Red Hat Certified Specialist in Ansible Automation

A Brief Word on YAML

What is YAML and Why Does it Matter?

- YAML is a markup language used for formatting data
- YAML is generally considered easier for humans to read
- Ansible Playbooks use YAML syntax

Basic YAML Syntax

- YAML is composed of key-value pairs, lists, and dictionaries
- Files open with three hyphens on the first line and close with three periods
- List items are designated with a single hyphen and a space
- Each list item should have the same indentation
- Dictionaries are designated with a colon and a space followed by indented key-value pairs

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

```

Multiple Line Values

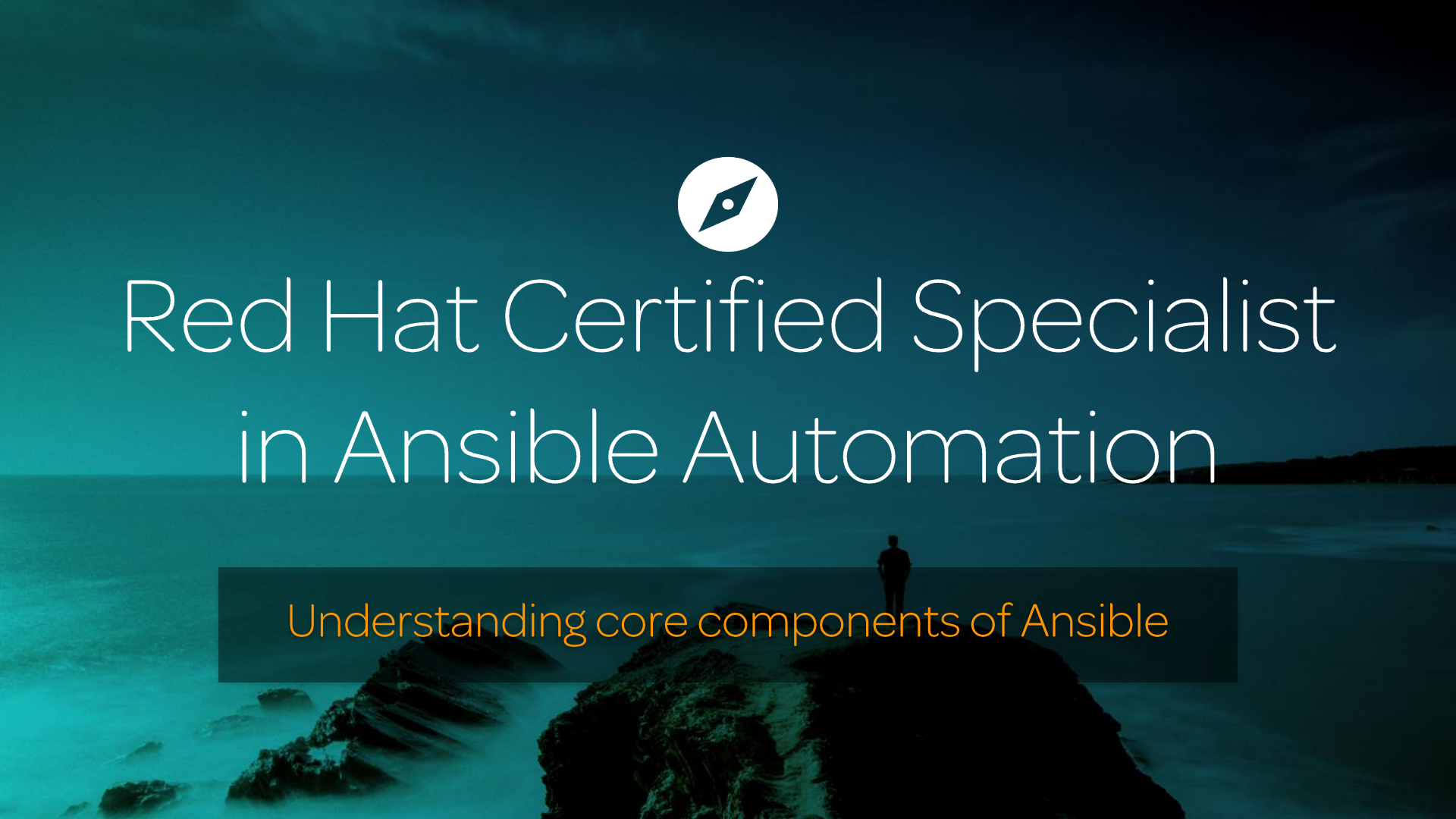
- It is frequently useful when working on Ansible playbooks to format input with line breaks
- YAML has a means of letting us do this
- The pipe and right angle bracket (| and >) may be used to allow for line breaks within YAML
- Pipe will take each line break as part of the input in the data that follows it
- The right-angle bracket will ignore line breaks in the data that follows it
- A typical use case is breaking up parameters for readability

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

