Ansible Windows Workshop

Introduction to Ansible Automation for Windows
Housekeeping

- Timing
- Breaks
- Takeaways
What you will learn

- Introduction to Ansible automation
- How Ansible works for Windows automation
- Understanding Ansible modules and playbooks
- Using Ansible Tower to scale automation to the enterprise
- Reusing automation with Ansible Roles
Introduction

Topics Covered:

- Why Automate?
- How Ansible Windows Automation works
- Understanding Inventory
- An example Ansible Playbook
Automation happens when one person meets a problem they never want to solve again
Teams are automating...

Lines Of Business
Network
Security
Operations
Developers
Infrastructure
Ad-hoc Automation is happening in silos

- Ansible used in silo
- DIY scripting automation
- Open source config management tool
- Proprietary vendor supplied automation

Is organic automation enough?
Why Ansible?

Simple
- Human readable automation
- No special coding skills needed
- Tasks executed in order
- Usable by every team
- Get productive quickly

Powerful
- App deployment
- Configuration management
- Workflow orchestration
- Network automation
- Orchestrate the app lifecycle

Agentless
- Agentless architecture
- Uses OpenSSH & WinRM
- No agents to exploit or update
- Get started immediately
- More efficient & more secure
### What can I do using Ansible?

Automate the deployment and management of your entire IT footprint.

**Do this...**

- Orchestration
- Configuration Management
- Application Deployment
- Provisioning
- Continuous Delivery
- Security and Compliance

**On these...**

- Firewalls
- Load Balancers
- Applications
- Containers
- Clouds
- Servers
- Infrastructure
- Storage
- Network Devices
- And more...
When automation crosses teams, you need an automation platform.
Red Hat Ansible Automation Platform

Network | Lines of business | Security | Operations | Infrastructure | Developers
--- | --- | --- | --- | --- | ---

Engage | **Ansible Hosted Services**: Engage users with an automation focused experience

Scale | **Ansible Tower**: Operate & control at scale

Create | **Ansible Engine**: Universal language of automation

Fueled by an open source community
Red Hat Ansible Tower  
by the numbers:

94%  
Reduction in recovery time following a security incident

84%  
Savings by deploying workloads to generic systems appliances using Ansible Tower

67%  
Reduction in man hours required for customer deliveries

Financial summary:

146%  
ROI on Ansible Tower

<3 MONTHS  
Payback on Ansible Tower

SOURCE: "The Total Economic Impact™ Of Red Hat Ansible Tower, a June 2018 commissioned study conducted by Forrester Consulting on behalf of Red Hat."  
WINDOWS AUTOMATION

90+ Windows Modules

1,300+ Powershell DSC resources

ansible.com/windows
WHAT CAN I DO USING ANSIBLE FOR WINDOWS

Native Windows support uses PowerShell remoting to manage Windows in the same Ansible agentless way

- Install and uninstall MSIs
- Gather facts on Windows hosts
- Enable and disable Windows features
- Start, stop, and manage Windows Services
- Create and Manage local users and groups
- Manage Windows packages via Chocolatey package manager
- Manage and install Windows updates
- Fetch files from remote sites
- Push and execute any Powershell scripts
**Ansible automates technologies you use**

