

WORKFORCE DEVELOPMENT



Content Usage Parameters

Content refers to material including instructor guides, student guides, lab guides, lab or hands-on activities, computer programs, etc. designed for use in a training program



1

Content is subject to
copyright protection

2

Content may only be
leveraged by students
enrolled in the training
program

3

Students agree not to
reproduce, make
derivative works of,
distribute, publicly perform
and publicly display in any
form or medium outside of
the training program

4

Content is intended as
reference material only to
supplement the instructor-
led training

Automation Developer



Lab page

<https://jruels.github.io/automation-dev/>



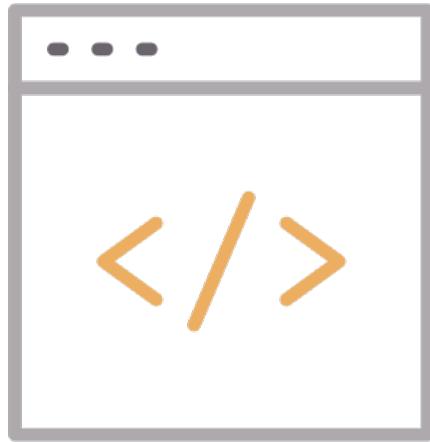
Rolling Updates



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

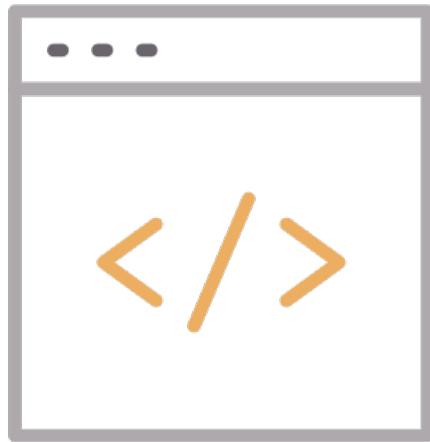
Now that web1 & web2 are complete Ansible continues with web3 & web4

```
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

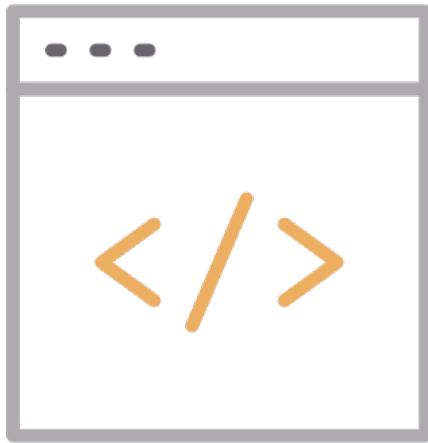


The batch size can be specified as a list with integer, or percent:

```
- 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.

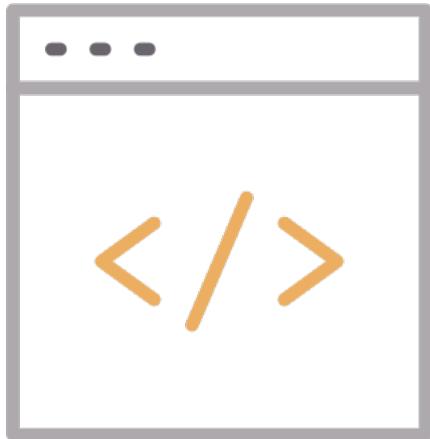
Vault Id

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):

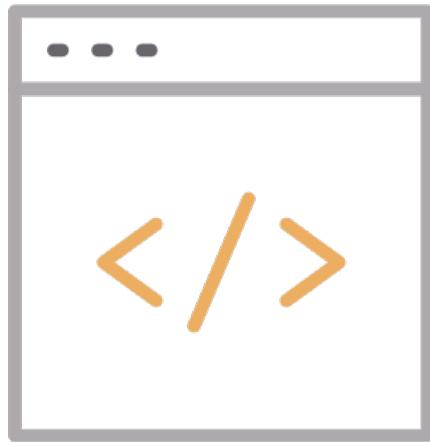
```
--vault-id label@source
```



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
```

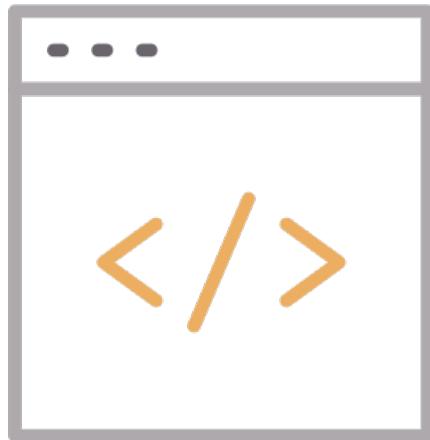
Prompt for vault password