```
10    - name: write the apache config file
11      template: >
12        src=/srv/httpd.j2
13        dest=/etc/httpd.conf
14      notify:
15      - restart apache
```

Quotes and When to Use Them

- YAML has a relatively small set of special characters: [ ] { } : > |
- You must use double quotes to escape a special character
- Technically, you only need to escape a colon if it is followed by a space
- You will frequently need to escape curly braces { } as they are special to both YAML and Ansible:
    - Ansible variables are formatted "{{ variable_name }}" – the double quotes are required.
    - This will be discussed more when we go over variables in Ansible.
- Boolean values auto-convert in YAML
- If you want a literal "Yes", you must use quotes
- Floating Point Numbers are taken as numeric unless quoted



Red Hat Certified Specialist in Ansible Automation

Understanding core components of Ansible

Overview

- Inventories
- Modules
- Variable
- Facts
- Plays
- Playbooks
- Configuration Files

Inventories

- Inventories are how Ansible can locate and run against multiple systems
- You can think of an inventory as a list of hosts
- By default, Ansible uses /etc/ansible/hosts as its inventory, but this is configurable
- Inventories may be formatted as an INI file or as a yaml file. Note: file formats are interchangeable

```
mail.example.com

[webservers]
raleigh.example.com
dallas.example.com

[dbservers]
db[1:4].example.com
```

```
all:
  hosts:
    mail.example.com
  children:
    webservers:
      hosts:
        raleigh.example.com
        dallas.example.com
    dbservers:
      hosts:
        db[1:4].example.com
```

Inventories continued

- Inventory files may simply consist of a list of hostnames but can be much more robust
- It is also possible to define groups of hosts, host or group level variables, and groups of groups within the inventory
- There are a number of variables that may be used within the inventory to control how ansible connects to and interacts with target hosts

Break Slide – Demo: Working with Inventories

Modules

- Modules are essentially tools for particular tasks
- Modules can take (and usually do) take parameters
- They return JSON
- Can run from the command line or within a playbook
- There are a significant number of modules for many kinds of work
- Custom modules can be written

Variables

- Variable names should be letters, numbers, and underscores
- Variables should always start with a letter
- Can be scoped by a group, host, or even in a playbook
- Typically used for configuration values and various parameters
- Variables can also be used to store the return value of executed commands
- Ansible variables may also be dictionaries
- There are a number of predefined variables used by Ansible

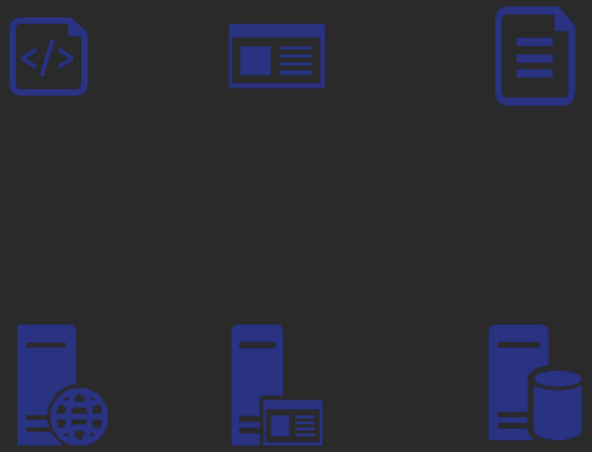
Facts

- Facts provide certain information about a given target host
- Facts are discovered by Ansible automatically when it reaches out to a host
- Fact gathering may be disabled
- Facts may be cached between playbook executions, but this is not default behavior

