

Ansible Automation





WORKFORCE DEVELOPMENT



DEFINING VARIABLES - PLAY



You can define different variables for each individual host, or set shared variables for a group of hosts in your inventory. You can also define variables in a play.

```
- hosts: webservers
  vars:
    http_port: 80
```

NOTE: When you define variables in a play, they are only visible to tasks executed in that play.

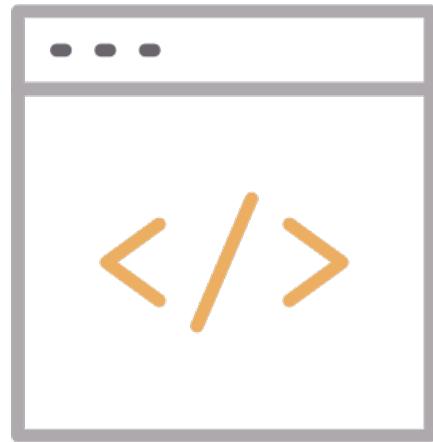
DEFINING VARIABLES – EXTERNAL FILE



You can define variables in reusable variables files and/or in reusable roles. When you define variables in reusable variable files, the sensitive variables are separated from playbooks.

This separation enables storing playbooks in source control and even share them without exposing passwords or other sensitive data.

DEFINING VARIABLES – EXTERNAL FILE



This example shows how you can include variables defined in an external file:

```
- hosts: all
  remote_user: root
  vars:
    favcolor: blue
  vars_files:
    - /var/external_vars.yml
```

DEFINING VARIABLES – EXTERNAL FILE

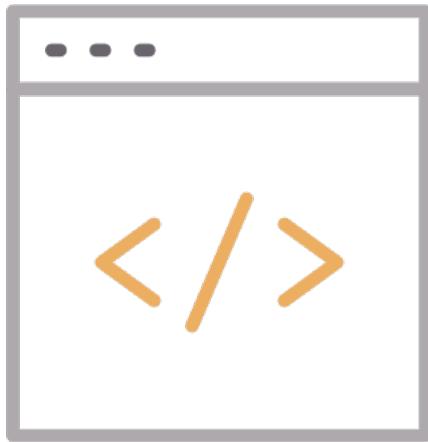


The contents of each variables file is a simple YAML dictionary.

For example:

```
---  
var1: myfavorite  
var2: mysecondfavorite
```

DEFINING VARIABLES – COMMAND LINE

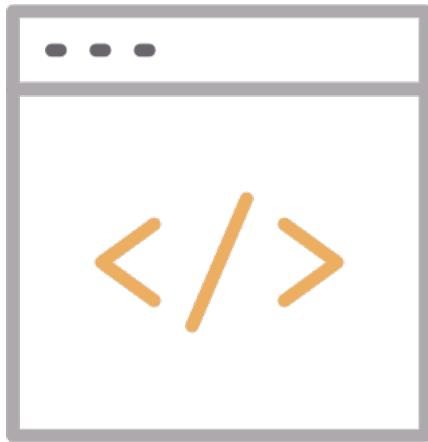


You can define variables when you run your playbook by passing variables at the command line.

--extra-vars (-e)

```
ansible-playbook playbook.yml --extra-vars "red"
```

DEFINING VARIABLES – COMMAND LINE



If you have a lot of special characters, use a JSON or YAML file containing the variable definitions.

```
ansible-playbook release.yml -extra-vars  
"@some_var_file.json"
```

VARIABLE PRECEDENCE



Ansible does apply variable precedence, and you might have a use for it. Here is the order of precedence from least to greatest (the last listed variables override all other variables):

1. Command line values (for example, -u my_user, these are not variables)
2. role defaults (defined in role/defaults/main.yml)
3. Inventory file or script group vars
4. Inventory group_vars/all
5. Playbook group_vars/all
6. inventory group_vars/*
7. playbook group_vars/*
8. inventory file or script host vars
9. inventory host_vars/*
10. playbook host_vars/*

VARIABLE PRECEDENCE



11. host facts / cached set_facts
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 (for example –e "user=my_user") – always wins precedence.

WORKING WITH VARIABLES - LIST



A list variable combines a variable name with multiple values. The multiple values can be stored as an itemized list or in square brackets [], separated with commas.

Defining variables as lists:

You can define variables with multiple values using YAML lists. For example:

Region:

- northeast
- southeast
- midwest

WORKING WITH VARIABLES - LIST



Referencing list variables:

When you use variables defined as a list (also called an array), you can use individual, specific fields from that list. The first item in a list is item 0, the second item is item 1.

For example, to select the first element in the array:

```
region: "{{ region[0] }}"
```

POP QUIZ: DISCUSSION

What does this expression return?

```
region: "{{ region[0] }}"
```



POP QUIZ: DISCUSSION

What does this expression return?

```
region: "{{ region[0] }}"
```

- A: northeast
- B: southeast
- C: midwest



POP QUIZ: DISCUSSION

What does this expression return?

```
region: "{{ region[0] }}"
```

- A: northeast
- B: southeast
- C: midwest



WORKING WITH VARIABLES - DICTIONARY



A dictionary stores the data in key-value pairs. Usually, dictionaries are used to store related data, such as the information contained in an ID or a user profile.

You can define more complex variables using YAML dictionaries. A YAML dictionary maps keys to values. For example:

```
region:  
  field1: one  
  field2: two
```

WORKING WITH VARIABLES - DICTIONARY



When you use variables defined as a key:value dictionary (also called a hash), you can use individual, specific fields from that dictionary using either bracket notation or dot notation:

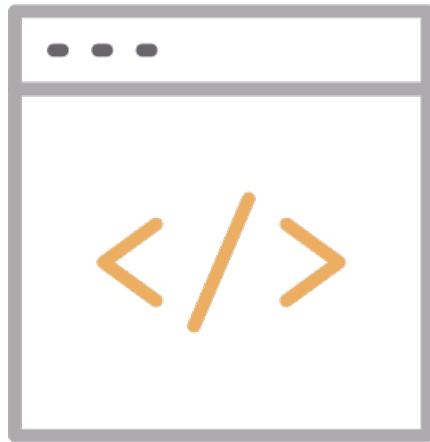
```
foo['field']  
foo.field
```

Dot notation can cause problems because some keys collide with attributes and methods of python dictionaries.

PROTIP: Use bracket notation in most cases.

REGISTERING VARIABLES

You can create variables from the output of an Ansible task with the task keyword `register`. You can use registered variables in any later tasks in your play.



```
- hosts: web_servers
  tasks:
    - name: Register shell command output as variable
      shell: /usr/bin/foo
      register: foo_result
      ignore_errors: true

    - name: Run shell command using output from
      previous task
      shell: /usr/bin/bar
      when: foo_result.rc == 5
```

REGISTERING VARIABLES



Registered variables are stored in memory. You cannot cache registered variables for use in future plays. Registered variables are only valid on the host for the rest of the current playbook run.

If a task fails or is skipped, Ansible still registers a variable with a failure or skipped status, unless the task is skipped based on tags.

QUERYING NESTED DATA



Many registered variables (and facts) are nested YAML or JSON data structures. You cannot access values from these nested data structures with the simple {{ foo }} syntax. You must use either bracket notation or dot notation. For example, to reference an IP address from your facts using the bracket notation:

```
{ { ansible_facts["eth0"]["ipv4"]["address"] } }
```

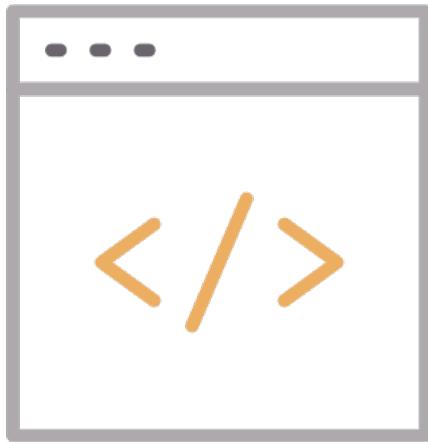
OPTIONAL VARIABLES



By default, Ansible requires values for all variables in a templated expression. However, you can make specific variables optional.

For example, you might want to use a system default for some items and control the value for others. To make a variable optional, set the default value to the special variable `omit`:

OPTIONAL VARIABLES



In this example, the default mode for the files `/tmp/foo` and `/tmp/bar` is determined by the umask of the system.

Ansible does not send a value for `mode`. Only the third file, `/tmp/baz`, receives the `mode=0444` option.

```
- name: Touch files with optional mode
  ansible.builtin.file:
    dest: "{{ item.path }}"
    state: touch
    mode: "{{ item.mode | default('omit') }}"
  loop:
    - path: /tmp/foo
    - path: /tmp/bar
    - path: /tmp/baz
      mode: "0444"
```

HANDLERS



Sometimes you want a task to run only when a change is made on a machine. For example, you may want to restart a service if a task updates the configuration of that service, but not if the configuration is unchanged. Ansible uses handlers to address this use case. Handlers are tasks that only run when notified.

POP QUIZ: DISCUSSION

How have you used handlers?



POP QUIZ: DISCUSSION

When should handlers be used?

- When a configuration file is updated, and the service needs to be restarted.
- When a new release is rolled out

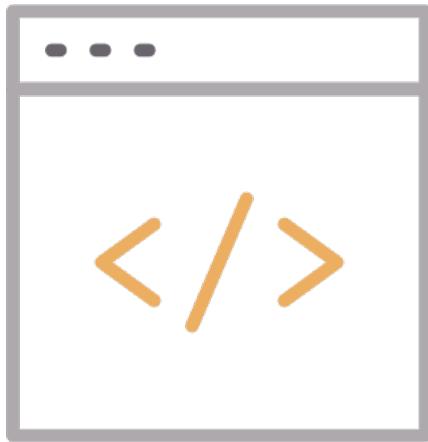


HANDLERS

```
---
```

```
- name: Verify Apache installation
  hosts: web
  vars:
    http_port: 80
    max_clients: 200
  tasks::
    - name: Install Apache
    - yum:
    - name: httpd
...
    - name: Apache config
      template:
        src: httpd.j2
        dest: /etc/httpd.conf
      notify:
        - Restart Apache
  handlers:
    - name: Restart Apache
      service:
        name: httpd
        state: restarted
```

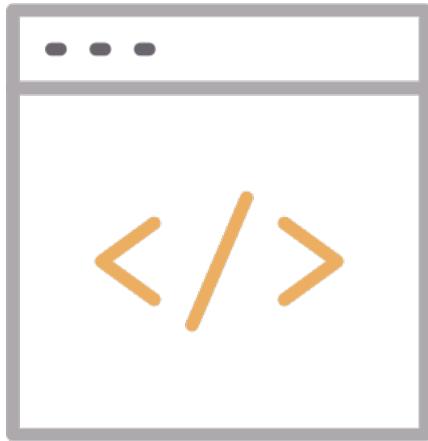
NOTIFY HANDLERS



Tasks can instruct one or more handlers to execute using the `notify` keyword. The `notify` keyword can be applied to a task and accepts a list of handler names that are notified on a task change.

```
tasks:  
  name: Template file  
  template:  
    src: template.j2  
    dest: /etc/foo.conf  
  notify:  
    - Restart apache  
    - Restart memcached
```

NOTIFY HANDLERS



Tasks can instruct one or more handlers to execute using the `notify` keyword. The `notify` keyword can be applied to a task and accepts a list of handler names that are notified on a task change.

```
tasks:  
  notify:  
    - Restart apache  
    - Restart memcached  
handlers:  
  - name: Restart memcached  
    service:  
      name: memcached  
      state: restarted  
  - name: Restart Apache  
    service:  
      name: apache  
      state: restarted
```

SPECIFYING HANDLERS



Handlers must be named in order for tasks to be able to notify them using the `notify` keyword.