```
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

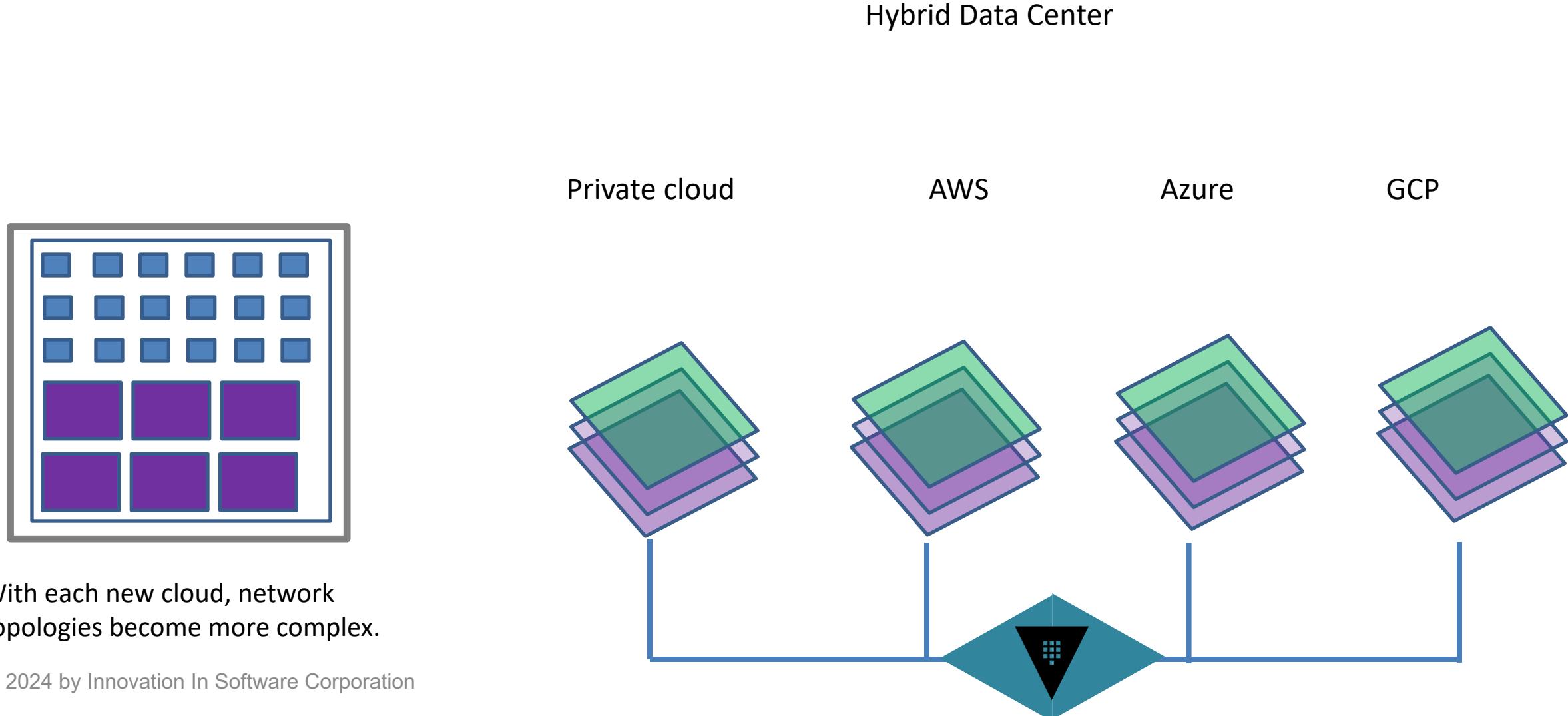
# Rekeying
ansible-vault rekey play.yml task.yml report.yml