```
[user@scoldham1 ~]$ ansible -i ansbile/inventory.yaml scoldham6.mylabserver.com -m setup
scoldham6.mylabserver.com | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.31.34.97"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::8ec:3cff:febd:769c"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "08/24/2006",
        "ansible_bios_version": "4.2.amazon",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-693.21.1.el7.x86_64",
            "LANG": "en_US.UTF-8",
```

Plays and Playbooks

- The goal of a play is to map a group of hosts to some well-defined roles
- A play may use one or more modules to achieve a desired end state on a group of hosts
- A playbook is a series of plays
- A playbook may deploy new web servers, install a new application to existing application servers, and run SQL against some database servers to support the new application



Configuration Files

- Several possible locations (in order processed):
  - ANSIBLE_CONFIG (an environment variable)
  - ansible.cfg (in the current directory)
  - .ansible.cfg (in the home directory)
  - /etc/ansible/ansible.cfg
- Configuration can also be set in environment variables
- Some commonly used settings:
  - ansible_managed
  - forks
  - Inventory

Red Hat Certified Specialist in Ansible Automation

Ad-Hoc Ansible Commands

Overview

- What is an ad-hoc command in Ansible?
- Use cases for ad-hoc commands
- Ad-hoc vs Playbook
- Ansible command syntax
- Common modules

What is an ad-hoc command in Ansible?

- You can run ansible either ad-hoc or as a playbook
- Both methods have the same capabilities
- Ad-hoc commands are effectively one-liners

Use cases for Ad-hoc

- Operational commands
  - Checking log contents
  - Daemon control
  - Process management
- Informational commands
  - Check installed software
  - Check system properties
  - Gather system performance information (CPU, disk space, memory use)
- Research
  - Work with unfamiliar modules on test systems
  - Practice for playbook engineering

Ad-hoc vs Playbook

Ad-hoc mode

- Command: ansible
- Effective for one-time commands, operational activities, information gathering, and research
- Similar to a single bash command

Playbook mode

- Command: ansible-playbook
- Effective for deployments, routine tasks, system deployment
- Similar to bash script

Ansible command syntax

```
[user@linuxacademy ~]$ ansible httpd -i ansbile/inv.ini -b -m yum -a "name=httpd state=latest" -f 100
scoldham4.mylabserver.com | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: amazon-id, rhui-lb, search-disabled-repos\nResolving Dependencies\n--> Running transaction check\n---> Package ht
tpd.x86_64 0:2.4.6-67.el7_4.6 will be installed\n--> Processing Dependency: httpd-tools = 2.4.6-67.el7_4.6 for package: httpd-2.4.6-67.el7
_4.6.x86_64\n--> Processing Dependency: /etc/mime.types for package: httpd-2.4.6-67.el7_4.6.x86_64\n--> Processing Dependency: libapr-1.so
.0()(64bit) for package: httpd-2.4.6-67.el7_4.6.x86_64\n--> Processing Dependency: libaprutil-1.so.0()(64bit) for package: httpd-2.4.6-67.
el7_4.6.x86_64\n--> Running transaction check\n---> Package apr.x86_64 0:1.4.8-3.el7_4.1 will be installed\n---> Package apr-util.x86_64 0
:1.5.2-6.el7 will be installed\n---> Package httpd-tools.x86_64 0:2.4.6-67.el7_4.6 will be installed\n---> Package mailcap.noarch 0:2.1.41
-2.el7 will be installed\n--> Finished Dependency Resolution\n\nDependencies Resolved\n\n================================================================
================================\n Package      Arch    Version          Repository                          Size\n================================
================================================\nInstalling:\n httpd        x86_64  2.4.6-67.el7_4.6  rhui-REGION-rhel-server-rel
eases  1.2 M\nInstalling for dependencies:\n apr          x86_64  1.4.8-3.el7_4.1   rhui-REGION-rhel-server-releases  103 k\n apr-util
 x86_64  1.5.2-6.el7       rhui-REGION-rhel-server-releases   92 k\n httpd-tools  x86_64  2.4.6-67.el7_4.6  rhui-REGION-rhel-server-releas
es   88 k\n mailcap      noarch  2.1.41-2.el7      rhui-REGION-rhel-server-releases   31 k\n\nTransaction Summary\n================================
================================================\nInstall  1 Package (+4 Dependent packages)\n\nTotal download size: 1.5 M\nInsta
lled size: 4.3 M\nDownloading packages:\n--------------------------------------------------------------------------------\nTotal
                                           2.3 MB/s | 1.5 MB  00:00     \nRunning transaction check\nRunning transaction test\nTransaction test s
ucceeded\nRunning transaction\n  Installing : apr-1.4.8-3.el7_4.1.x86_64                                 1/5 \n  Installing : apr-util-1
.5.2-6.el7.x86_64                                2/5 \n  Installing : httpd-tools-2.4.6-67.el7_4.6.x86_64                                3/5 \
n  Installing : mailcap-2.1.41-2.el7.noarch                                4/5 \n  Installing : httpd-2.4.6-67.el7_4.6.x86_64
                  5/5 \n  Verifying  : mailcap-2.1.41-2.el7.noarch                                1/5 \n  Verifying  : httpd-2.4.6-67
.el7_4.6.x86_64                                2/5 \n  Verifying  : apr-1.4.8-3.el7_4.1.x86_64                                3/5 \n  V
erifying  : apr-util-1.5.2-6.el7.x86_64                                4/5 \n  Verifying  : httpd-tools-2.4.6-67.el7_4.6.x86_64
               5/5 \n\nInstalled:\n  httpd.x86_64 0:2.4.6-67.el7_4.6                                                \n\nDependency Instal
led:\n  apr.x86_64 0:1.4.8-3.el7_4.1              apr-util.x86_64 0:1.5.2-6.el7     \n  httpd-tools.x86_64 0:2.4.6-67.el7_4.6      mailca
p.noarch 0:2.1.41-2.el7     \n\nComplete!\n"
    ]
}
```

Common Modules

Ping	Setup	Yum
Validate server is up and reachable No required parameters	Gather ansible facts No required parameters	Use yum package manager name

Common Modules

Service	User	Copy
Control Daemons name	Manipulate system users name	Copy files src and dest

Common Modules

File	Git	
Work with files path	Interact with git repositories repo and dest	

Review

- What is an ad-hoc command in Ansible?
- Use cases for ad-hoc commands
- Ad-hoc vs Playbook
- Ansible command syntax
- Common modules

Red Hat Certified Specialist in Ansible Automation

Inventory Management

Overview

- Use both static and dynamic inventories to define groups of hosts:
  - What is the inventory
  - File formats
  - Static vs. Dynamic
  - Variables and inventories
- Utilize an existing dynamic inventory script:
  - On dynamic inventories
  - Some popular options

What is an Inventory

- An inventory is a list of hosts that Ansible manages
- Inventory location may be specified as follows:
    - Default: /etc/ansible/hosts
    - Specified by CLI: ansible -i <filename>
    - Can be set in ansible.cfg
- The inventory file may contain hosts, patterns, groups, and variables
- You may specify the inventory as a directory containing a series of inventory files (both static and dynamic)
- The inventory may be specified in YAML or INI format
- Can be static or dynamic

File Formats

- YAML
- INI

yaml_inventory_example_com