Alternately, handlers can utilize the `listen` keyword. Using this handler keyword, handlers can listen on topics that can group multiple handlers as follows:

SPECIFY HANDLERS

```
tasks:::  
  - name: Restart everything  
    command: echo "this task will restart the web services"  
    notify: "restart web services"  
  
handlers:  
  - name: Restart memcached  
    service:  
      name: memcached  
      state: restarted  
      listen: "restart web services"  
  
  - name: Restart Apache  
    service:  
      name: httpd  
      state: restarted  
      listen: "restart web services"
```

Asynchronous Actions & Polling



Ansible runs tasks synchronously, holding the connection to the remote node open until the action is completed. This means within a playbook; each task blocks the next task.

Subsequent tasks will not run until the current task completes.

Challenges:

- Slow
- Long running tasks block all subsequent tasks

Asynchronous Actions & Polling



Playbooks support asynchronous mode and polling, with a simplified syntax.

You can use asynchronous mode in playbooks to avoid connection timeouts or to avoid blocking subsequent tasks. The behavior of asynchronous mode in a playbook depends on the value of poll.

Asynchronous Actions & Polling



For long running tasks, connections to the host can timeout.

If you want to set a longer timeout limit for a certain task in your playbook, use `async` with `poll`.

Ansible will still block the next task in your playbook, waiting until the `async` task either completes, fails or times out. However, the task will only time out if it exceeds the timeout limit you set with the `async` parameter.

Asynchronous Actions & Polling

To avoid timeouts on a task, specify its maximum runtime and how frequently you would like to poll for status:

```
---
```

```
tasks:
  - name: Long running task (15 sec), wait for up to 45 sec, poll every 5 sec
    command: /bin/sleep 15
    async: 45
    poll: 5
```

- DEFAULT_POLL_INTERVAL
The default polling value is 15 seconds.

There is no default for the async time limit. If you omit the `async` keyword the tasks run synchronously.

Default `async` job cache file: `~/.ansible_async`

Asynchronous Actions & Polling



If you want to run multiple tasks in a playbook concurrently, use `async` with `poll` set to 0.

When you set `poll: 0`, Ansible starts the task and immediately moves on to the next task without waiting for a result.

Each `async` task runs until it either completes, fails or times out (runs longer than its `async` value). The playbook run ends **without checking back on `async` tasks**.

Asynchronous Actions & Polling

Playbook with asynchronous task:

```
---
```

```
tasks:
  - name: Long running task, allow for 45 sec, fire and forget
    command: /bin/sleep 15
    async: 45
    poll: 0
```

Be careful! Operations that require a lock (yum, apt, etc.) should not be run using `async` if you intend to run other commands later in the playbook on them.

When running with `poll: 0`, Ansible will not automatically cleanup the `async` job cache file. It will need to be cleaned up manually using the `async_status` module with `mode: cleanup`.

Asynchronous Actions & Polling

Check the status of an `async` task

```
- name: async task
  yum:
    name: docker-io
    state: present
  async: 1000
  poll: 0
  register: yum_sleeper

- name: Check status of async task
  async_status:
    jid: "{{ yum_sleeper.ansible_job_id }}"
  register: job_result
  until: job_result.finished
  retries: 100
  delay: 10
```

Check And Diff Mode



Ansible provides two modes of execution that validate tasks:

- Check mode
 - Ansible runs without making any changes on remote systems.
- Diff mode
 - Ansible provides before-and-after comparisons.

Check And Diff Mode

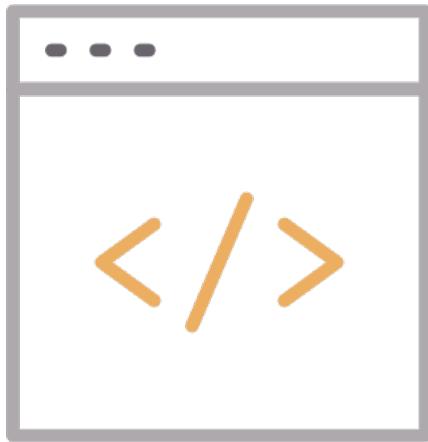


Check mode is just a simulation. It will not generate output for tasks that use conditionals based on registered variables (results of prior tasks). However, it is great for validating configuration management playbooks that run on one node at a time.

To run an entire playbook in check mode:

```
ansible-playbook foo.yml --check
```

Check And Diff Mode



It is also possible to specify that a task always or never runs in check mode regardless of command-line argument.

```
tasks:  
  - name: Always change system  
    command: /bin/change_stuff --even-in-check-mode  
    check_mode: no  
  
  - name: Never change system  
    lineinfile:  
      line: "important config"  
      dest: /path/to/config.conf  
      state : present  
    check_mode: yes  
    register: changes_to_important_config
```

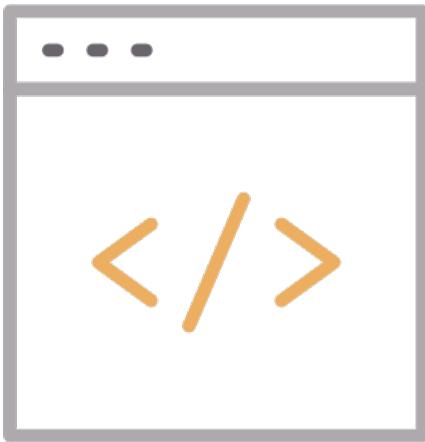
Check And Diff Mode



Running single tasks with `check_mode` can be useful for testing Ansible modules, either to test the module itself or to test the conditions under which it would make changes.

Combining `check_mode` and `register` provides even more detail on potential changes.

Check And Diff Mode



It is possible to skip a task or ignore errors when using `check_mode` by specifying `ansible_check_mode` boolean

```
tasks:  
  - name: Skip in check mode  
    git:  
      repo: ssh://git@github.com/me/hello-world.git  
      dest: /home/me/hello-world  
      when: not ansible_check_mode  
  
  - name: Ignore errors in check mode  
    git:  
      repo: ssh://git@github.com/me/hello-world.git  
      dest: /home/me/hello-world  
      ignore_errors: "{{ ansible_check_mode }}"
```

Check And Diff Mode



The `--diff` option for `ansible-playbook` can be used with `--check` or alone.

When you run in diff mode, any module that supports diff mode reports the changes made or, if used with `--check`, the changes that would have been made.

Diff mode is most common in modules that manipulate files (for example, the `template` module) but other modules might also show 'before and after' information (for example, the `user` module).

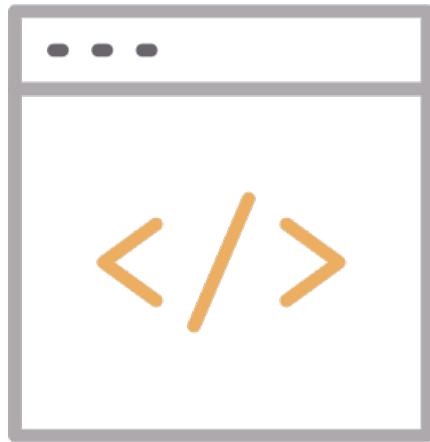
Rolling Updates



Batch size:

By default, Ansible will try to manage all the machines referenced in a play in parallel. For a rolling update use case, you can define how many hosts Ansible should manage at a single time by using the `serial` keyword:

Rolling Updates



Example playbook utilizing serial keyword.

```
- name: test play
  hosts: webservers
  serial: 2
  gather_facts: False

  tasks:
    - name: task one
      command: hostname
    - name: task two
      command: hostname
```

With 4 hosts in the group 'webservers', 2 would complete the play before moving onto the next 2 hosts.

Rolling Updates

In the previous example, if we had 4 hosts in the group 'webservers', 2 would complete the play before moving on to the next 2 hosts:

```
PLAY [webservers] ****
```

```
TASK [task one] ****
```

```
changed: [web2]
```

```
changed: [web1]
```

```
TASK [task two] ****
```

```
changed: [web1]
```

```
changed: [web2]
```

```
PLAY [webservers] ****
```

Rolling Updates

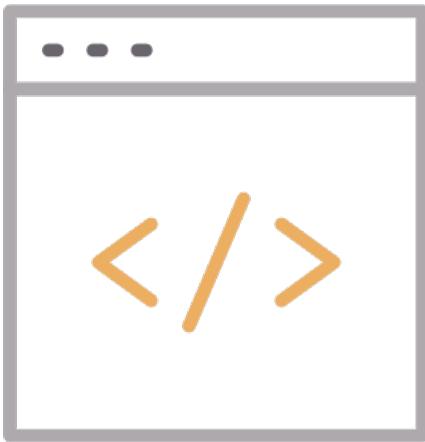
Now that web1 & web2 are complete Ansible continues with web3 & web4

```
PLAY [webservers] *****
TASK [task one] *****
changed: [web3]
changed: [web4]

TASK [task two] *****
changed: [web3]
changed: [web4]

PLAY RECAP *****
web1 : ok=2 changed=2 unreachable=0 failed=0
web2 : ok=2 changed=2 unreachable=0 failed=0
web3 : ok=2 changed=2 unreachable=0 failed=0
web4 : ok=2 changed=2 unreachable=0 failed=0
```

Rolling Updates

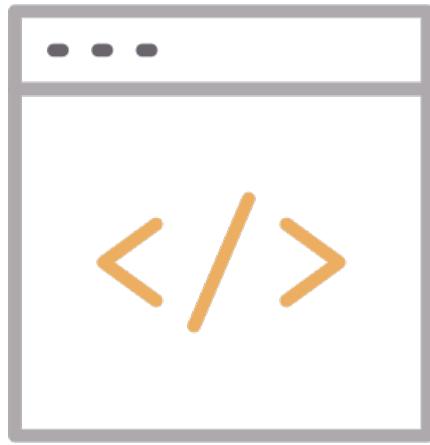


The `serial` keyword can also be specified as a percentage, which will be applied to the total number of hosts in a play, in order to determine the number of hosts per pass:

```
- name: test play
  hosts: webservers
  serial: 30%
```

If the number of hosts does not divide equally into the number of passes, the final pass will contain the remainder.

Rolling Updates

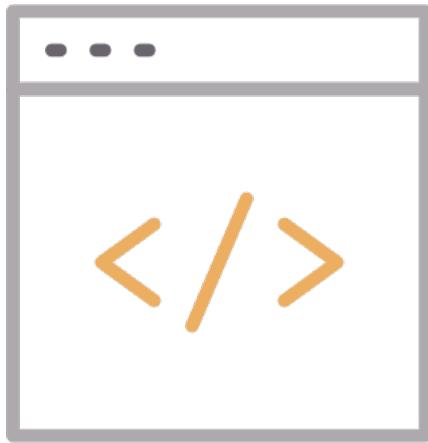


The batch size can be specified as a list with integer, or percent:

```
- name: test play
  hosts: webservers
  serial:
    - 1
    - 5
    - 10
```

Above the first batch would contain a single host, next 5, and if any left, each would have 10 until complete.

Rolling Updates



The batch size can be specified as a list with integer, or percent:

```
- name: test play
  hosts: webservers
  serial:
    - "10%"
    - "20%"
    - "100%"
```

Rolling Updates



Maximum Threshold Percentage:
Ansible executes tasks on all hosts in the defined group unless `serial` is defined.

In some situations, such as with the rolling updates, it may be desirable to abort the play when a certain threshold of failures have been reached. To achieve this, you can set a maximum failure percentage on a play as follows:

```
- hosts: webservers
  max_fail_percentage: 30
  serial: 10
```

Ansible Vault



Ansible Vault



Ansible Vault encrypts variables and files so you can protect sensitive content such as passwords or keys rather than leaving it visible as plaintext in playbooks or roles.

To use Ansible Vault you need one or more passwords to encrypt and decrypt content.

Use the passwords with the `ansible-vault` command-line tool to create and view encrypted variables, create encrypted files, encrypt existing files, or edit, re-key, or decrypt files. You can then place encrypted content under source control and share it more safely.

Ansible Vault



Ansible Vault can prompt for a password every time, or you can configure it to use a password file.

```
#ansible.cfg  
[defaults]  
vault_password_file = ~/.vault_pass
```

Ansible Vault



Each time you encrypt a variable or file with Ansible Vault, you must provide a password. When you use an encrypted variable or file in a command or playbook, you must provide the same password that was used to encrypt it.