Time to automate is measured in minutes, **50+ certified** platforms

<table>
<thead>
<tr>
<th>Cloud</th>
<th>Virt &amp; Container</th>
<th>Windows</th>
<th>Network</th>
<th>Security</th>
<th>Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>Docker</td>
<td>ACLs</td>
<td>Arista</td>
<td>Checkpoint</td>
<td>Dynatrace</td>
</tr>
<tr>
<td>Azure</td>
<td>Kubernetes</td>
<td>Files</td>
<td>Aruba</td>
<td>Cisco</td>
<td>Datadog</td>
</tr>
<tr>
<td>Digital Ocean</td>
<td>OpenStack</td>
<td>Packages</td>
<td>Bigswitch</td>
<td>CyberArk</td>
<td>LogicMonitor</td>
</tr>
<tr>
<td>Google</td>
<td>OpenShift</td>
<td>IIS</td>
<td>Cisco</td>
<td>Fortinet</td>
<td>New Relic</td>
</tr>
<tr>
<td>OpenStack</td>
<td>VMware</td>
<td>Registry</td>
<td>Ericsson</td>
<td>Juniper</td>
<td>F5</td>
</tr>
<tr>
<td>Rackspace</td>
<td></td>
<td>Shares</td>
<td>FRR</td>
<td>IBM</td>
<td>Sensu</td>
</tr>
<tr>
<td>+more</td>
<td></td>
<td>Services</td>
<td>Juniper</td>
<td>Palo Alto</td>
<td>+more</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Configs</td>
<td>Meraki</td>
<td>Snort</td>
<td>Devops</td>
</tr>
<tr>
<td><strong>Red Hat</strong></td>
<td><strong>Storage</strong></td>
<td><strong>Users</strong></td>
<td><strong>OpenvSwitch</strong></td>
<td><strong>Jira</strong></td>
<td><strong>Jira</strong></td>
</tr>
<tr>
<td>Products</td>
<td>Infinidat</td>
<td><strong>Domains</strong></td>
<td>Ruckus</td>
<td>GitHub</td>
<td><strong>GitHub</strong></td>
</tr>
<tr>
<td>RHEL</td>
<td>Netapp</td>
<td><strong>Updates</strong></td>
<td>VyOS</td>
<td>Vagrant</td>
<td><strong>Vagrant</strong></td>
</tr>
<tr>
<td>Satellite</td>
<td>Pure Storage</td>
<td>+more</td>
<td>+more</td>
<td>Jenkins</td>
<td><strong>Jenkins</strong></td>
</tr>
<tr>
<td>Insights</td>
<td>+more</td>
<td></td>
<td></td>
<td>Slack</td>
<td><strong>Slack</strong></td>
</tr>
<tr>
<td>+more</td>
<td></td>
<td></td>
<td></td>
<td>+more</td>
<td>+more</td>
</tr>
</tbody>
</table>

+more
PLAYBOOKS ARE WRITTEN IN YAML
Tasks are executed sequentially
Invoke Ansible modules
---
- name: start IIS/stop firewall
  hosts: windows-web
  become: yes
  tasks:

- name: IIS is running
  win_service:
    name: W3Svc
    state: running

- name: firewall service is stopped/disabled
  win_service:
    name: MpsSvc
    state: stopped
    start_mode: disabled
- **name**: Start the SNMP service
  - **win_service**:
    - **name**: SNMP
    - **state**: started
Modules do the actual work in Ansible, they are what gets executed in each playbook task.

- Written in Powershell
- Modules can be idempotent
- Modules take user input in the form of parameters

```yaml
tasks:
  - name: start IIS
    win_service:
      name: W3Svc
      state: running
```
Windows modules

Ansible modules for Windows automation typically begin with `win_*`

- **win_copy** - Copies files to remote locations on windows hosts
- **win_service** - Manage and query Windows services
- **win_domain** - Ensures the existence of a Windows domain
- **win_reboot** - Reboot a windows machine
- **win_regedit** - win_regedit – Add, change, or remove registry keys and values
- **win_ping** - A windows version of the classic ping module
- **win_dsc** - Invokes a PowerShell DSC configuration
- **win_acl** - Set file/directory/registry permissions for a system user or group
PLUGINS ARE "GEARS IN THE ENGINE"
Code that plugs into the core engine
Adaptability for various uses & platforms

{{ some_variable | to_nice_yaml }}
List of systems in your infrastructure that automation is executed against

[web]
webserver1.example.com
webserver2.example.com

[db]
dbserver1.example.com

[switches]
leaf01.internal.com
leaf02.internal.com

[firewalls]
checkpoint01.internal.com

[lb]
f5-01.internal.com
CLOUD
Red Hat Openstack, Red Hat Satellite, VMware, AWS EC2, Rackspace, Google Compute Engine, Azure
CMDB
ServiceNow, Cobbler, BMC, Custom cmdb
AUTOMATE EVERYTHING
Red Hat Enterprise Linux, Cisco routers, Arista switches, Juniper routers, Windows hosts, Check Point firewalls, NetApp storage, F5 load balancers and more
Tower Introduction

Topics Covered:

- What is Ansible Tower?
- Job Templates
  - Inventory
  - Credentials
  - Projects
Ansible Tower is a UI and RESTful API allowing you to scale IT automation, manage complex deployments and speed productivity.