```yaml
---
all:
  hosts:
    mail.example.com
      ansible_port: 5556
      ansible_host: 192.168.0.10
  children:
    webservers:
      hosts:
        httpd1.example.com
        httpd2.example.com
      vars:
        http_port: 8080
    labservers:
      hosts:
        lab[01:99].example.com
...
```

ini_inventory_example_com

```ini
mail.example.com ansible_port=5556 ansible_host=192.168.0.10

[webservers]
httpd1.example.com
httpd2.example.com

[webservers:vars]
http_port=8080

[labservers]
lab[01:99].example.com
```

Static vs. Dynamic

Static:

- INI or YAML format
- Maintained by hand
- Easy to manage for static configuration

Dynamic:

- Executable (bash script, python script, etc.)
- Script returns JSON containing inventory information
- Good for use with cloud resources subject to sudden change

Variables and Inventories

- Ansible recommends that variables not be defined in inventory files:
  - Should be stored in YAML files located relative to inventory file
  - group_vars
  - host_vars
- Files named by host or group and may end in yml or yaml

```
[user@scoldham1 inventory]$ cat inventory
mail.example.com

[httpd]
httpd1.example.com
httpd2.example.com
[user@scoldham1 inventory]$
```

```
[user@scoldham1 inventory]$ cat group_vars/httpd
http_port: 8080
[user@scoldham1 inventory]$ cat host_vars/httpd1.example.com
opt_dir: /opt
```

```
[user@scoldham1 inventory]$ ls -lR
.:
total 4
drwxrwxr-x. 2 user user 18 Mar 27 15:54 group_vars
drwxrwxr-x. 2 user user 31 Mar 27 15:55 host_vars
-rw-rw-r--. 1 user user 64 Mar 27 15:56 inventory