POP QUIZ: DISCUSSION

Things to consider:

- Do you want to encrypt all your content with the same password, or use different passwords for different needs?
- Where do you want to store your password(s)?



Ansible Vault



Small teams can use a single password for everything encrypted. Store the vault password securely in a file or secret manager.

If you have a large team or many sensitive values to manage it is recommended to use multiple passwords.

You can use different passwords for different users or different levels of access. Depending on your needs, you might want a different password for each encrypted file, for each directory, or for each environment.

Ansible Vault



You might have a playbook that includes two vars files, one for the dev environment and one for the production environment, encrypted with two different passwords.

When you run the playbook, select the correct vault password for the environment you are targeting, using a vault ID.

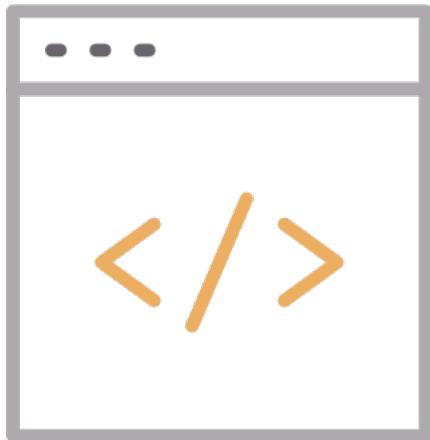
Vault Id

A vault ID is an identifier for one or more vault secrets.

Vault IDs provide labels to distinguish between individual vault passwords.

To use vault IDs, you must provide an ID label of your choosing and a source to obtain its password (either prompt or a file path):

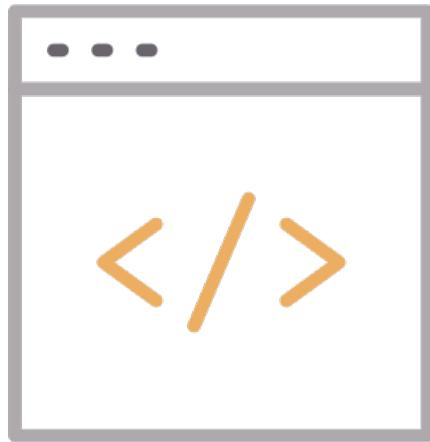
```
--vault-id label@source
```



This switch is available for all commands that interact with vaults:

- ansible-vault
- ansible-playbook
- etc.

Ansible Vault



Create a new encrypted data file

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

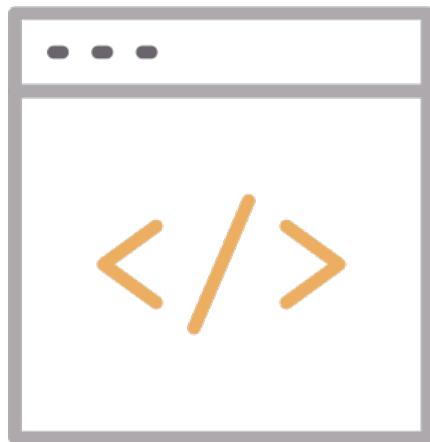
Prompt for vault password

```
ansible-playbook --ask-vault-pass myplay.yml
```

Use password file

```
ansible-playbook --vault-password-file pass myfile.yml
```

Ansible Vault



Common vault commands

```
# Edit encrypted files
ansible-vault edit playbook.yml

# Rekeying
ansible-vault rekey play.yml task.yml report.yml

# Encrypt existing files
ansible-vault encrypt foo.yml bar.yml baz.yml

# Decrypting files
ansible-vault decrypt task.yml run.yml play.yml

# View encrypted files
ansible-vault view break.yml fix.yml fun.yml
```

Ansible Tags



Tags



If you have a large playbook, it may become useful to be able to run only a specific part of it rather than running everything in the playbook. Ansible supports a `tags` attribute for this reason.

Tags can be applied at multiple levels including:

- Tasks
- Roles
- Plays
- Blocks

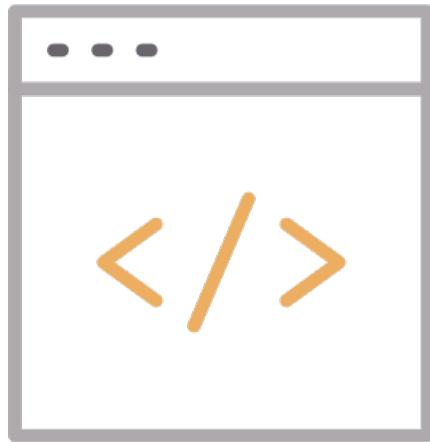
Tags



At the simplest level, you can apply one or more tags to an individual task. You can add tags to tasks in playbooks, in task files, or within a role.

It is also possible to add the same tag to multiple tasks.

Tags



Here's an example showing tasks to install and configure software. Using tags, it is possible to specify which task runs.

```
tasks:  
- name: Install  
  yum:  
    name:  
    - httpd  
    - memcached  
    state: present  
  tags:  
  - packages  
  - webservers  
- name: Configure  
  template:  
    src: templates/src.j2  
    dest: /etc/foo.conf  
  tags:  
  - configuration
```

Tags



This example shows the 'ntp' tag

- ```
- name: Install ntp
 yum:
 name: ntp
 state: present
 tags: ntp

- name: Install nslookup
 yum:
 name: nslookup
 state: present
```

If you ran these tasks in a playbook with `--tags ntp`, Ansible would run the one tagged `ntp` and skip the other.

# Tags



## Inheritance:

No one wants to add the same tag to multiple tasks. To avoid repeating code you can tag the play, block, or role. Ansible applies the tags down the dependency chain to all child tasks.

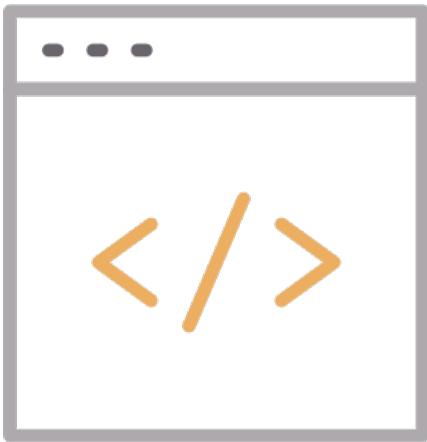
Blocks are useful for applying a tag to many, but not all, of the tasks in your play.

Plays are better suited if every task in the play should have the same tags.

Adding tags to roles allows you to run specific roles.

# Tags

Define tags at the block level



```
- name: ntp tasks
 tags: ntp
 block:
 - name: Install ntp
 yum:
 name: ntp
 state: present

 - name: Enable and run ntp
 service:
 name: ntpd
 state: started
 enabled: yes
 tags: ntp

 - name: Install utils
 yum:
 name: ntf-utils
 state: present
```

# Ansible Galaxy



# Ansible Collections



You can extend Ansible by adding custom modules or plugins. You can create them from scratch or copy existing ones for local use.

A simple way to share plugins and modules with your team or organization is by including them in a collection and publishing the collection on Ansible Galaxy.

# Ansible Collections



## Modules:

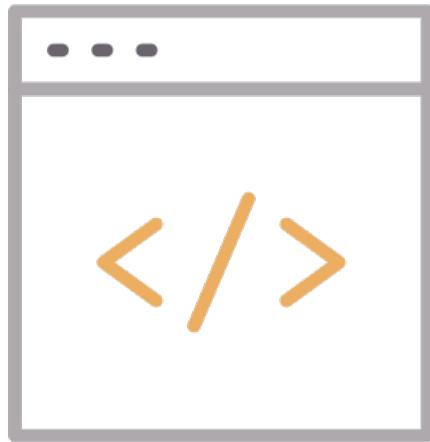
Modules are reusable, standalone scripts that can be used by the Ansible API, the `ansible` command, or the `ansible-playbook` command.

Modules provide a defined interface. Each module accepts arguments and returns information to Ansible by printing a JSON string to stdout before exiting. Modules execute on the target system (usually that means on a remote system) in separate processes.

## Plugins:

Plugins extend Ansible's core functionality and execute on the control node within the `/usr/bin/ansible` process. Plugins offer options and extensions for the core features of Ansible - transforming data, logging output, connecting to inventory, and more

# Ansible Galaxy



Use ansible-galaxy to install collection or role

```
ansible-galaxy (collection|role) install
```

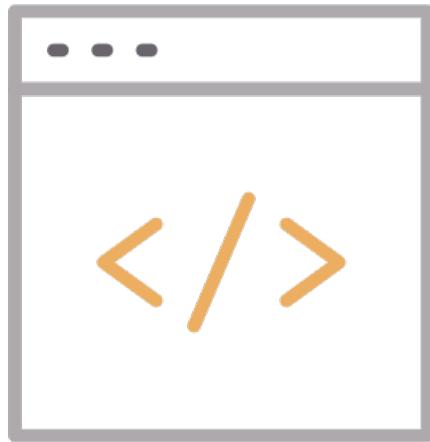
You can install from the community, or any .tar.gz file.

```
ansible-galaxy collection install azure.azcollection
```

Use new collection (full namespace, collection, collections element)

```
- name: Azure collection
 hosts: localhost
 collections:
 - azure.azcollection
 tasks:
 - azure_rm_storageaccount:
 resource_group: myRG
 name: myStorageAccount
 account_type: Standard_LRS
```

# Ansible Galaxy



Use ansible-galaxy to install role

```
ansible-galaxy role install
```

You can install from the community, or any .tar.gz file.

```
ansible-galaxy role install weareinteractive.users
```

Use new role:

```
- name: Create user
 hosts: all
 roles:
 - weareinteractive.users
 vars:
 users:
 - username: newuser
 append: yes
 password: 6paD7LIRYpWiv7
```

# Ansible Roles

## Lab five



# WHY THE ANSIBLE AUTOMATION PLATFORM?

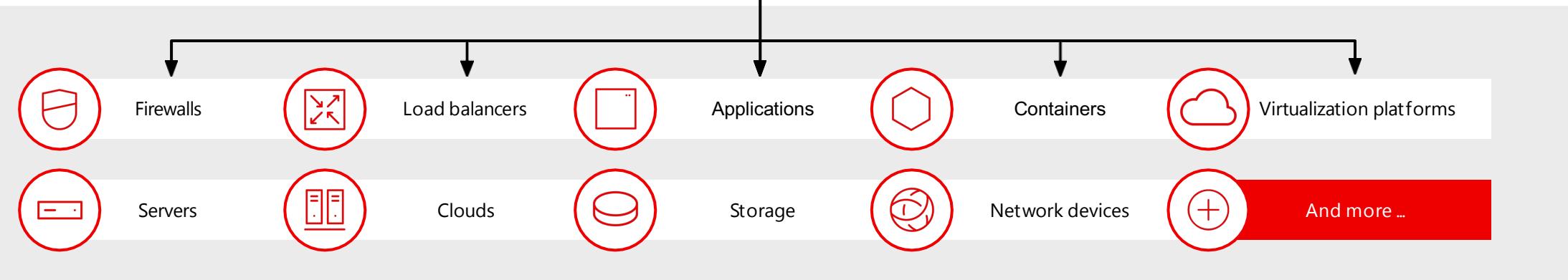
## Automate the deployment and management of automation

Your entire IT footprint

Do this...