• Role-based access control

• Deploy entire applications with push-button deployment access

• All automations are centrally logged

• Powerful workflows match your IT processes
Red Hat Ansible Automation Platform

Engage Ansible SaaS: Engage users with an automation focused experience

Create Ansible Engine: Universal language of automation

Fueled by an open source community

Scale
- Control: Web UI and API
- Delegation: Role Based Access Controls
- Scale: Scalable Execution Capacity

Lines of business:
- Network
- Operations
- Security
- Infrastructure
- Developers
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push button</td>
<td>An intuitive user interface experience makes it easy for novice users to execute playbooks you allow them access to.</td>
</tr>
<tr>
<td>RESTful API</td>
<td>With an API first mentality every feature and function of Tower can be API driven. Allow seamless integration with other tools like ServiceNow and Infoblox.</td>
</tr>
<tr>
<td>RBAC</td>
<td>Allow restricting playbook access to authorized users. One team can use playbooks in check mode (read-only) while others have full administrative abilities.</td>
</tr>
<tr>
<td>Centralized logging</td>
<td>All automation activity is securely logged. Who ran it, how they customized it, what it did, where it happened - all securely stored and viewable later, or exported through Ansible Tower’s API.</td>
</tr>
<tr>
<td>Workflows</td>
<td>Ansible Tower’s multi-playbook workflows chain any number of playbooks, regardless of whether they use different inventories, run as different users, run at once or utilize different credentials.</td>
</tr>
</tbody>
</table>
Everything in Ansible Tower revolves around the concept of a **Job Template**. Job Templates allow Ansible Playbooks to be controlled, delegated and scaled for an organization.

Job templates also encourage the reuse of Ansible Playbook content and collaboration between teams.

A **Job Template** requires:
- An **Inventory** to run the job against
- A **Credential** to login to devices.
- A **Project** which contains Ansible Playbooks
Inventory

Inventory is a collection of hosts (nodes) with associated data and groupings that Ansible Tower can connect to and manage.

- Hosts (nodes)
- Groups
- Inventory-specific data (variables)
- Static or dynamic sources
Credentials

Credentials are utilized by Ansible Tower for authentication with various external resources:

- Connecting to remote machines to run jobs
- Syncing with inventory sources
- Importing project content from version control systems
- Connecting to and managing network devices

Centralized management of various credentials allows end users to leverage a secret without ever exposing that secret to them.
A project is a logical collection of Ansible Playbooks, represented in Ansible Tower.

You can manage Ansible Playbooks and playbook directories by placing them in a source code management system supported by Ansible Tower, including Git, Subversion, and Mercurial.
Exercise 1

- Configuring Ansible Tower
Ad-hoc Commands

Topics Covered:

- What are ad-hoc commands
- Common options
- Run from
  - Command line
  - Ansible Tower
An ad-hoc command is a single Ansible task to perform quickly, but don’t want to save for later.
Ad-hoc Commands: Common Options

- **-m MODULE_NAME, --module-name=MODULE_NAME**
  Module name to execute the ad-hoc command

- **-a MODULE_ARGS, --args=MODULE_ARGS**
  Module arguments for the ad-hoc command

- **-b, --become**
  Run ad-hoc command with elevated rights such as sudo, the default method

- **-e EXTRA_VARS, --extra-vars=EXTRA_VARS**
  Set additional variables as key=value or YAML/JSON

- **--version**
  Display the version of Ansible

- **--help**
  Display the MAN page for the Ansible tool
Ad-hoc Commands

# check all my inventory hosts are ready to be managed by Ansible
$ ansible all -m ping

# collect and display the discovered facts for the localhost
$ ansible localhost -m setup

# run the uptime command on all hosts in the web group
$ ansible web -m command -a "uptime"
Ad-hoc Commands from Tower

SNOW DYNAMIC INVENTORY

Select an inventory source by clicking the check box beside it. The inventory source can be a single host or a selection of multiple hosts.

HOSTS

server01.rhdemo.io

server02.rhdemo.io
Exercise 2

- Ad-hoc Commands
Playbooks

Topics Covered:

- Variables
  - Facts
  - Precedence
- Tasks
  - Handlers
Variables

Ansible can work with metadata from various sources and manage their context in the form of variables.

- Command line parameters
- Plays and tasks
- Files
- Inventory
- Discovered facts
- Roles
Facts are bits of information derived from examining a host systems that are stored as variables for later use in a play.

```bash
$ ansible localhost -m setup
localhost | success >> {
    "ansible_facts": {
        "ansible_default_ipv4": {
            "address": "192.168.1.37",
            "alias": "wlan0",
            "gateway": "192.168.1.1",
            "interface": "wlan0",
            "macaddress": "c4:85:08:3b:a9:16",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "192.168.1.0",
            "type": "ether"
        }
    }
}
```
Variable Precedence

The order in which the same variable from different sources will override each other.

