---
2  # This playbook installs and enables IIS on Windows hosts
3
4  - name: Install IIS
5    hosts  windows
6    gather_facts: false
7    tasks:
8      - name: Install IIS
9        win_feature:
10         name: "Web-Server"
11         state: present
12         restart: yes
13         include_sub_features: yes
14         include_management_tools: yes
```

Creating Ansible Playbooks

Lab Exercise

- That's it! It's time to run our playbook and install IIS. If you do see any errors when running your install_iis.yml playbook, troubleshoot as a group and ask your instructor for assistance. Good work!! ☺

```
---
# This playbook installs and enables IIS on Windows hosts

- name: Install IIS
  hosts  windows
  gather_facts: false
  tasks:
    - name: Install IIS
      win_feature:
        name: "Web-Server"
        state: present
        restart: yes
        include_sub_features: yes
        include_management_tools: yes
```

```
ansible-playbook install_iis.yml
```

Creating Ansible Playbooks

In this section we will cover:

- ❑ What the different sections of the playbook do
- ❑ Managing package installs using playbooks
- ❑ How to use “handlers” to execute additional playbooks
- ❑ Examples of playbooks and further documentation

Creating Ansible Playbooks

Creating basic Ansible playbooks

- **Tip! Start small.** In the spirit of the Agile methodology, build your "minimum viable product" (MVP) and then iterate on that.
- What are the different sections of a playbook?
  - Name: Simple but clear description of what task you're running. (Example: - name: Install nginx package)
  - Task: Where you specify which modules to use and which "state" the module should leave your target in.
  - Hosts: The list of servers within your inventory file that you're running your playbook on. (Example: hosts: all << this will run on all hosts in your inventory)
  - Module name: This is the name of the Ansible module you want to use to execute against your host.

Creating Ansible Playbooks

Creating basic Ansible playbooks – Group Lab

- Let's build out our own example playbook, step by step, as a group. Can you spot the name, tasks, hosts and modules? What is this playbook going to do for us?

```
---
- name: Install nginx
  hosts: host.name.ip
  become: true

  tasks:
  - name: Add epel-release repo
    yum:
      name: epel-release
      state: present

  - name: Install nginx
    yum:
      name: nginx
      state: present

```

Creating Ansible Playbooks

Creating basic Ansible playbooks – Group Lab

- Okay, let's break this down. First, we can see that the **name** states plainly what this playbook does. It installs the web service nginx package. Second, based on the **hosts** value we're going to run this on "all" servers in our inventory file.

```
1  ---
2  - name: Install nginx
3    hosts: all
4    tasks:
5    - name: Add epel-release repo
6      yum:
7        name: epel-release
8        state: present
9
10   - name: Install nginx
11     yum:
12       name: nginx
13       state: present
14
15
16
```

Creating Ansible Playbooks

Creating basic Ansible playbooks – Group Lab

- Let's keep building out the playbook. We start getting into the tasks that our playbook is executing. Notice we keep the names of the tasks as clear as we do the playbook name description. Our first task will be to ensure the epel-release repository is present because this is where we're going to get our nginx install packages. You'll notice here that we're using the "yum" module because of the flavor of Linux we're running on.

```
1  ---
2  - name: Install nginx
3    hosts: all
4
5    tasks:
6    - name: Add epel-release repo
7      yum:
8        name: epel-release
9        state: present
10
11   - name: Install nginx
12     yum:
13       name: nginx
14       state: present
15
16
```

Creating Ansible Playbooks

Creating basic Ansible playbooks – Group Lab

- Finally, we execute the install of the nginx package and we tell Ansible to ensure that at the end of the install, the nginx package should exist or be in a “present” state. Once again, we’re using the “yum” module to execute our install.

```
1  ---
2  - name: Install nginx
3    hosts: all
4    tasks:
5    - name: Add epel-release repo
6      yum:
7        name: epel-release
8        state: present
9
10   - name: Install nginx
11     yum:
12       name: nginx
13       state: present
14
15
16
```

Creating Ansible Playbooks

Creating basic Ansible playbooks – Group Lab

- As we mentioned at the beginning of this section, it is best to start small and then iterate on your playbook. So that is what we're going to do. Now that we have the nginx package installed, we want to now start copying in some html files as part of our NGINX web server config.
- In the example on the next slide, you'll notice that we're simply appending to our existing playbook. We're adding a new task with the name of "Insert Index Page" and we're going to use the "template" module to copy the index.html from our local source (src) location to the destination (dest) on the NGINX server.
  NOTE! When copying files to a destination, you must include the file as part of the fully qualified path

Creating Ansible Playbooks

Creating basic Ansible playbooks – Group Lab

```
---
- name: Install nginx
  hosts: host.name.ip
  become: true

  tasks:
  - name: Add epel-release repo
    yum:
      name: epel-release
      state: present