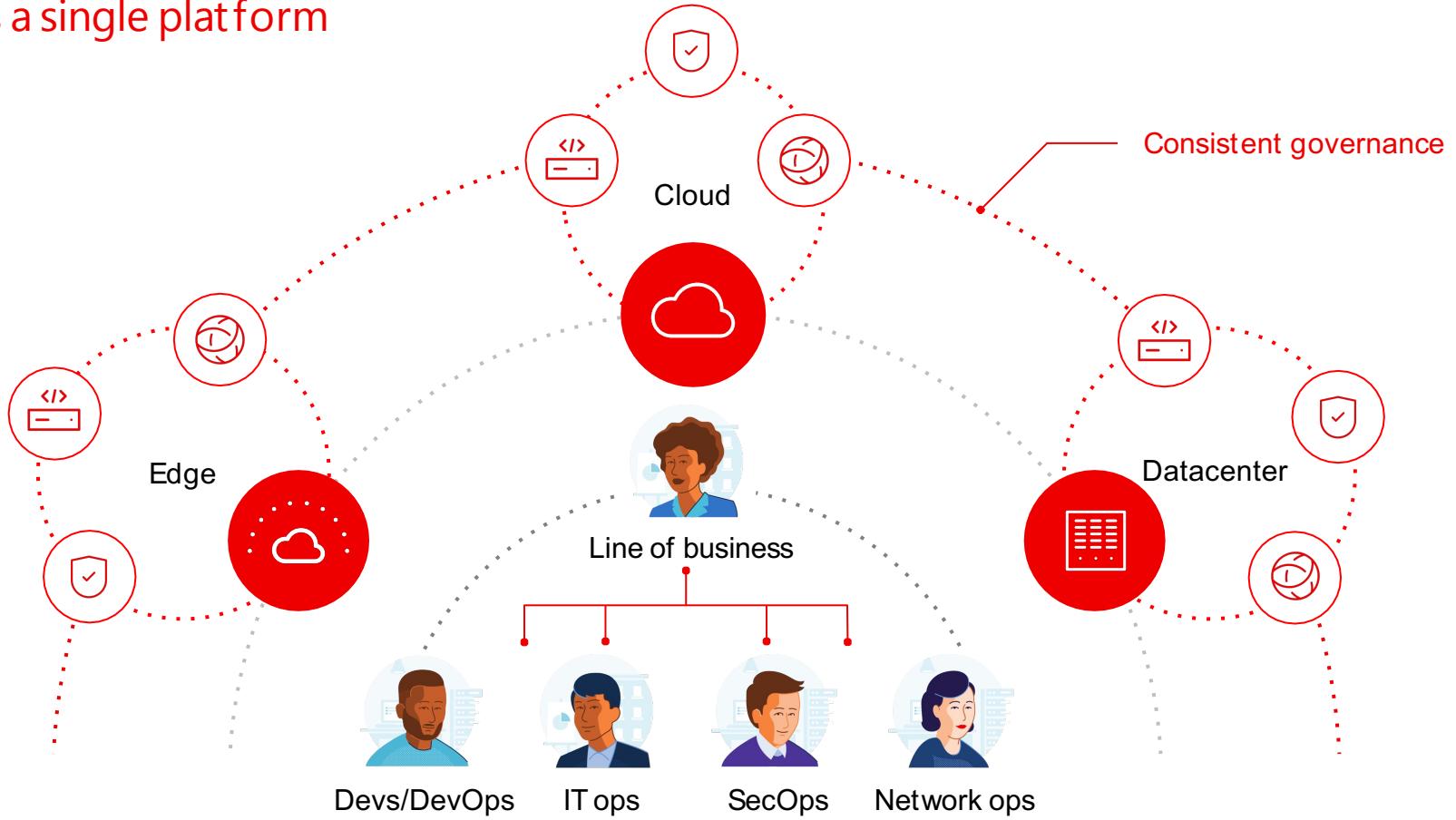
Orchestrate      Manage configurations      Deploy applications      Provision / deprovision      Deliver continuously      Secure and comply

On these...



# BREAK DOWN SILOS

Different teams a single platform





## Red Hat Ansible Automation Platform



Content creators



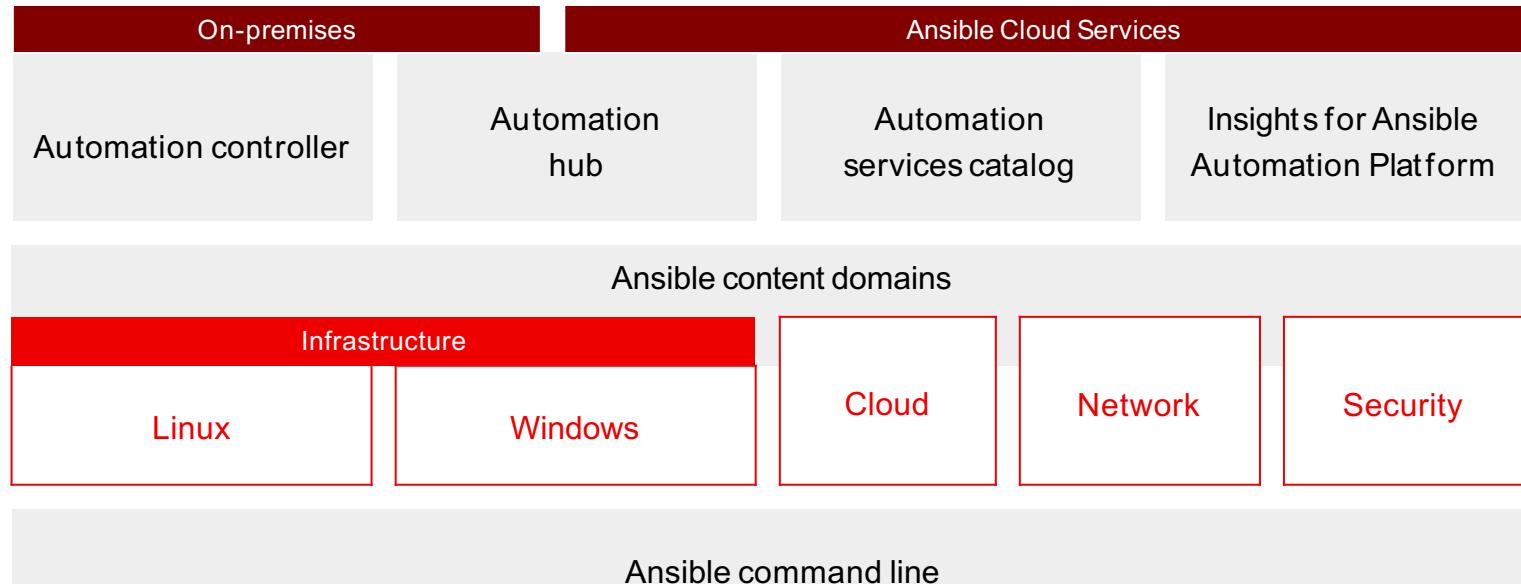
Operators



Domain experts



Users



Fueled by an  
open source community

# AUTOMATION PLATFORM CONCEPTS



A control node is installed with Ansible and is used to run playbooks.

- Contains the Ansible Engine software, the playbook, and its supporting files.
- Red Hat Automation Platform is a control node that also provides a central web interface, authentication, and API for Ansible.

A managed host is a machine that is managed by Ansible automation.

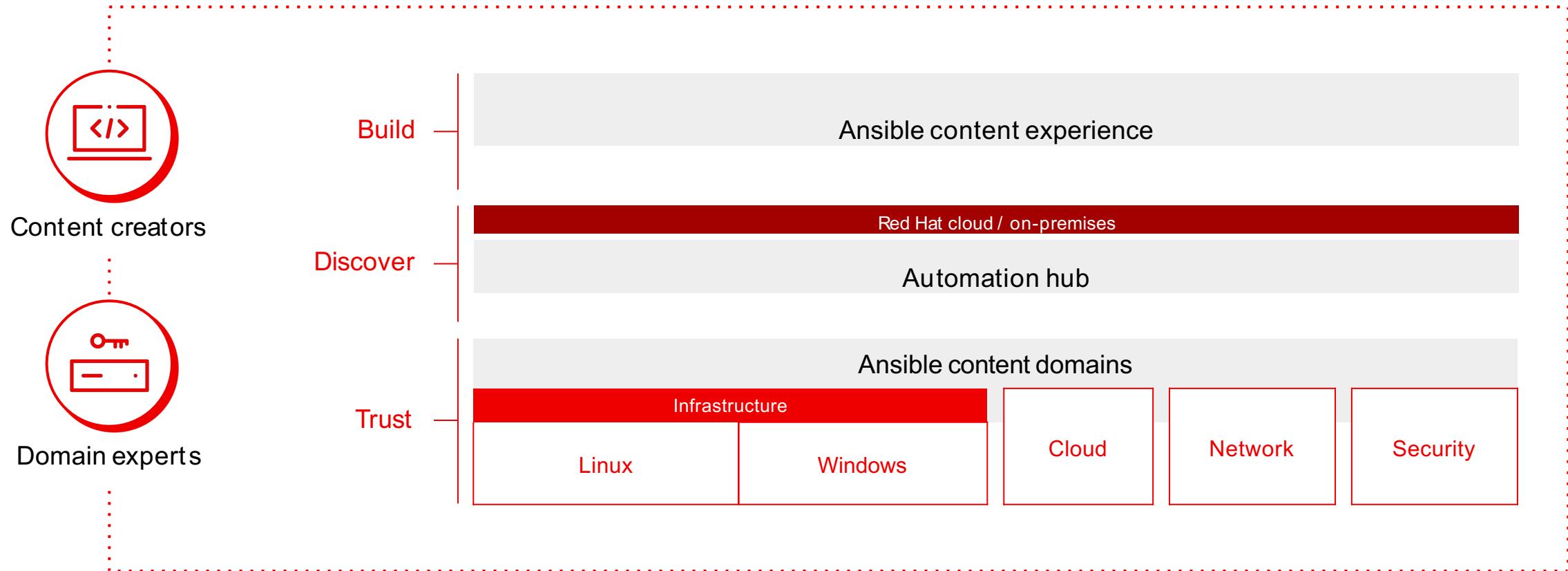
- Does not have Ansible installed
- Does need to be configured to allow Ansible to connect to the host
- Must be listed in the inventory (or generated by a dynamic inventory script or plugin)