# 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
```

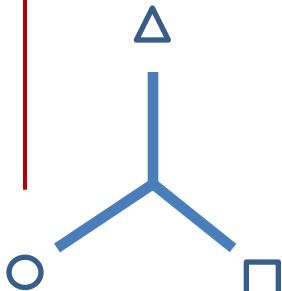
Secure Infrastructure using Vault



Vault Objectives



- Provide single source of secrets for humans and machines .
- Scale to meet security needs of largest organizations.
- Allow for complete secret lifecycle management.



Eliminate Secret Sprawl



Securely Store any Secret



Secret Governance

Use Cases

Secrets Management

Secrets, identity, and access policy management workflow to secure any infrastructure and application resources.

Encryption as a Service

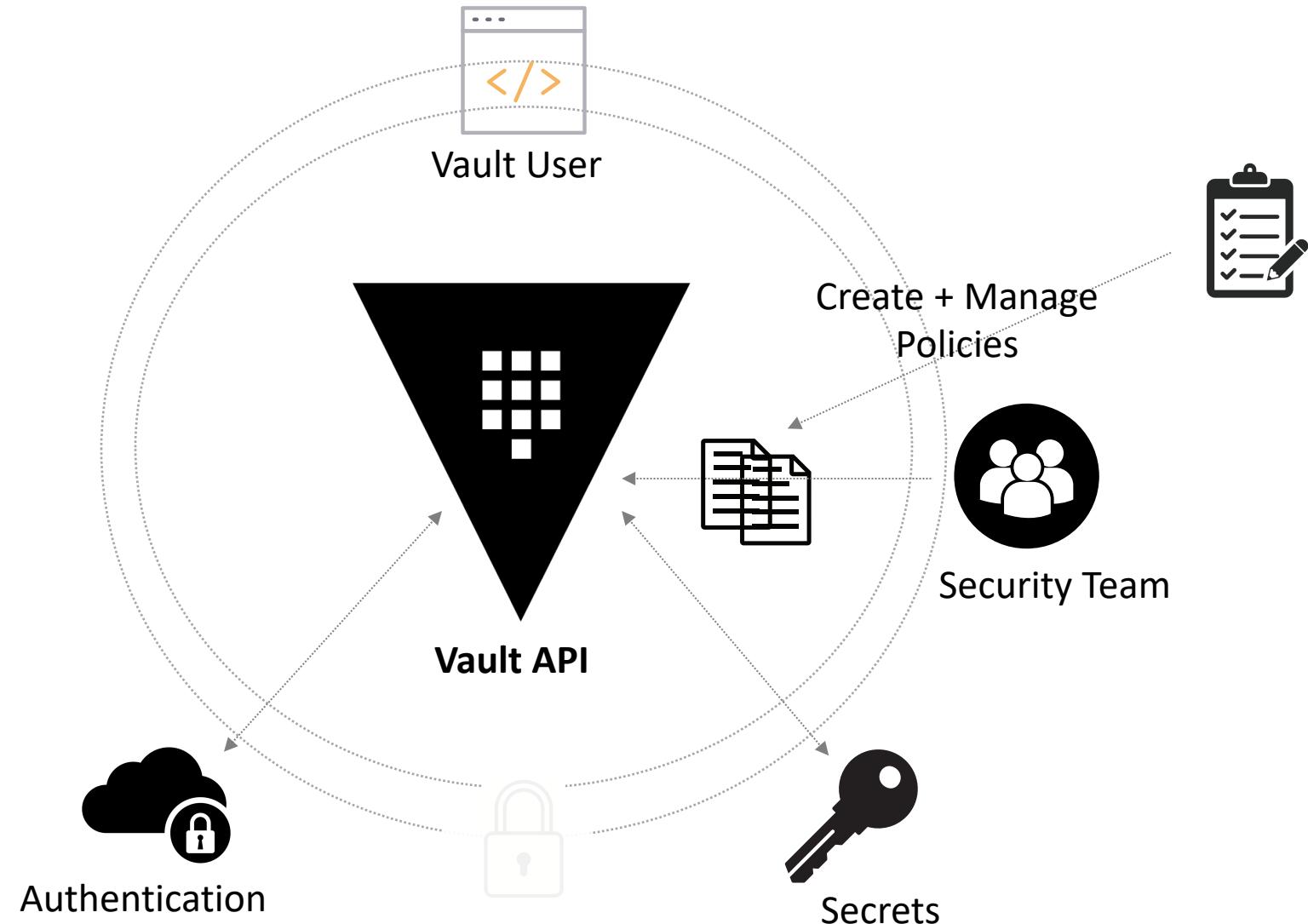
One workflow to create and control the keys used to encrypt your data

Identity Access Management

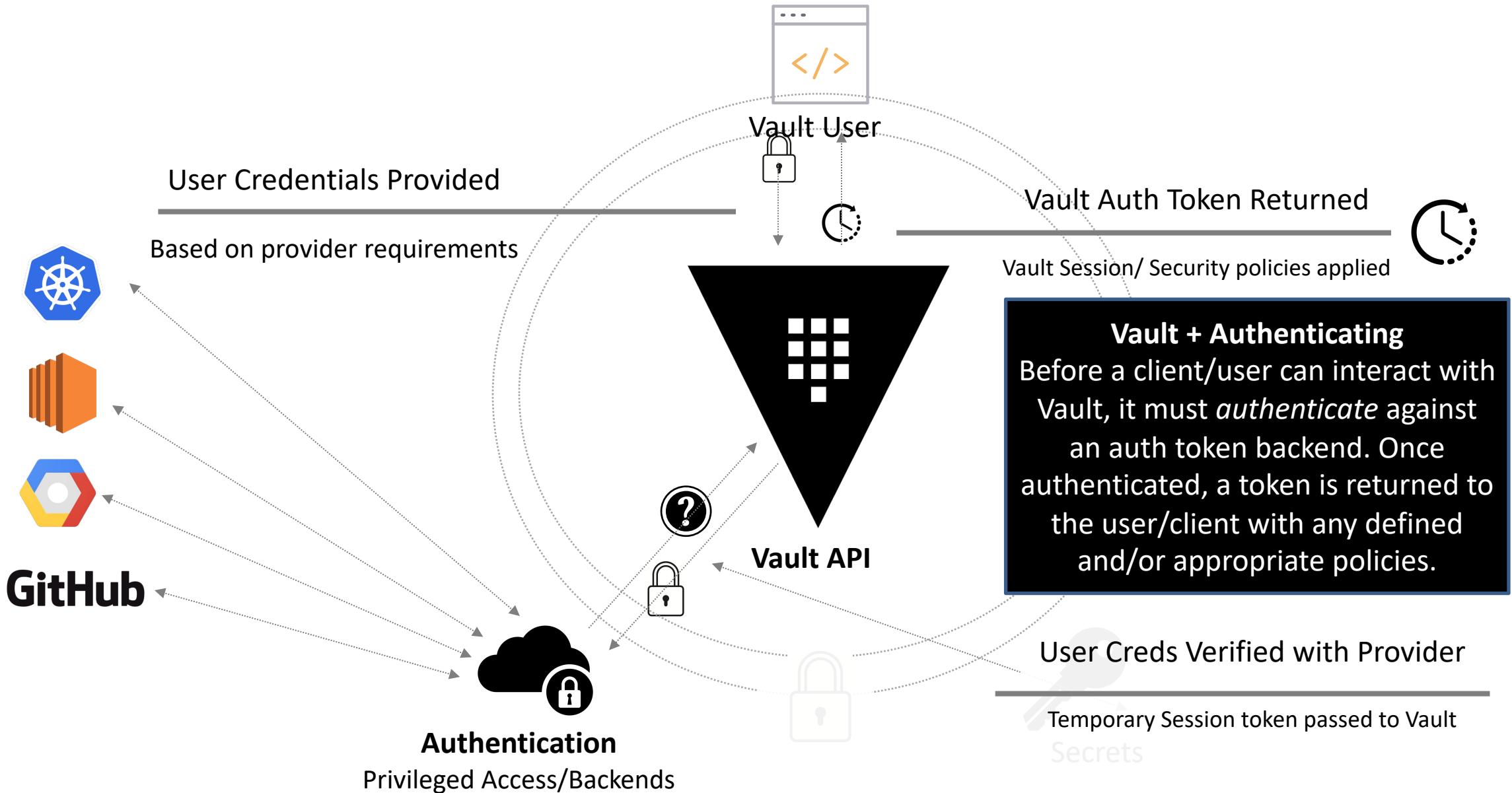
Empower developers and operators to securely make application and infrastructure changes.

Vault - Security Policies

Policies with Vault
Vault uses policies to manage and safeguard access and secret distribution to applications and infrastructure. Policies provide a declarative way to **grant** and **deny** access to operations and paths.

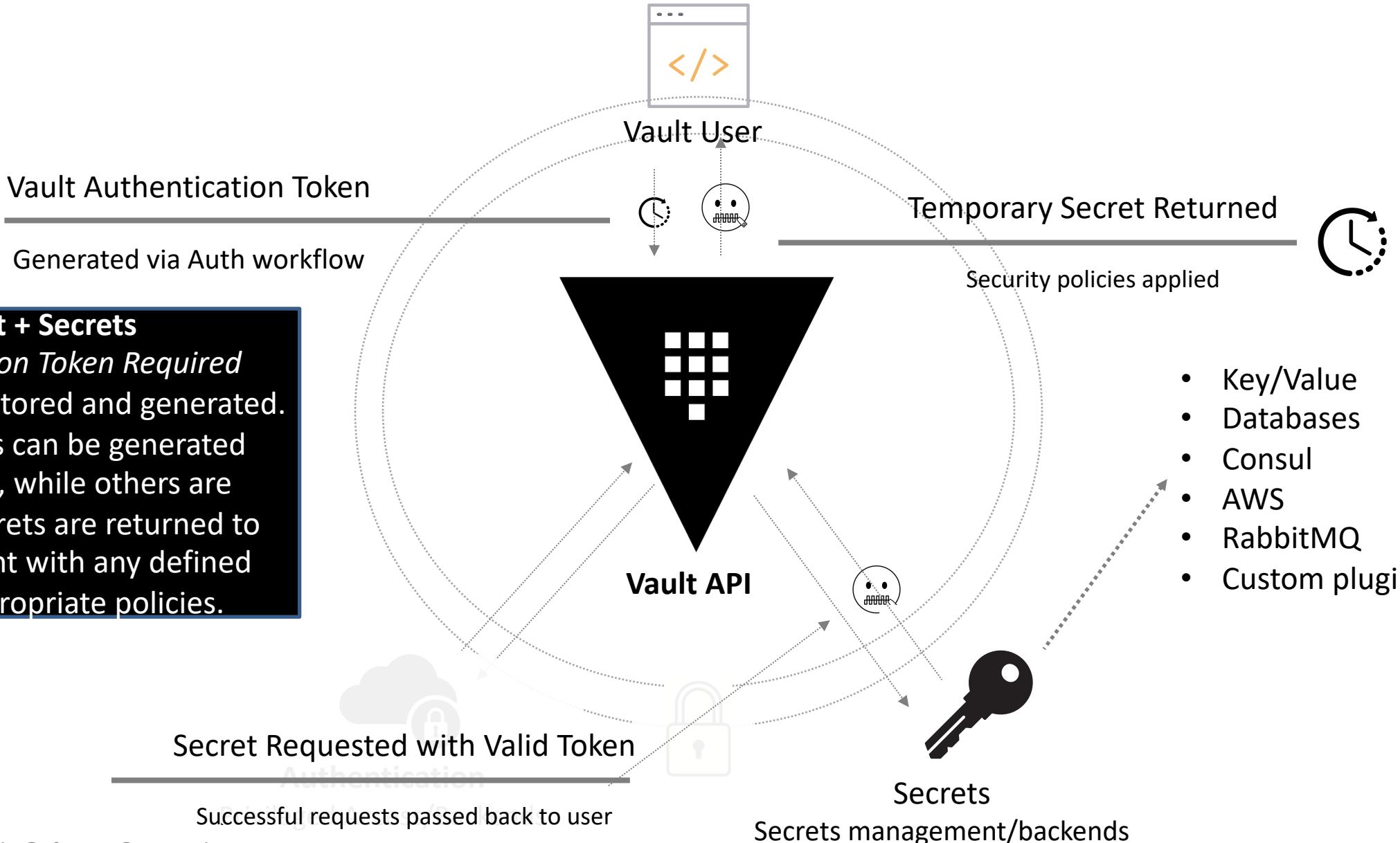