./group_vars:
total 4
-rw-rw-r--. 1 user user 16 Mar 27 15:54 httpd

./host_vars:
total 4
-rw-rw-r--. 1 user user 14 Mar 27 15:55 httpd1.example.com
[user@scoldham1 inventory]$
```

On Dynamic Inventories

- Specifying an executable file as the inventory is considered a dynamic inventory
- JSON output is expected to be returned to STDOUT from the executable
- The implementation (python, java, C, bash, etc) does not matter as long as the program can run on the control host and behaves as Ansible expects
- The program/script must respond to two possible parameters:
    - --list
    - --host [hostname]

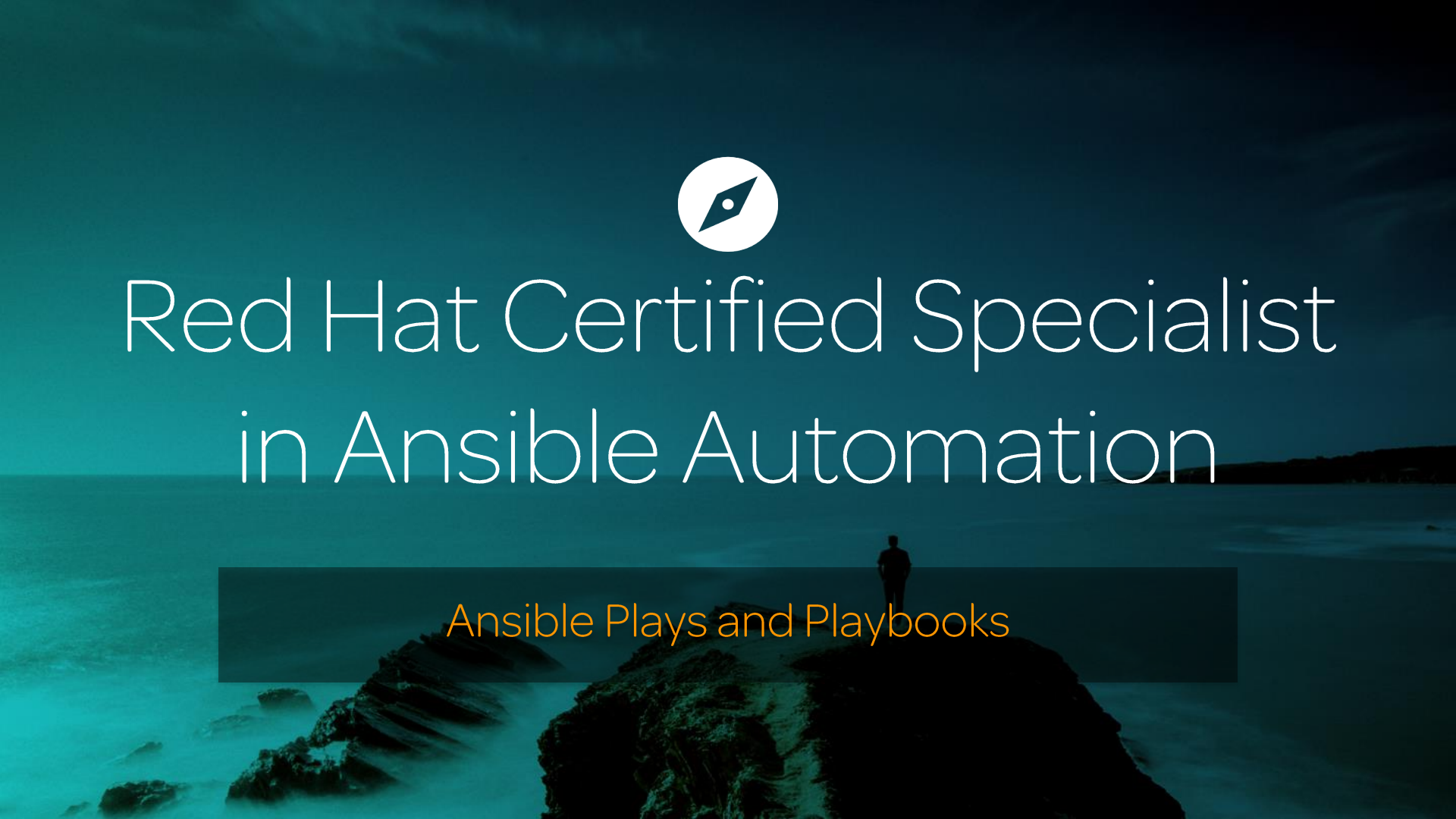
- The program script must return JSON in the format Ansible is expecting
- Do not forget to make the file executable:
    - chmod +x dynamic.py
- Using dynamic inventories, you can pull inventory information from the likes of:
    - A cloud provider
    - LDAP
    - Cobbler
    - Other CMDB software

Some Popular Options

- AWS EC2
- Rackspace
- OpenStack
- Plenty of documentation available

Review

- Use both static and dynamic inventories to define groups of hosts:
  - What is the inventory
  - Static vs. Dynamic
  - File formats
  - Variables and inventories
- Utilize an existing dynamic inventory script:
  - On dynamic inventories
  - Some popular options



Red Hat Certified Specialist in Ansible Automation

Ansible Plays and Playbooks

Overview

- Know how to work with commonly used Ansible modules
- Create playbooks to configure systems to a specified state
- Use variables to retrieve the results of running commands
- Use conditionals to control play execution
- Configure error handling
- Selectively run specified tasks in playbooks using tags

Know how to work with Commonly Used Ansible Modules

- Core modules to be familiar with:
  - Working with files: copy, archive, unarchive, get_url
  - user, group
  - ping
  - service
  - yum
- Lineinfile module
- htpasswd
- Shell and command modules
- Script module
- Debug module
- See http://docs.ansible.com/ansible/latest/modules/modules_by_category.html

Create Playbooks to Configure Systems to a Specified State

- Reviewing plays and playbooks:
  - Plays map a group of hosts to well-defined roles
  - Playbooks are used to orchestrate more complex activities, such as system or application deployment
- Brush up on YAML
- Basic Syntax
- Idempotency
- Pro tips:
  - Retry
  - Limit
  - Watch out for spaces

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

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

- hosts: databases
  remote_user: root

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