# ANSIBLE AUTOMATION PLATFORM INFRASTRUCTURE



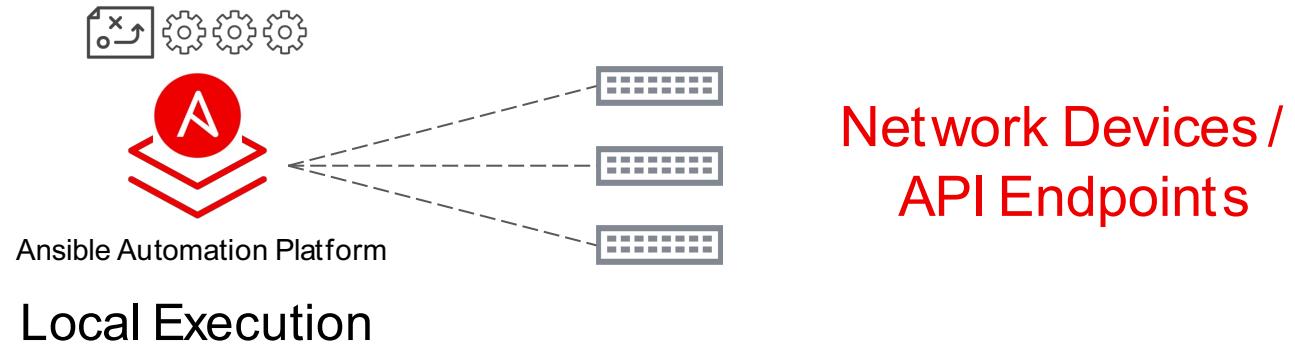
# Create

## The automation lifecycle

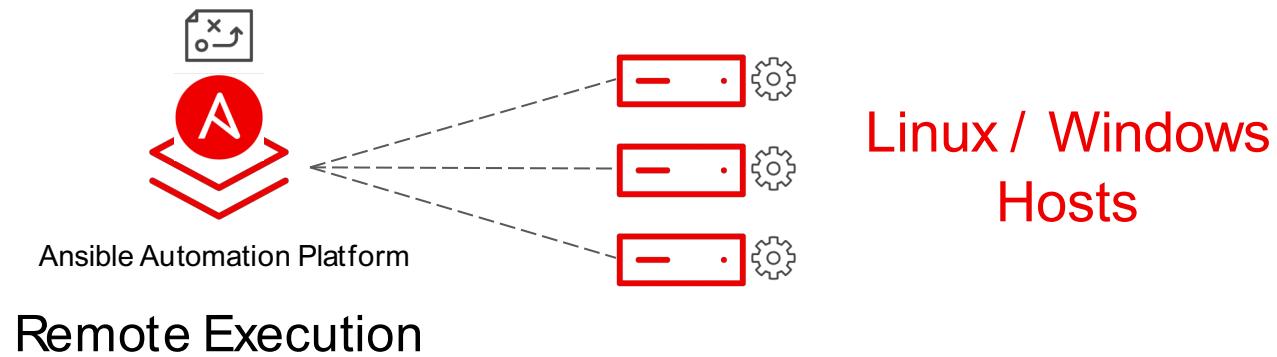


# HOW ANSIBLE AUTOMATION WORKS

Module code is executed locally on the control node



Module code is copied to the managed node, executed, then removed



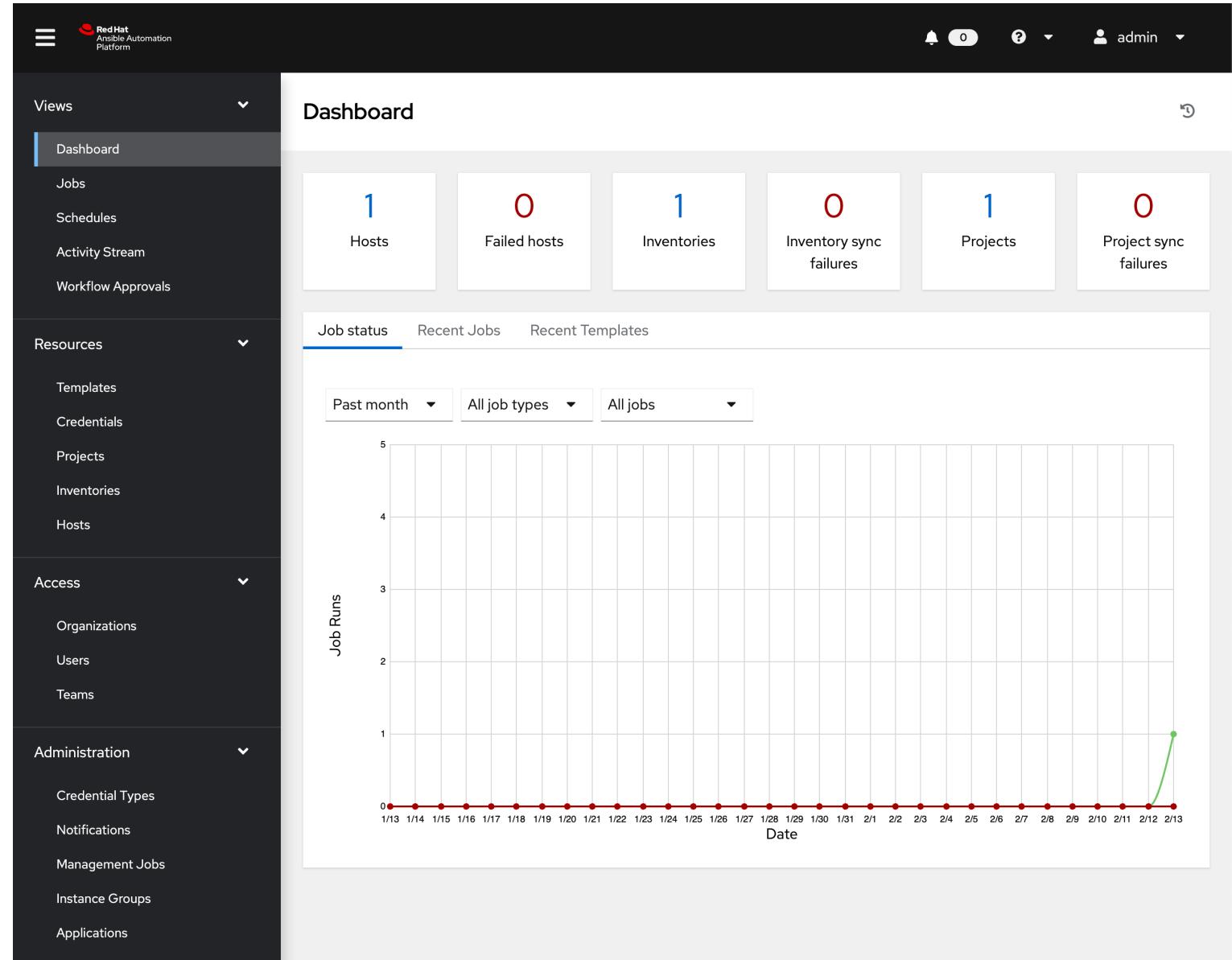
# Installation Architecture Options

Red Hat Ansible Automation Platform can be implemented using one of the following architectures:

- Single Machine with Integrated Database
- Single Machine with Remote Database
- Multi Machine Cluster with Remote Database
- OpenShift Pod with Remote Database

# AUTOMATION PLATFORM DASHBOARD

- The main control center for Red Hat Ansible Tower.
- Displayed when you log in.
- Composed of four reporting sections:
  - Summary
  - Job Status
  - Recently Used
  - TemplatesRecent Job Runs



# Dashboard Navigation

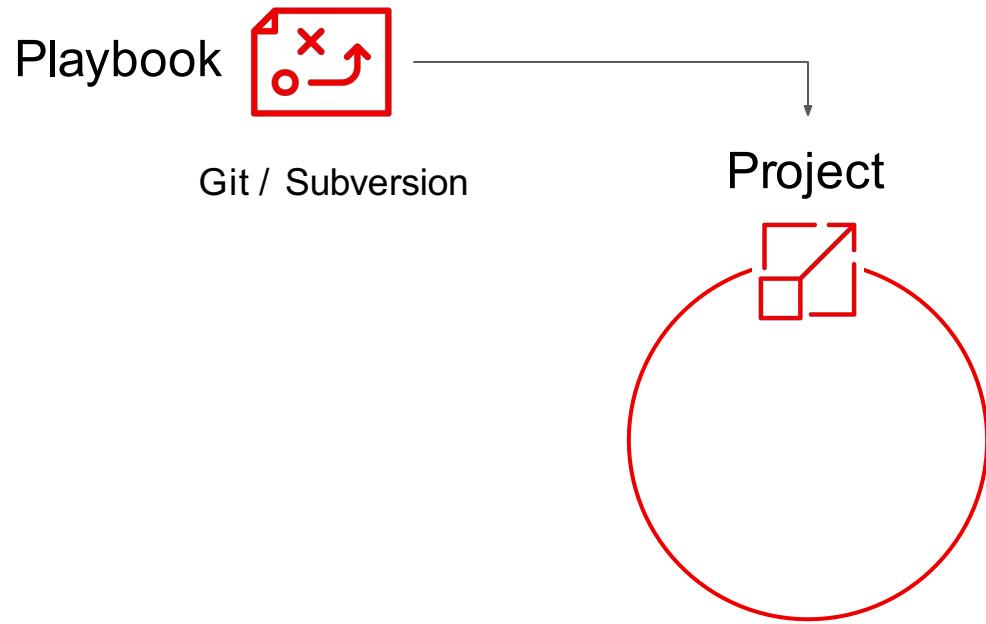
The dashboard contains links to common resources.

|                   |                                                                  |
|-------------------|------------------------------------------------------------------|
| Organizations     | Control what resources are visible to which users.               |
| Teams             | Groups of users that need to access the same resources           |
| Users             | User accounts                                                    |
| Jobs              | A history of previous Ansible runs.                              |
| Templates         | Prepared playbooks settings that can be "launched" to run a job. |
| Credentials       | Authentication secrets for managed hosts and Git repositories    |
| Projects          | Sources of Ansible Playbooks (usually Git repo)                  |
| Inventories       | Inventories of managed hosts                                     |
| Inventory Scripts | Dynamic inventory                                                |
| Notifications     | Configurable for job completion or failure                       |
| Management Jobs   | Special jobs used to maintain Ansible Platform.                  |