1. command line values (eg “-u user”)
2. role defaults [1]
3. inventory file or script group vars [2]
4. inventory group_vars/all [3]
5. playbook group_vars/all [3]
6. inventory group_vars/* [3]
7. playbook group_vars/* [3]
8. inventory file or script host vars [2]
9. inventory host_vars/* [3]
10. playbook host_vars/* [3]
11. host facts / cached set_facts [4]
12. play vars
13. play vars_prompt
14. play vars_files
15. role vars (defined in role/vars/main.yml)
16. block vars (only for tasks in block)
17. task vars (only for the task)
18. include_vars
19. set_facts / registered vars
20. role (and include_role) params
21. include params
22. extra vars (always win precedence)
Tasks

Tasks are the application of a module to perform a specific unit of work.

- **win_file**: A directory should exist
- **win_package**: A package should be installed
- **win_service**: A service should be running
- **win_template**: Render a configuration file from a template
- **win_get_url**: Fetch an archive file from a URL
- **win_copy**: Copy a file from your repository or a remote source
Tasks

tasks:
- name: Ensure IIS Server is present
  win_feature:
    name: Web-Server
    state: present

- name: Ensure latest index.html file is present
  win_copy:
    src: files/index.html
    dest: c:\www\

- name: Restart IIS
  win_service:
    name: IIS Admin Service
    state: restarted
Handler Tasks

Handlers are special tasks that run at the end of a play if notified by another task when a change occurs.

*If a package gets installed or updated, notify a service restart task that it needs to run.*
Handler Tasks

tasks:
- name: Ensure IIS Server is present
  win_feature:
    name: Web-Server
    state: present
  notify: Restart IIS

- name: Ensure latest index.html file is present
  win_copy:
    src: files/index.html
    dest: c:\www\n
handlers:
- name: Restart IIS
  win_service:
    name: IIS Admin Service
    state: restarted
Plays and playbooks

Plays are ordered sets of tasks to execute against host selections from your inventory. A playbook is a file containing one or more plays.
Plays and playbooks

---
- **name**: Ensure IIS is installed and started
  
  **hosts**: web
  
  **become**: yes

  **vars**:
  
  - **service_name**: IIS Admin Service

  **tasks**:
  - **name**: Ensure IIS Server is present
    
    **win_feature**:
    
    - **name**: Web-Server
      
      **state**: present

  - **name**: Ensure latest index.html file is present
    
    **win_copy**:
    
    - **src**: files/index.html
      
      **dest**: c:\www\

  - **name**: Ensure IIS is started
    
    **win_service**:
    
    - **name**: "{{ service_name }}"
      
      **state**: started
---
- **name**: Ensure IIS is installed and started
  
  *hosts*: web
  
  *become*: yes
  
  *vars*:
  
  - *service_name*: IIS Admin Service

**tasks**:
- **name**: Ensure IIS Server is present
  
  *win_feature*:
  
  - *name*: Web-Server
  
  - *state*: present

- **name**: Ensure latest index.html file is present
  
  *win_copy*:
  
  - *src*: files/index.html
  
  - *dest*: c:\www\n
- **name**: Ensure IIS is started
  
  *win_service*:
  
  - *name*: "{{ service_name }}"
  
  - *state*: started
- name: Ensure IIS is installed and started
  hosts: web
  become: yes
  vars:
    service_name: IIS Admin Service
  tasks:
  - name: Ensure IIS Server is present
    win_feature:
      name: Web-Server
      state: present
  - name: Ensure latest index.html file is present
    win_copy:
      src: files/index.html
      dest: c:\www\
  - name: Ensure IIS is started
    win_service:
      name: "{{ service_name }}"
      state: started
Privilege escalation

---

- **name**: Ensure IIS is installed and started  
  **hosts**: web  
  **become**: yes  
  **vars**:  
    - service_name: IIS Admin Service

**tasks**:  
- **name**: Ensure IIS Server is present  
  **win_feature**:  
    - name: Web-Server  
    - state: present

- **name**: Ensure latest index.html file is present  
  **win_copy**:  
    - src: files/index.html  
    - dest: c:\www\

- **name**: Ensure IIS is started  
  **win_service**:  
    - name: "{{ service_name }}"  
    - state: started
- name: Ensure IIS is installed and started
  hosts: web
  become: yes
  vars:
    service_name: IIS Admin Service

  tasks:
  - name: Ensure IIS Server is present
    win_feature:
      name: Web-Server
      state: present

  - name: Ensure latest index.html file is present
    win_copy:
      src: files/index.html
      dest: c:\www\