```

Use Variables to Retrieve the Results of Running Commands

- The register keyword
- May be referenced within the play
- Many attributes returned

```
[user@scoldham1 ansbile]$ cat reg.yml
---
- hosts: scoldham2.mylabserver.com
  tasks:
    - name: copy a file
      copy:
        src: testfile
        dest: /home/user/testregister
        mode: 400
      register: var
    - name: output debug info
      debug: msg="debug info is {{ var }}"
```

```
[user@scoldham1 ansbile]$ ansible-playbook reg.yml

PLAY [scoldham2.mylabserver.com] ***********************************************

TASK [Gathering Facts] *********************************************************
ok: [scoldham2.mylabserver.com]

TASK [copy a file] *************************************************************
changed: [scoldham2.mylabserver.com]

TASK [output debug info] *******************************************************
ok: [scoldham2.mylabserver.com] => {
    "msg": "debug info is {u'src': u'/home/user/.ansible/tmp/ansible-tmp-152234
9b8192783b78de27040d4f8b1c', u'group': u'user', u'uid': 1001, u'dest': u'/home/
5c7c7156197bd69ccea3020', u'changed': True, 'failed': False, u'state': u'file',
:user_home_t:s0', u'mode': u'0620', u'owner': u'user', 'diff': [], u'size': 15}
}

PLAY RECAP *********************************************************************
scoldham2.mylabserver.com  : ok=3    changed=1    unreachable=0    failed=0
```

Use Conditionals to Control Play Execution

```
- name: template configuration file
  template:
    src: template.j2
    dest: /etc/foo.conf
  notify:
    - roll web
```

```
handlers:
- name: restart apache
    service:
      name: apache
      state: restarted
  listen: "roll web"
```

Use Conditionals to Control Play Execution

- When
- With_items
- With_files

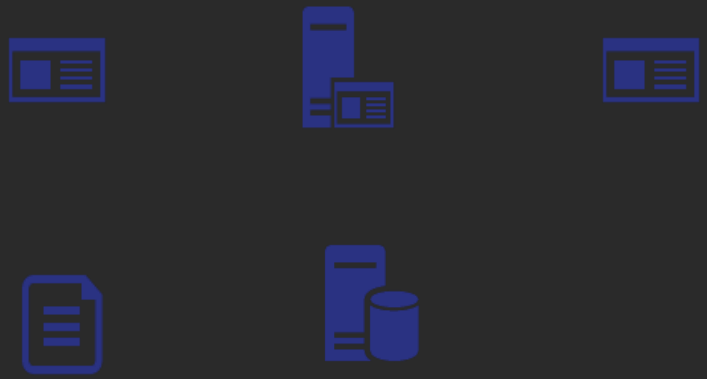
```
---
- hosts: all
  tasks:
    - name: copy files
      copy:
        src: "{{item}}"
        dest: "/home/user/{{item}}"
        mode: 400
      with_items:
        - file1
        - file2
        - file3
```

Configure Error Handling

- Ignoring acceptable errors
- Defining failure conditions
- Defining "Changed"
- Blocks

Selectively Run Specified Tasks in Playbooks Using Tags

- How Ansible uses tags



Red Hat Certified Specialist in Ansible Automation

Ansible Templates

Overview

- Template basics
- Template module
- Template file

Template Basics

- Templates give the ability to provide a skeletal file that can be dynamically completed using variables
- The most common template use case is configuration file management
- Templates are generally used by providing a template file on the ansible control node, and then using the template module within your playbook to deploy the file to a target server or group
- Templates are processed using the Jinja2 template language