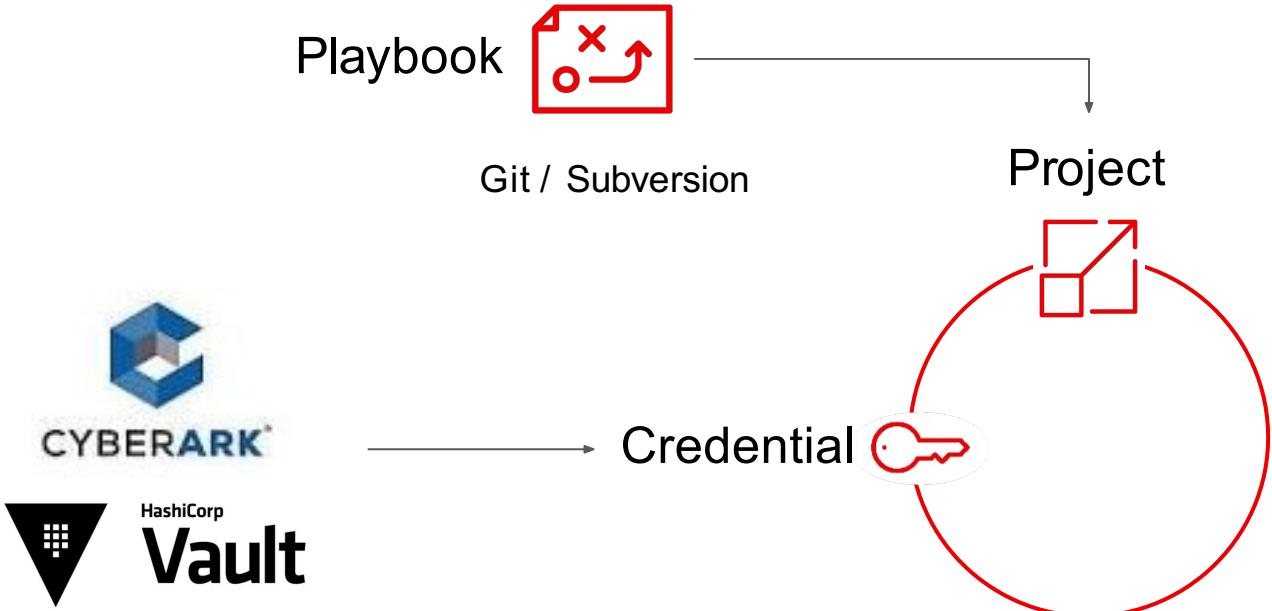
# Automation Controller



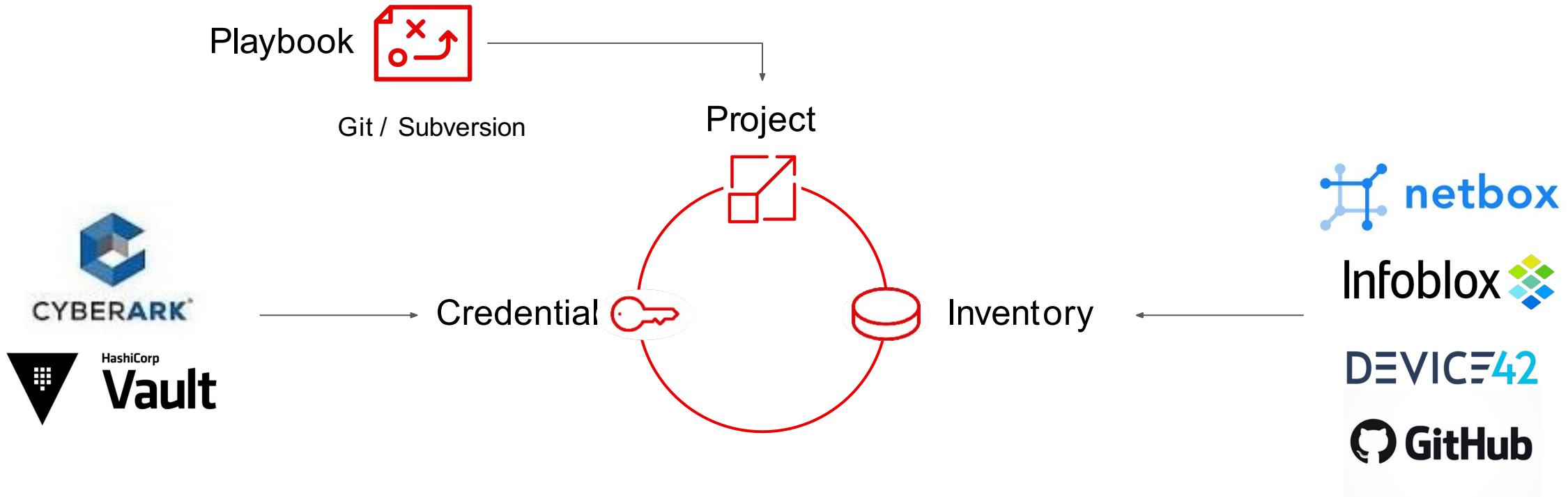
# Anatomy of An Automation Job



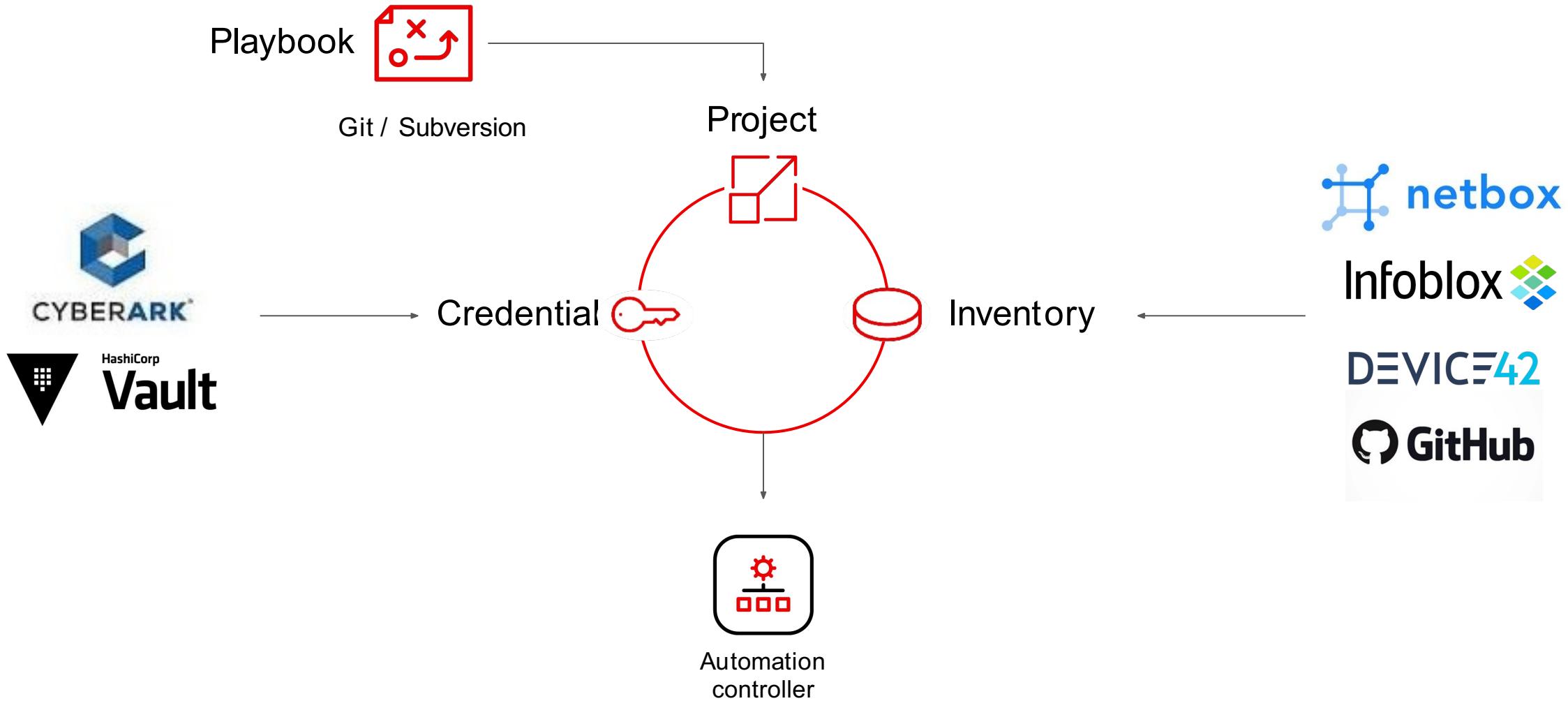
# Anatomy of An Automation Job



# Anatomy of An Automation Job



# Anatomy of An Automation Job



# Organization

- Newly created users inherit specific roles from their organization based on their user type.
- Assign additional roles to a user after creation to grant permissions to view, use, or change other Ansible Platform objects.

Organizations

The screenshot shows a list of organizations in the Ansible Platform. The interface includes a search bar, an 'Add' button, and a delete button. The table has columns for Name, Members, Teams, and Actions. Two entries are listed: 'Default' and 'NewOrg'. Both entries have 0 members and 0 teams. Each entry has an edit icon (pencil) in the Actions column. Navigation buttons at the bottom indicate 1-2 of 2 items and 1 of 1 page.

| <input type="checkbox"/> Name    | Members | Teams | Actions |
|----------------------------------|---------|-------|---------|
| <input type="checkbox"/> Default | 0       | 0     |         |
| <input type="checkbox"/> NewOrg  | 0       | 0     |         |

# Team

- You can apply permissions at the team level.

Teams

Create New Team

Save Cancel

Name \*

Description

Organization \*

# User Roles

- An organization is also one of these objects.
- There are three roles that users can be assigned:
- Admin
- Auditor
- User

Users

Create New User

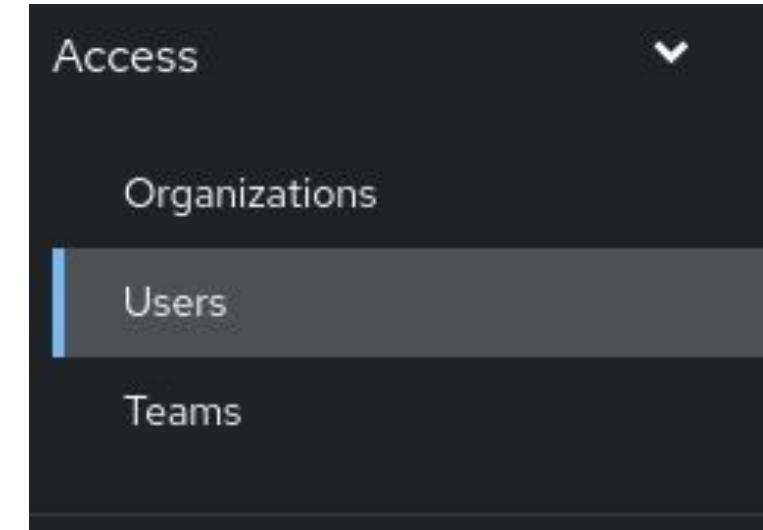
The screenshot shows a user creation interface. At the top, there are fields for First Name (User), Last Name (One), and Email (user1@domain.com). Below these are fields for Username (user1), Password, and Confirm Password, each with a visibility icon. A dropdown menu for User Type is open, showing options: ✓ Normal User (selected), System Auditor, and System Administrator. At the bottom are Save and Cancel buttons.

|                                                         |                |                    |
|---------------------------------------------------------|----------------|--------------------|
| First Name                                              | Last Name      | Email              |
| User                                                    | One            | user1@domain.com   |
| Username *                                              | Password *     | Confirm Password * |
| user1                                                   | .....          | .....              |
| User Type *                                             | Organization * |                    |
| ✓ Normal User<br>System Auditor<br>System Administrator | NewOrg         |                    |

Save Cancel

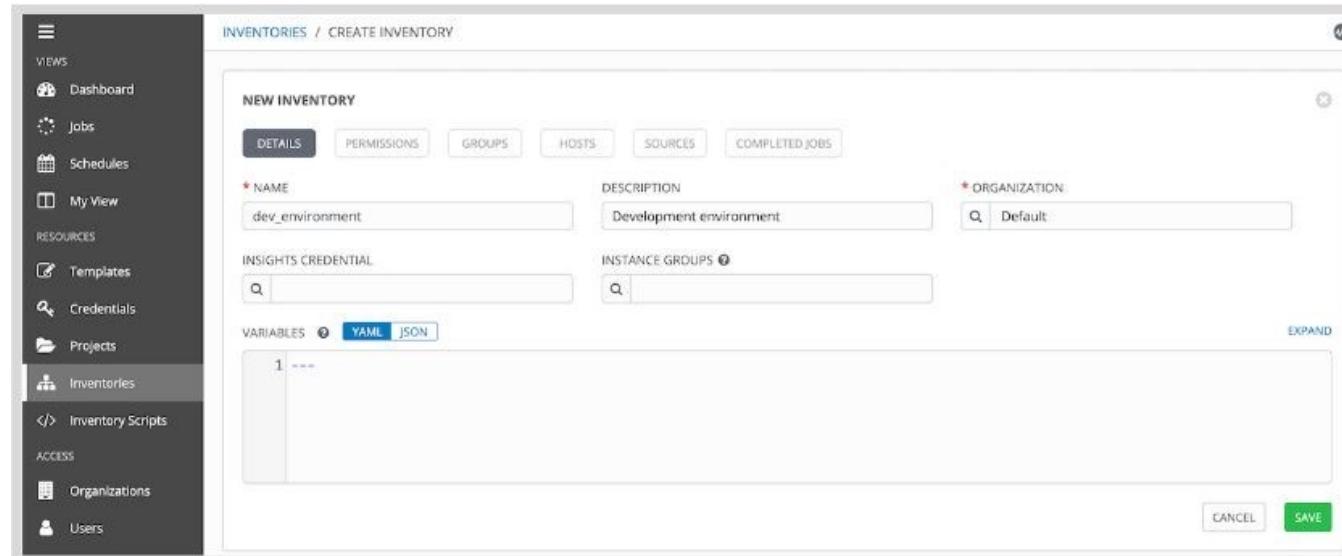
# User Management

- An **organization** is a logical collection of users, teams, projects, inventories and more. All entities belong to an organization.
- A **user** is an account to access Ansible Automation Controller and its services given the permissions granted to it.
- **Teams** provide a means to implement role-based access control schemes and delegate responsibilities across organizations.



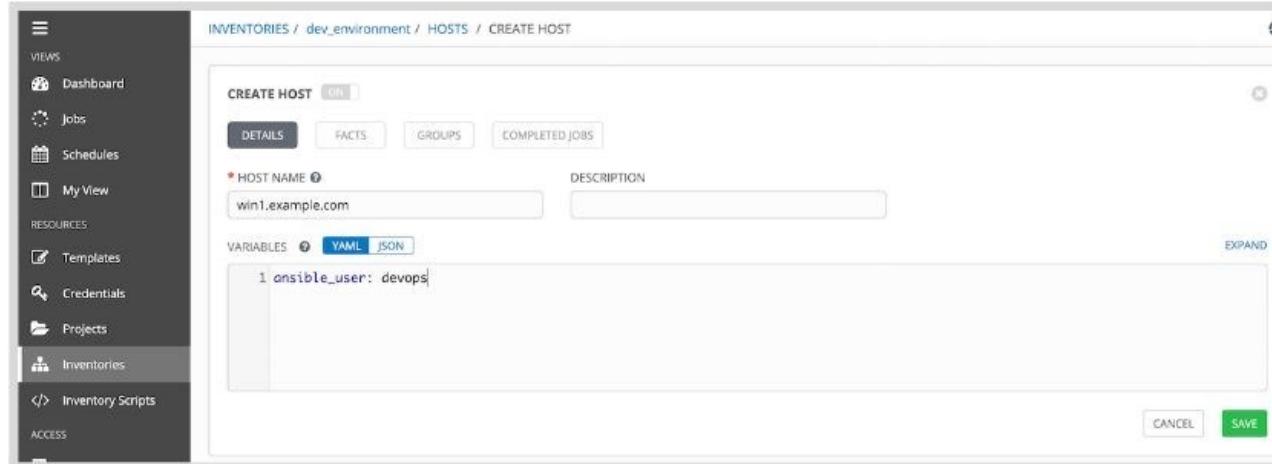
# Inventory

- Log into Ansible Platform (the admin user will work for this example).
- Click on **Inventories**.
- In the INVENTORIES window, click the + button.
- Enter a NAME for the inventory and its ORGANIZATION (often “Default”)



# Inventory

- In the Ansible Platform GUI, click the **Inventories** menu, then click on the name of the inventory.
  - Click the **HOSTS** button, then click on **+**. This displays the “Create a new host” tooltip.
  - In the **HOST NAME** field enter the hostname or IP address of the managed host.
  - In the **VARIABLES** text box, you can set values for variables that apply only to this host.
  - Click **SAVE**.