  - name: Install nginx
    yum:
      name: nginx
      state: present

  - name: Insert Index Page
    template:
      src: index.html
      dest: /usr/share/nginx/html/index.html
```

Creating Ansible Playbooks

Creating basic Ansible playbooks – Group Lab

- Okay, let's keep adding to our playbook. Now that we have the nginx package installed and an index.html file copied to the destination location, we want our playbook to actually start the nginx service so that we can hit the web server.

- In the example on the next slide, you'll notice that we're simply appending to our existing playbook. We're adding a new task with the name of "Start NGiNX" and we're going to use the "service" module to start the nginx service. Finally, we tell our Ansible playbook to expect a final state of "started" for the nginx service.

Creating Ansible Playbooks

Creating basic Ansible playbooks – Group Lab

```
17    - name: Insert Index Page
18      template:
19        src: index.html
20        dest: /usr/share/nginx/html/index.html
21
22    - name: Start NGiNX
23      service:
24        name: nginx
25        state: started
26
27
```

Creating Ansible Playbooks

Creating basic Ansible playbooks – Group Lab

- We could continue building on this playbook, adding more tasks in this same fashion. A general rule of thumb or best practice is to not try to do too much in any one playbook. This could cause the entire playbook to fail if any one task fails to execute successfully.

- So now this is what our full Ansible playbook looks like for installing a fully functional NGiNX server with the required nginx packages, index.html file as well as a running nginx service.

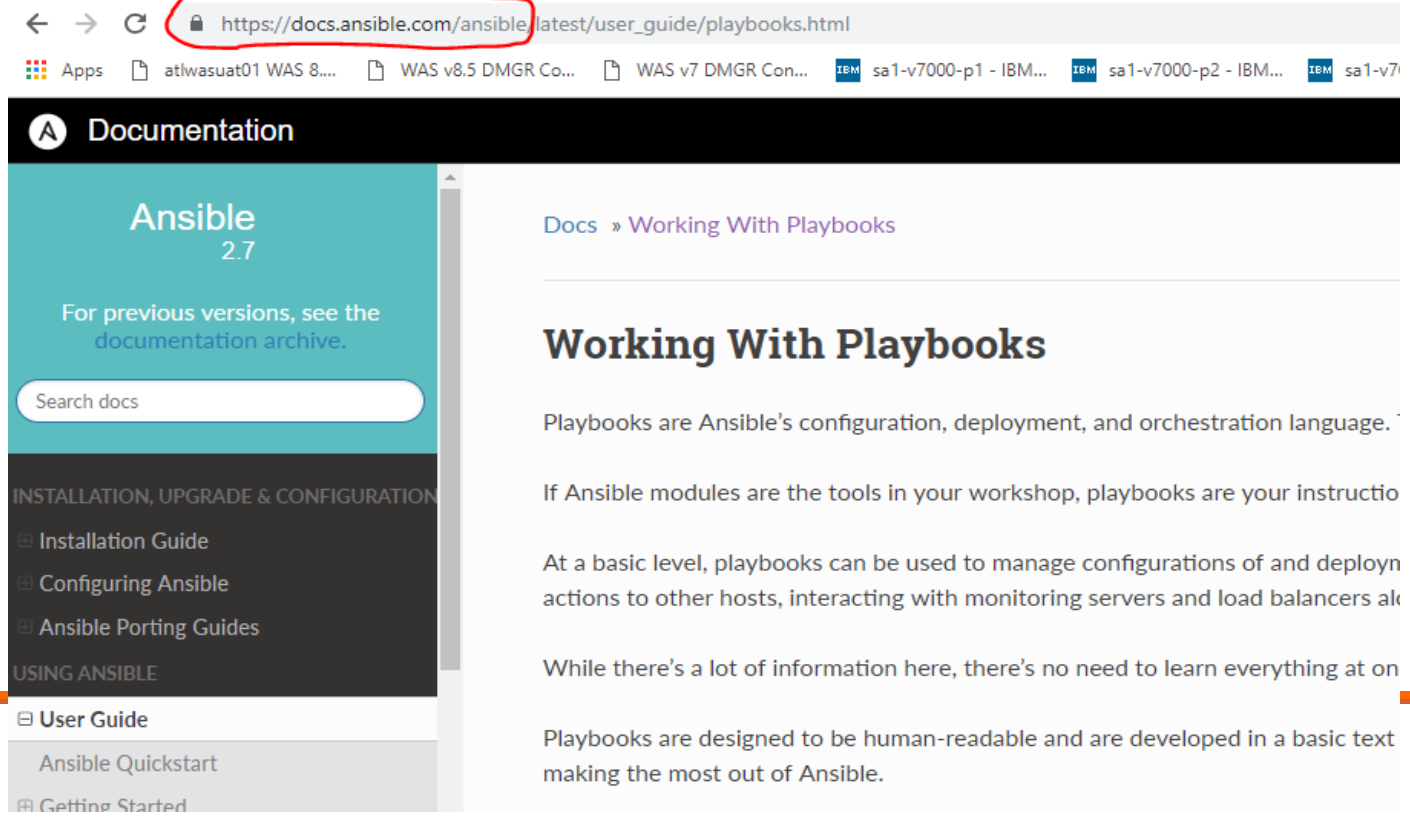
Creating Ansible Playbooks