Template Module

- The template module is used to deploy template files
- There are two required parameters:
  - src – the template to use (on the ansible control host)
  - dest – where the resulting file should be located (on the target host)
- A useful optional parameter is validate which requires a successful validation command to run against the result file prior to deployment
- It is also possible to set basic file properties using the template module

```
---
- hosts: webservers

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

Template File

- Template file are essentially little more than text files
- Template files are designated with a file extension of .j2
- Template files have access to the same variables that the play that calls them does

```
#Apache HTTP server -- main configuration
#
# {{ ansible_managed }}

 ## General configuration
 ServerRoot {{ httpd_ServerRoot }}
 Listen {{ httpd_Listen }}

 Include conf.modules.d/*.conf

 User apache
 Group apache

 ## 'Main' server configuration
 ServerAdmin {{ httpd_ServerAdmin }}
 {% if httpd_ServerName is defined %}
 ServerName {{ httpd_ServerName }}
 {% endif %}

 DocumentRoot {{ httpd_DocumentRoot }}
```

Review

- Template basics
- Template module
- Template file

Red Hat Certified Specialist in Ansible Automation

Variables and Facts

Overview

- Ansible variables
- Dictionary variables
- Magic variables and filters
- What are facts?
- How to use facts
- Facts.d – custom facts

Ansible Variables

- Review on naming convention and quotes
- Some more places to define variables:
    - vars, vars_files, and vars_prompt
    - Command line: ansible-playbook play.yml –e '{"myVar":"myValue","anotherVar":"anotherValue"}'
    - Roles, blocks, and inventories
- Essential variable use:
    - - debug: msg="Look! I'm using my variable {{ myVar }} !"
- A note on quotes:
    - name: "{{ package }}"

Dictionary Variables

- Yaml formatting allows for python style dictionaries to be used as variables
- There are two formats to access dictionary values:
    - employee['name']
    - employee.name
- The bracket syntax is safer as the dot notation can have collisions with python in certain circumstances

```
employee:
    name: bob
    id: 42
```

Magic Variables and Filters

- Ansible defines several special variables knowns as magic variables
- You can use the variable hostvars to look at facts about other hosts in the inventory
- There is also a groups variable that provides inventory information
- Jinja2 filters can be useful in manipulating text format
- See http://jinja.pocoo.org/docs/2.10/templates/#builtin-filters

{{ hostvars['node1']['ansible_distribution'] }}

{{ groups['webservers'] }}

{{ groups['webservers']|join(' ') }}

What are facts

- Facts are information discovered by Ansible about a target system
- There are two ways facts are collected:
    - Using the setup module with an ad-hoc command: ansible all –m setup
    - Facts are gathered by default when a playbook is executed
- Fact gathering in playbooks may be disabled using the gather_facts attribute

How to use facts

- Any collected facts, they may be accessed through variables:
  - {{ ansible_defalt_ipv4.address }}
- It is possible to use filters with regex, in ad-hoc mode, to match certain fact names
- Facts may also be used with conditionals to have plays behave differently on hosts that meet certain criteria