Authentication Workflow



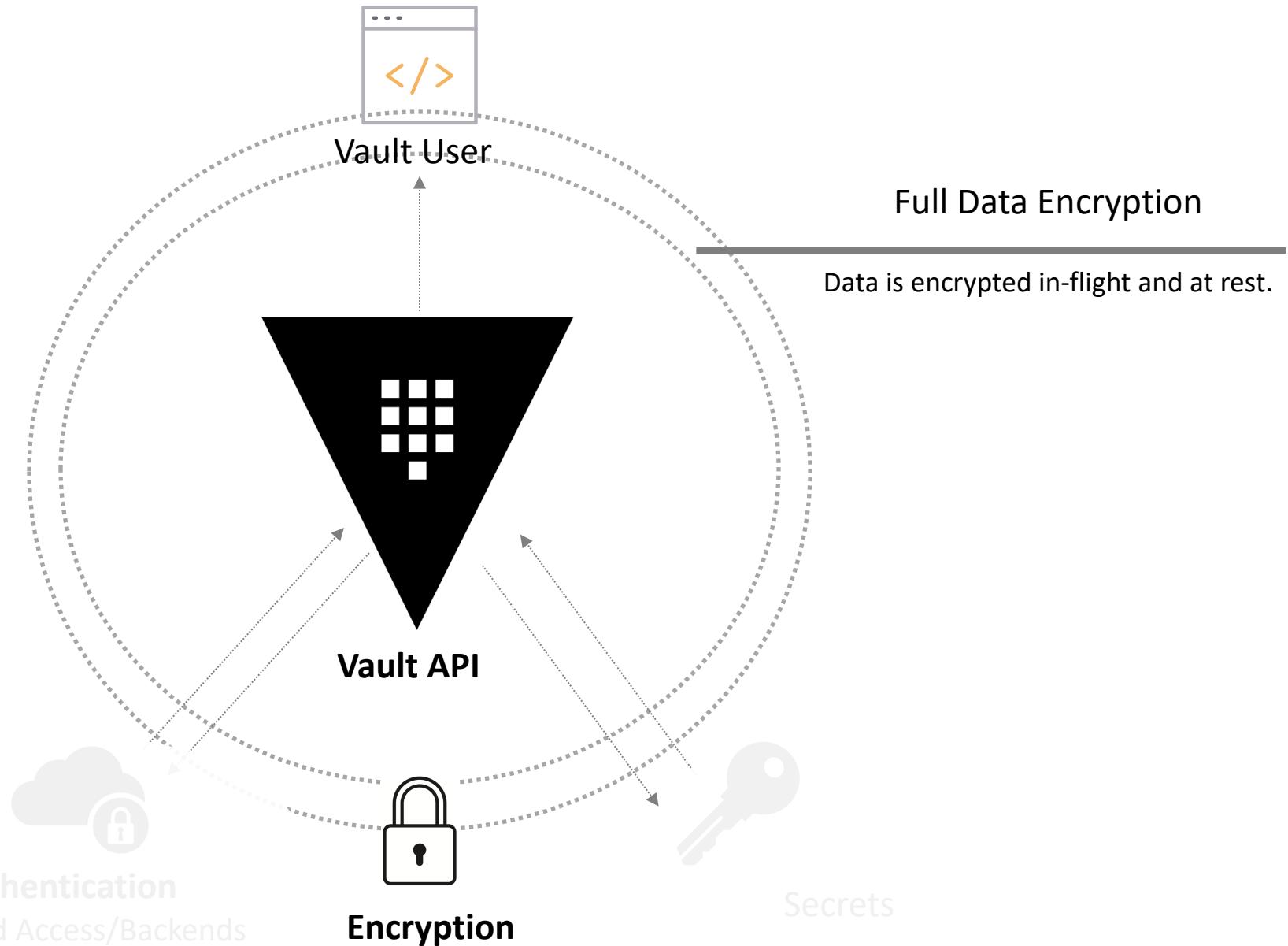
Secrets Workflow

Vault + Secrets
Authentication Token Required
Secrets can be stored and generated.
Some secrets can be generated dynamically, while others are verbatim. Secrets are returned to the user/client with any defined and/or appropriate policies.



Encryption(as a Service)

Encrypt everything
Vault believes that everything should always be encrypted. Vault uses ciphertext wrapping to encrypt all data at rest and in-flight. This minimizes exposure of secrets and sensitive information.