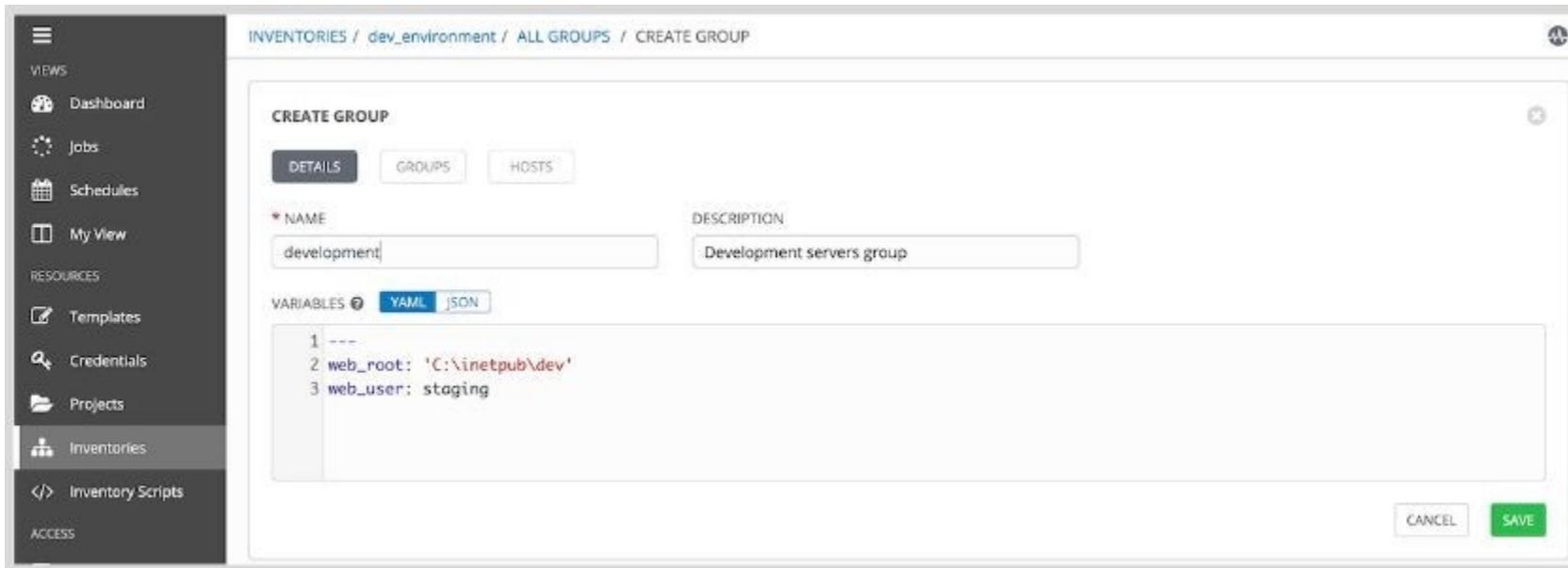


# Inventory Groups

- Groups allow you to organize hosts into a set that can be managed together
- Hosts may be in multiple groups at the same time
  - All hosts that are in a particular data center
  - All hosts that have a particular purpose
  - Dev / Test / Prod hosts can be grouped
- Groups can be nested
  - The *europe* group might include a *paris\_dc* group and a *london\_dc* group
- This allows you to run playbooks on particular groups
- This allows you to set a variable to a specific value for all hosts in a group

# Inventory Groups

- In the Ansible Platform GUI, click the **Inventories** menu, and click on the inventory to edit.
- Click the **GROUPS** button, then click on **+**. This will open the “Create a new group” tooltip.
- In the NAME field, enter the name of the group.
- Define any values for variables
- Click **SAVE**.

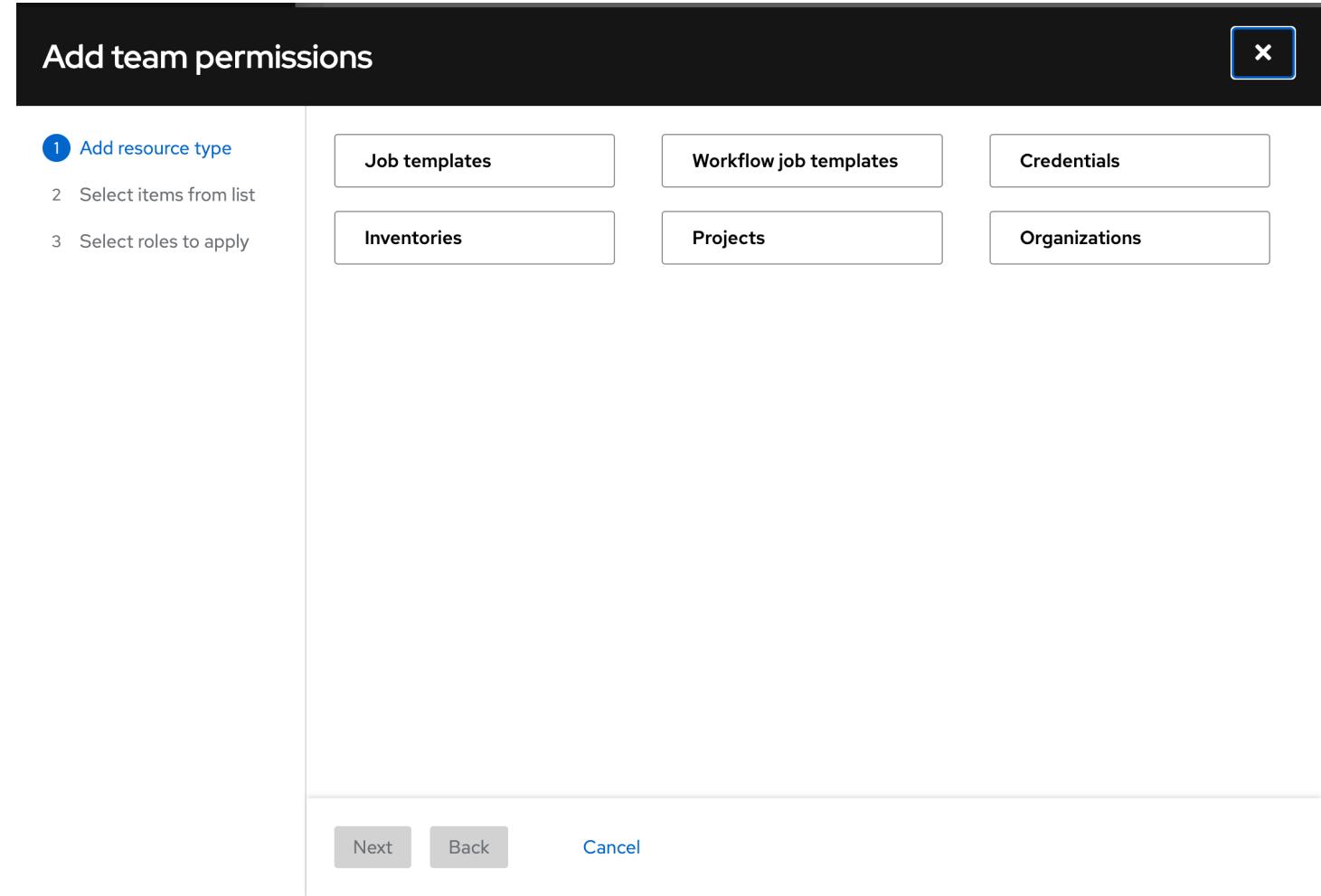


# Inventory Groups

- In the Ansible Platform GUI, click the **Inventories** menu, and click on the inventory to edit.
- Click the **GROUPS** button, then click on the group to edit.
- Click the **HOSTS** button, then click on **+**. This will open the “Add a host” tooltip. Select “New Host”.
- In the HOST NAME field, enter the hostname or IP address of the managed host to add.
- Define any values for variables that affect only that host (overriding any group variables).
- Click **SAVE**.

# Roles

- Create custom roles with specific permissions.



# Inventory Roles

| Role   | Description                                                                                                                                                                                                                                                                                |
|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Admin  | The inventory <b>Admin</b> role grants users full permissions over an inventory. These permissions include deletion and modification of the inventory. In addition, this role also grants permissions associated with the inventory roles <b>Use</b> , <b>Ad Hoc</b> , and <b>Update</b> . |
| Use    | The inventory <b>Use</b> role grants users the ability to use an inventory in a job template resource. This controls which inventory is used to launch jobs using the job template's playbook.                                                                                             |
| Ad Hoc | The inventory <b>Ad Hoc</b> role grants users the ability to use the inventory to execute ad hoc commands.                                                                                                                                                                                 |
| Update | The inventory <b>Update</b> role grants users the ability to update a dynamic inventory from its external data source.                                                                                                                                                                     |
| Read   | The inventory <b>Read</b> role grants users the ability to view the contents of an inventory.                                                                                                                                                                                              |

# Inventory Variables

When you manage a static inventory in the Ansible Platform web UI, you may define inventory variables directly in the inventory objects.

- Variables set in the inventory details affect all hosts in the inventory.
- Variables set in a group's details are the equivalent of `group_vars`.
- Variables set in a host's details are the equivalent of `host_vars`.