```
[user@scoldham1 ~]$ ansible scoldham2 -m setup -a "filter=*ipv4*"
scoldham2 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.31.109.20"
        ],
        "ansible_default_ipv4": {
            "address": "172.31.109.20",
            "alias": "eth0",
            "broadcast": "172.31.111.255",
            "gateway": "172.31.96.1",
            "interface": "eth0",
            "macaddress": "12:15:31:16:f0:68",
            "mtu": 9001,
            "netmask": "255.255.240.0",
            "network": "172.31.96.0",
            "type": "ether"
        }
    },
    "changed": false
}
```

Facts.d – custom facts

- It is possible to define custom facts on your servers using the facts.d directory
- Create /etc/ansible/facts.d directory on target system. All valid files within this directory ending in .fact are returned under ansible_local with facts
- Fact files may be INI, JSON, or an executable that returns JSON

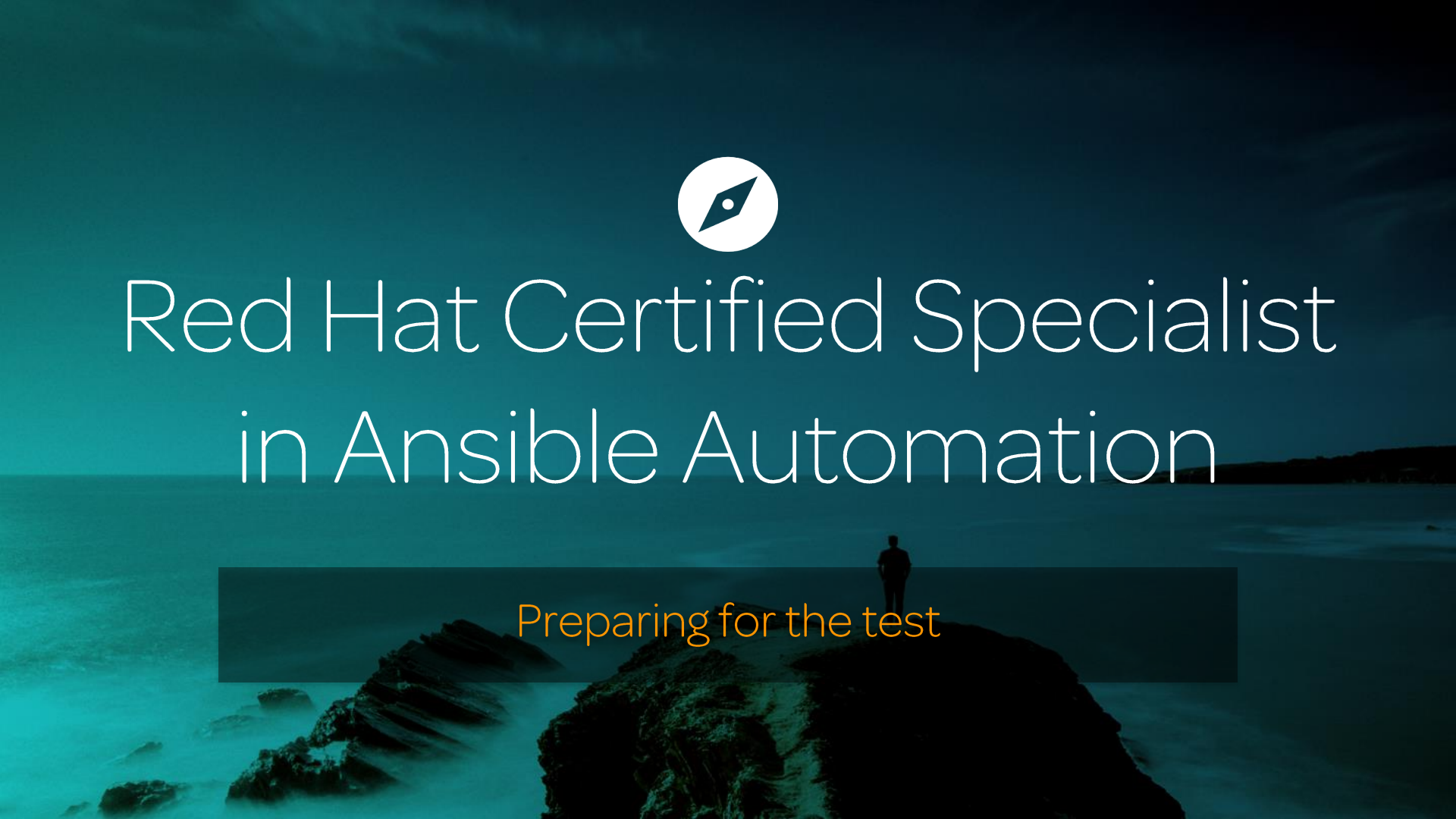
```
[user@scoldham2 ~]$ tree -A /etc/ansible/
/etc/ansible/
└── facts.d
    ├── data.fact
    ├── prefs.fact
    └── software_inv.fact

1 directory, 3 files
```

```
[user@scoldham1 ~]$ ansible scoldham2 -m setup -a "filter=ansible_local"
scoldham2 | SUCCESS => {
    "ansible_facts": {
        "ansible_local": {
            "data": {},
            "prefs": {},
            "software_inv": {
                "software": {
                    "adobe": "installed",
                    "dreamweaver": "uninstalled",
                    "office": "installed"
                }
            }
        }
    },
    "changed": false
}
```

Review

- Ansible variables
- Dictionary variables
- Magic variables and filters
- What are facts?
- How to use facts
- Facts.d – custom facts



Red Hat Certified Specialist in Ansible Automation

Preparing for the test

Preparing for the test

- Tools for preparation

- Quizzes
- Flash Cards
- Study Guide
- Learning Activities
- Interactive Diagram


Red Hat Certified Specialist
in Ansible Automation
What's Next?

After Certification

- Go further with Ansible

- Ansible and Amazon Web Services
- Deploying to AWS with Ansible and Terraform
- Using Ansible for Configuration Management and Deployments

- Red Hat Certified Architect Path - Certified Specialist Exams:

- OpenStack
- Platform-as-a-Service (OpenShift)
- Red Hat Virtualization
- Server Security and Hardening