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

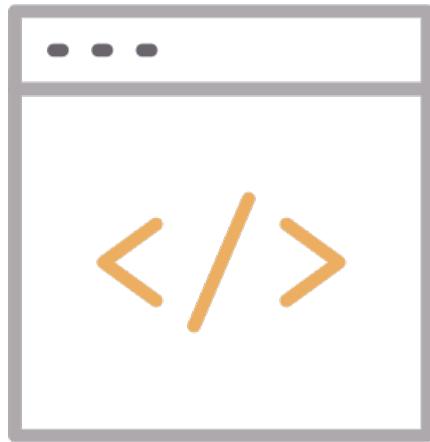
Tags



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.

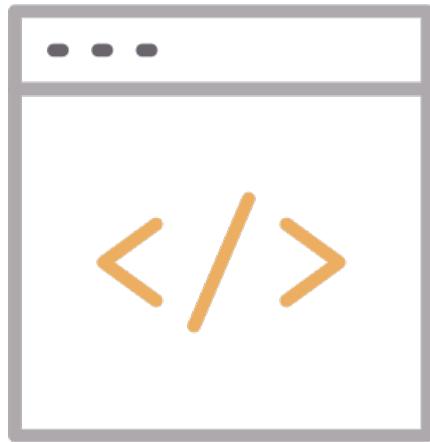
Blocks are useful for applying a tag to many, but not all, of the tasks in your play.

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:

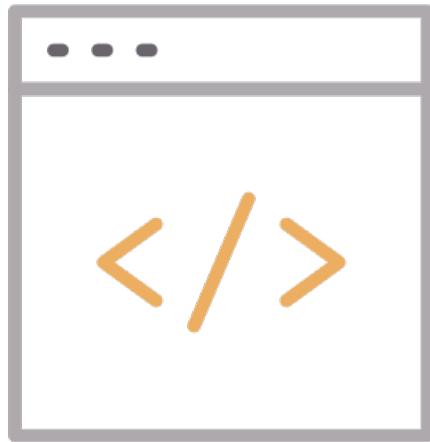
Modules are reusable, standalone scripts that can be used by the Ansible API, the `ansible` command, or the `ansible-playbook` command.

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
```