```
---
- name: Install nginx
  hosts: all


  tasks:
  - name: Add epel-release repo
    yum:
      name: epel-release
      state: present

  - name: Install nginx
    yum:
      name: nginx
      state: present

  - name: Insert Index Page
    template:
      src: index.html
      dest: /usr/share/nginx/html/index.html

  - name: Start NGiNX
    service:
      name: nginx
      state: started

```

Creating Ansible Playbooks

Additional Resources

- For additional resources on how to build out playbooks, you can go to https://www.ansible.com/blog/getting-started-writing-your-first-playbook and https://docs.ansible.com for examples, templates and existing playbooks that are ready for use in your environment.



Keeping IT Going

Module 4

This Section 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
```

- **<u>Other Useful Modules for Testing:</u>**
- **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.**

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.*

Gartner RAS Core research, 2010

Challenge Time

- **Exercise Objective A: Complete two challenges from the back of the lab guide.**

Exercise

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 Information: 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

Module Review: In this Module, you learned…

- Testing, validating and debugging Ansible configurations.
- Common issues and how to find the root cause.
- How to use ServerSpec, & RoleSpec.
- The value of Ansible with CI, CD and Continuous Improvement.
- Orchestration with Ansible.
- Some Ansible case studies that you can review and discuss back at your job.

Revisiting Your Ansible Needs & Next Steps

Module 5

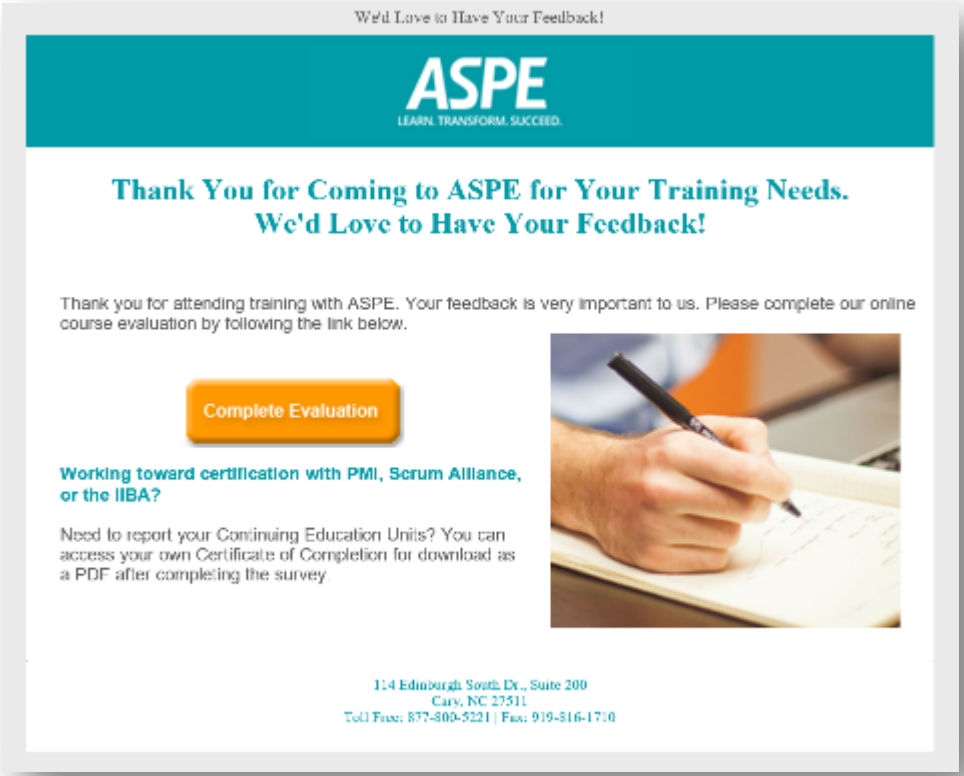
Preparing for Ansible Back at Work

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?**

Feedback Email

- If you haven't already, you will receive an email similar to the one here asking for your feedback on the course.
- This Evaluation will ask your feedback on many aspects of your training, including the course materials, instructor facilitation and labs where applicable.
- We appreciate you completing this survey to help us continue to better our content and deliveries in order to serve you better.



Questions