  - name: Ensure IIS is started
    win_service:
      name: "{{ service_name }}"
      state: started
---

- name: Ensure IIS is installed and started
  hosts: web
  become: yes
  vars:
    service_name: IIS Admin Service

  tasks:
  - name: Ensure IIS Server is present
    win_feature:
      name: Web-Server
      state: present

  - name: Ensure latest index.html file is present
    win_copy:
      src: files/index.html
      dest: c:\www\

  - name: Ensure IIS is started
    win_service:
      name: "{{ service_name }}"
      state: started
Exercise 3 & 4

- Your First Playbook
Advanced playbooks

Topics Covered:

- Templates
- Loops
- Conditionals
- Tags
- Blocks
Doing more with playbooks

Here are some more essential playbook features that you can apply:

- Templates
- Loops
- Conditionals
- Tags
- Blocks
Doing more with playbooks: **Templates**

Ansible embeds the **Jinja2 templating engine** that can be used to dynamically:

- Set and modify play variables
- Conditional logic
- Generate files such as configurations from variables
Loops can do one task on multiple things, such as create a lot of users, install a lot of packages, or repeat a polling step until a certain result is reached.

- name: Ensure IIS Server is present
  win_feature:
    name: "{{ item }}"
    state: present
  loop:
    - Web-Server
    - NET-Framework-Core
Doing more with playbooks: **Conditionals**

Ansible supports the conditional execution of a task based on the run-time evaluation of variable, fact, or previous task result.

- name: Ensure IIS Server is present
  win_feature:
    name: Web-Server
    state: present
  when: ansible_os_family == "Windows"
Doing more with playbooks: **Tags**

Tags are useful to be able to run a subset of a playbook on-demand.

```yaml
- name: Ensure IIS Server is present
  win_feature:
    name: "{{ item }}"
    state: present
  with_items:
    - Web-Server
    - NET-Framework-Core
  tags:
    - packages

- name: Copy web.config template to Server
  win_template:
    src: templates/web.config.j2
    dest: C:\inetpub\wwwroot\web.config
  tags:
    - configuration
```
Doing more with playbooks: Blocks

Blocks cut down on repetitive task directives, allow for logical grouping of tasks and even in play error handling.

- **block:**
  - **name:** Ensure IIS Server is present
    - **win_feature:**
      - **name:** "{{ item }}"
      - **state:** present
    - **with_items:**
      - **Web-Server**

- **name:** Copy web.config template to Server
  - **win_template:**
    - **src:** templates/web.config.j2
    - **dest:** C:\inetpub\wwwroot\web.config

  **when:** ansible_os_family == "Windows"
Exercise 5

- Practical Playbook Development
Sharing automation

Topics Covered:

- Roles
- Galaxy
Roles

Roles are a packages of closely related Ansible content that can be shared more easily than plays alone.

- Improves readability and maintainability of complex plays
- Eases sharing, reuse and standardization of automation processes
- Enables Ansible content to exist independently of playbooks, projects -- even organizations
- Provides functional conveniences such as file path resolution and default values
### Project with Embedded Roles Example

<table>
<thead>
<tr>
<th>site.yml</th>
<th>iis/</th>
</tr>
</thead>
<tbody>
<tr>
<td>roles/</td>
<td>files/</td>
</tr>
<tr>
<td>common/</td>
<td>templates/</td>
</tr>
<tr>
<td>files/</td>
<td>tasks/</td>
</tr>
<tr>
<td>templates/</td>
<td>handlers/</td>
</tr>
<tr>
<td>tasks/</td>
<td>vars/</td>
</tr>
<tr>
<td>handlers/</td>
<td>defaults/</td>
</tr>
<tr>
<td>vars/</td>
<td>meta/</td>
</tr>
</tbody>
</table>
Roles

Project with Embedded Roles Example

```yaml
# site.yml
---
- name: Execute common and iis role
  hosts: web
  roles:
    - common
    - iis
```
Roles

http://galaxy.ansible.com

Ansible Galaxy is a hub for finding, reusing and sharing Ansible content.

Jump-start your automation project with content contributed and reviewed by the Ansible community.
Exercise 6

- A Playbook Using Roles
Next Steps

GET STARTED
ansible.com/get-started
ansible.com/tower-trial

WORKSHOPS & TRAINING
ansible.com/workshops
Red Hat Training

JOIN THE COMMUNITY
ansible.com/community

SHARE YOUR STORY
Follow us @Ansible
Friend us on Facebook
Thank you