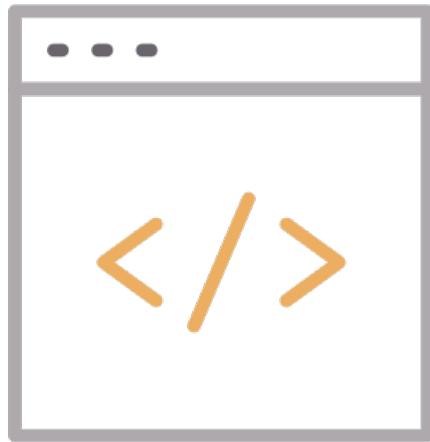
You can install from the community, or any .tar.gz file.

```
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
```

# Lab: Ansible Roles

# Ansible Facts

- Just like variables, really...
- ..but: coming from the host itself!
- Check them out with the setup module

```
tasks:
 - name: Collect all facts of host
 setup:
 gather_subset:
 - 'all'
```

# Ansible Facts



Ansible playbooks

```

```

- name: facts playbook  
 hosts: localhost

tasks:

- name: Collect all facts of host
- setup:
  - gather\_subset:
    - 'all'

```
$ ansible-navigator run playbook.yml
```

# Conditionals Via Vars

Example of using a variable labeled *my\_mood* and using it as a conditional on a particular task.

```
vars:
 my_mood: happy

tasks:
 - name: task, based on my_mood var
 debug:
 msg: "Yay! I am{{ my_mood }}!"
 when: my_mood == "happy"
```

# Conditionals Via Vars



Ansible Conditionals

```

- name: variable playbook test
hosts: localhost

vars:
 my_mood: happy

tasks:
 - name: task, based on my_mood var
 debug:
 msg: "Yay! I am{{ my_mood }}!"
 when: my_mood == "happy"
```

Alternatively

```
- name: task, based on my_mood var
debug:
 msg: "Ask at your own risk. I'm {{ my_mood }}!"
when: my_mood == "grumpy"
```

# Conditionals With Facts



Ansible Conditionals w/ Facts

```

- name: variable playbook test
 hosts: localhost

 tasks:
 - name: Install apache
 apt:
 name: apache2
 state: latest
 when: ansible_distribution == 'Debian' or
 ansible_distribution == 'Ubuntu'

 - name: Install httpd
 yum:
 name: httpd
 state: latest
 when: ansible_distribution == 'RedHat'
```

# Task State



Using Previous Task State

```

- name: variable playbook test
 hosts: localhost

 tasks:
 - name: Ensure httpd package is present
 yum:
 name: httpd
 state: latest
 register: http_results

 - name: Restart httpd
 service:
 name: httpd
 state: restart
 when: httpd_results.changed
```

# Variables And Loops



Ansible Variables & Loops

```

- name: Ensure users
 hosts: node1
 become: yes

 tasks:
 - name: Ensure user is present
 user:
 name: dev_user
 state: present

 - name: Ensure user is present
 user:
 name: qa_user
 state: present

 - name: Ensure user is present
 user:
 name: prod_user
 state: present
```