# Inventory Variables

INVENTORIES / MAIL SERVERS

MAIL SERVERS

DETAILS    PERMISSIONS    GROUPS    HOSTS    SOURCES    COMPLETED JOBS

\* NAME    DESCRIPTION    \* ORGANIZATION

MAIL SERVERS    Default

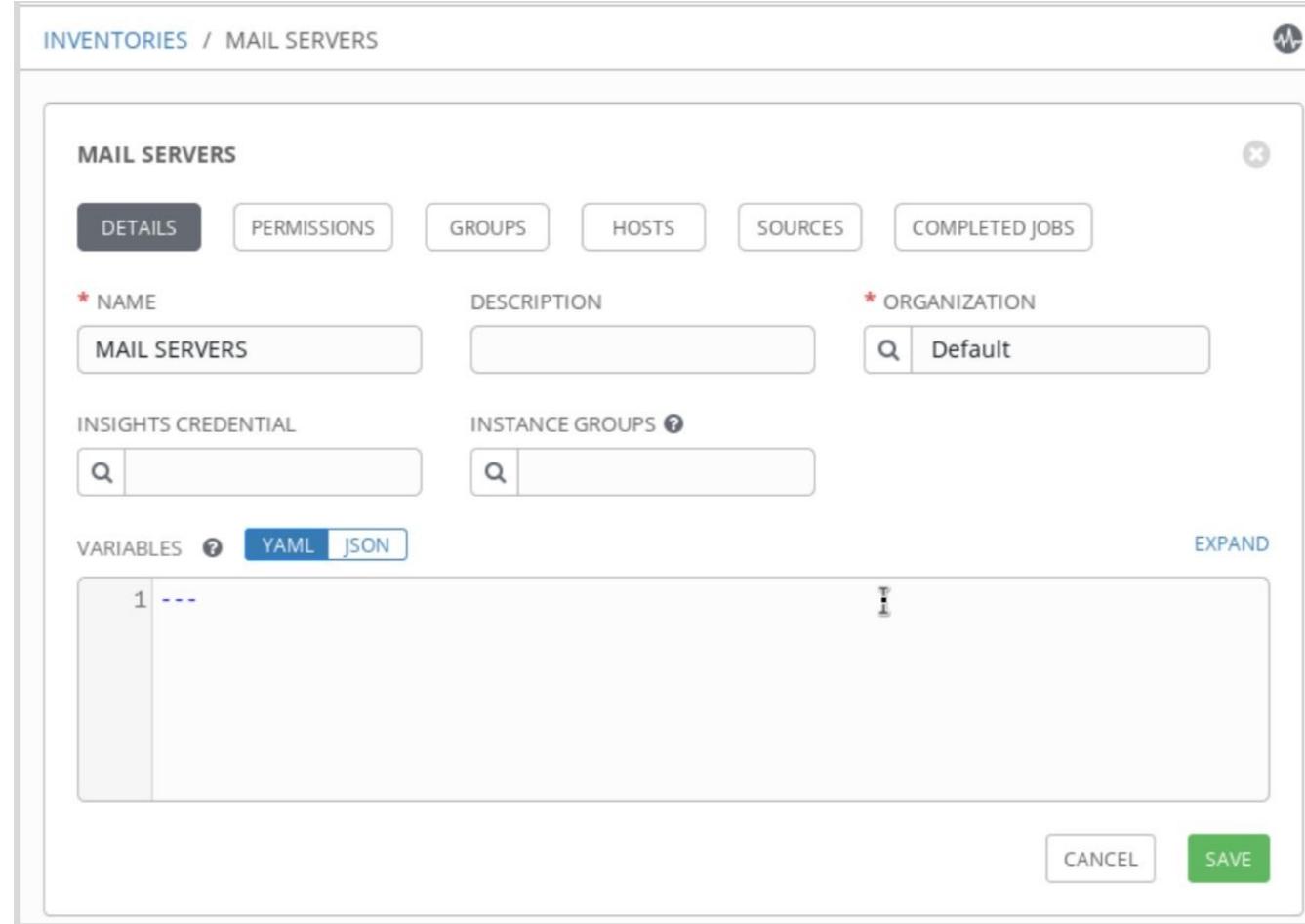
INSIGHTS CREDENTIAL    INSTANCE GROUPS

VARIABLES    EXPAND

YAML    JSON

```
1 ---
```

CANCEL    SAVE



# Inventory Group Variables

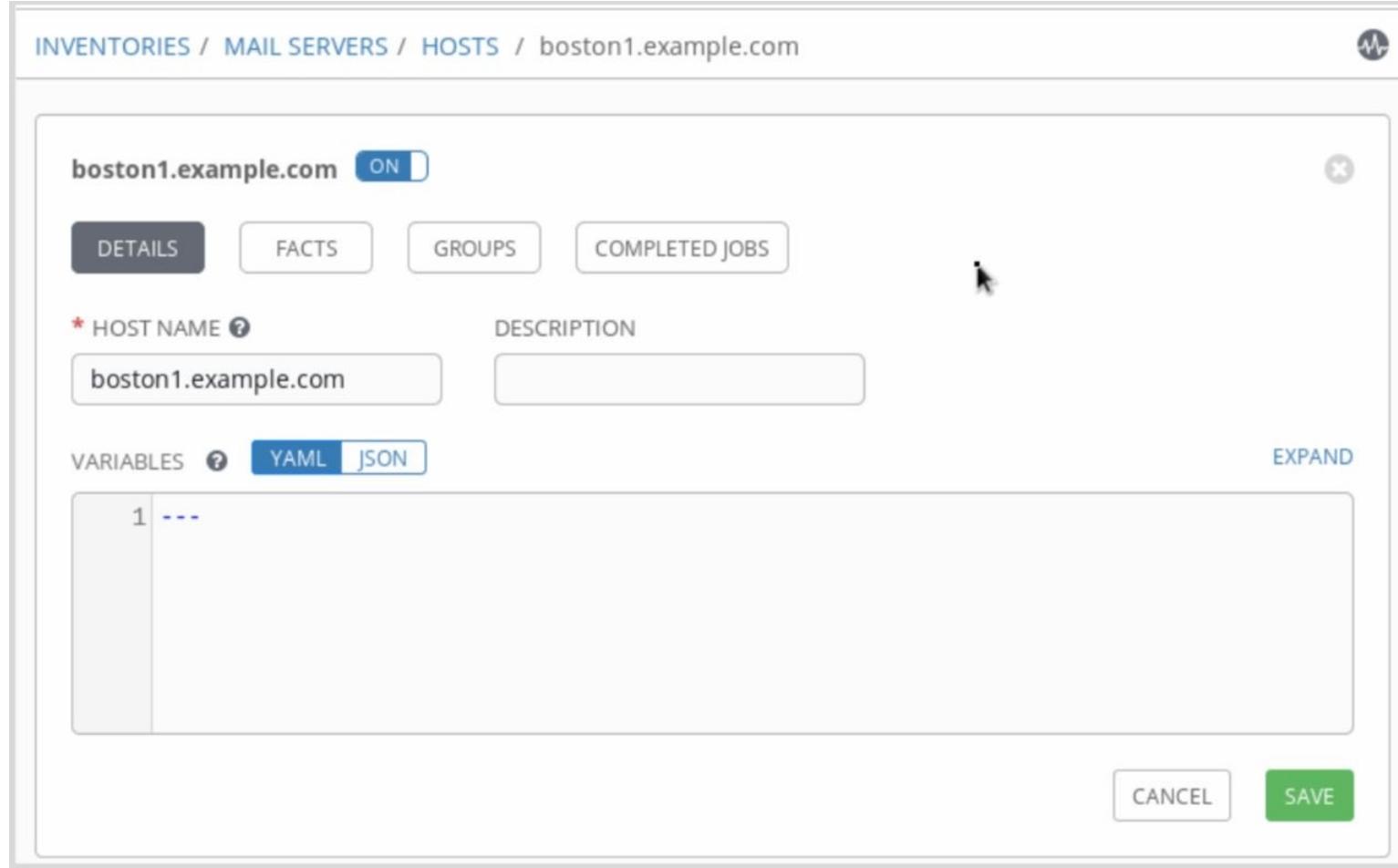
The screenshot shows the Ansible Tower interface for managing inventories. The top navigation bar includes links for Inventories, Mail Servers, All Groups, and southeast. A circular icon with a waveform is also present.

The main content area displays the details for the 'southeast' inventory group. The title 'southeast' is at the top, followed by three tabs: DETAILS (selected), GROUPS, and HOSTS. Below these are fields for NAME (southeast) and DESCRIPTION (empty). A 'VARIABLES' section contains a YAML editor with the following content:

```
1 ---
2 ntp: ntp-se.example.com
```

At the bottom right are 'CANCEL' and 'SAVE' buttons.

# Inventory Host Variables

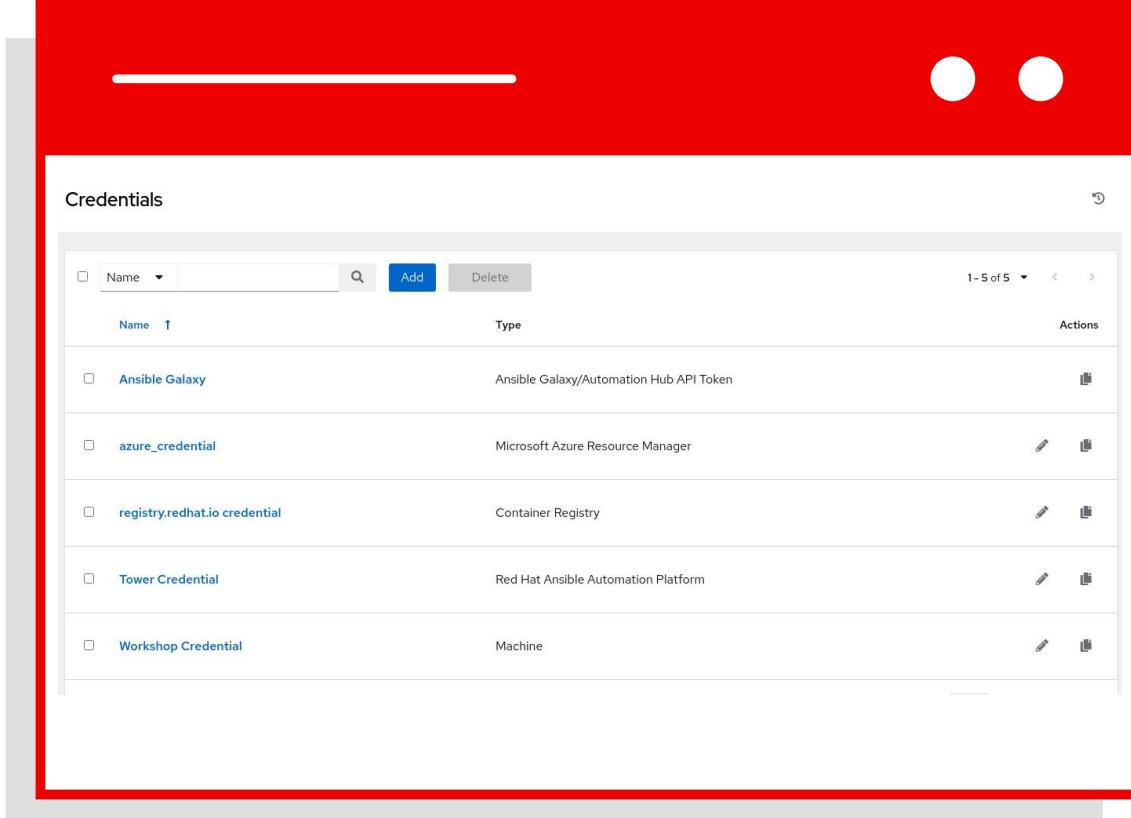


# Credentials

Credentials are utilized by Automation Controller 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.



The screenshot shows a table titled 'Credentials' with a red border. The table has columns for 'Name', 'Type', and 'Actions'. There are five rows of data:

| Name                          | Type                                    | Actions                                                                                                                                                                     |
|-------------------------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ansbile Galaxy                | Ansible Galaxy/Automation Hub API Token |       |
| azure_credential              | Microsoft Azure Resource Manager        |       |
| registry.redhat.io credential | Container Registry                      |       |
| Tower Credential              | Red Hat Ansible Automation Platform     |       |
| Workshop Credential           | Machine                                 |   |

# Credentials

- Create credentials in the UI
- This credential contains information that is used to access managed hosts in the **Inventory**.

CREDENTIALS / EDIT CREDENTIAL

Demo Credential

DETAILS    PERMISSIONS

\* NAME ?  
Demo Credential

DESCRIPTION ?  
Organization

ORGANIZATION  
SELECT AN ORGANIZATION

\* CREDENTIAL TYPE ?  
Machine

TYPE DETAILS

USERNAME  
admin

PASSWORD  
Prompt on launch

SSH PRIVATE KEY HINT: Drag and drop private file on the field below.

The screenshot shows a 'CREDENTIALS / EDIT CREDENTIAL' page. At the top, there's a title 'Demo Credential'. Below it are two tabs: 'DETAILS' (which is active) and 'PERMISSIONS'. Under 'DETAILS', there are fields for 'NAME' (containing 'Demo Credential') and 'DESCRIPTION' (empty). There's also a 'SELECT AN ORGANIZATION' button with a search icon. A section for 'CREDENTIAL TYPE' shows 'Machine' selected. In the 'TYPE DETAILS' section, 'USERNAME' is set to 'admin' and 'PASSWORD' is set to 'Prompt on launch'. At the bottom, there's a note about dragging and dropping an SSH private key file.

# AAP – Inventory and Credentials

## Lab six