# Variables And Loops



Ansible Variables & Loops

```

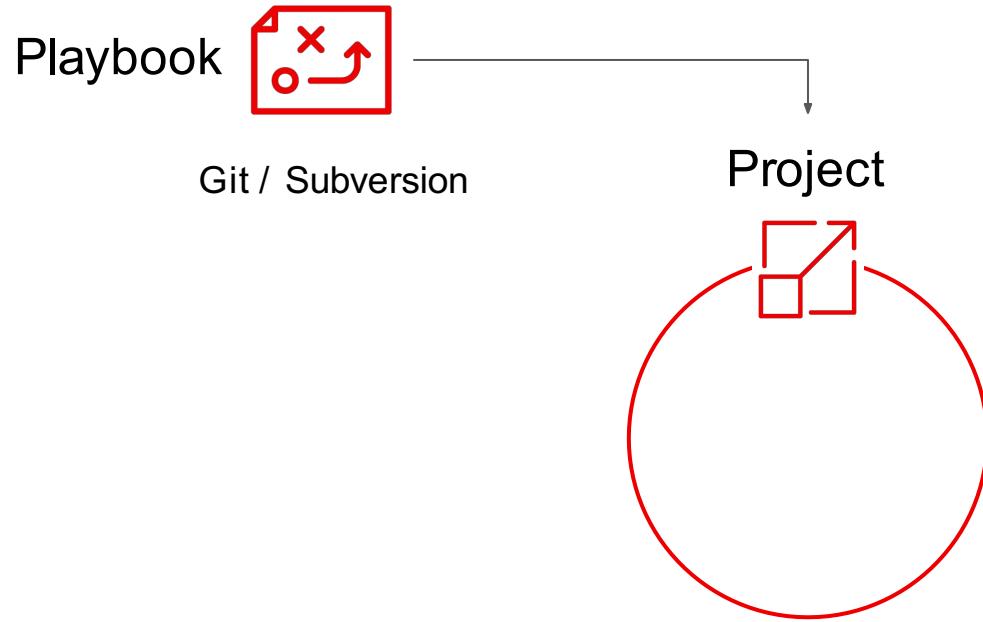
- name: Ensure users
 hosts: node1
 become: yes

 tasks:
 - name: Ensure user is present
 user:
 name: "{{item}}"
 state: present
 loop:
 - dev_user
 - qa_user
 - prod_user
```

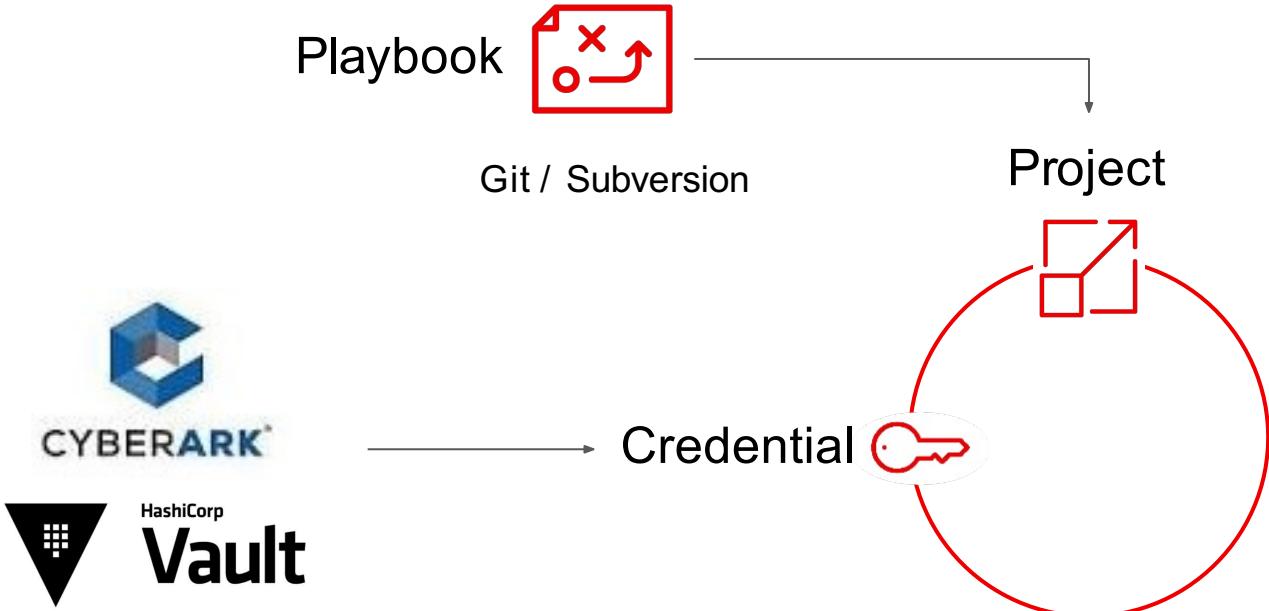
# Automation Controller



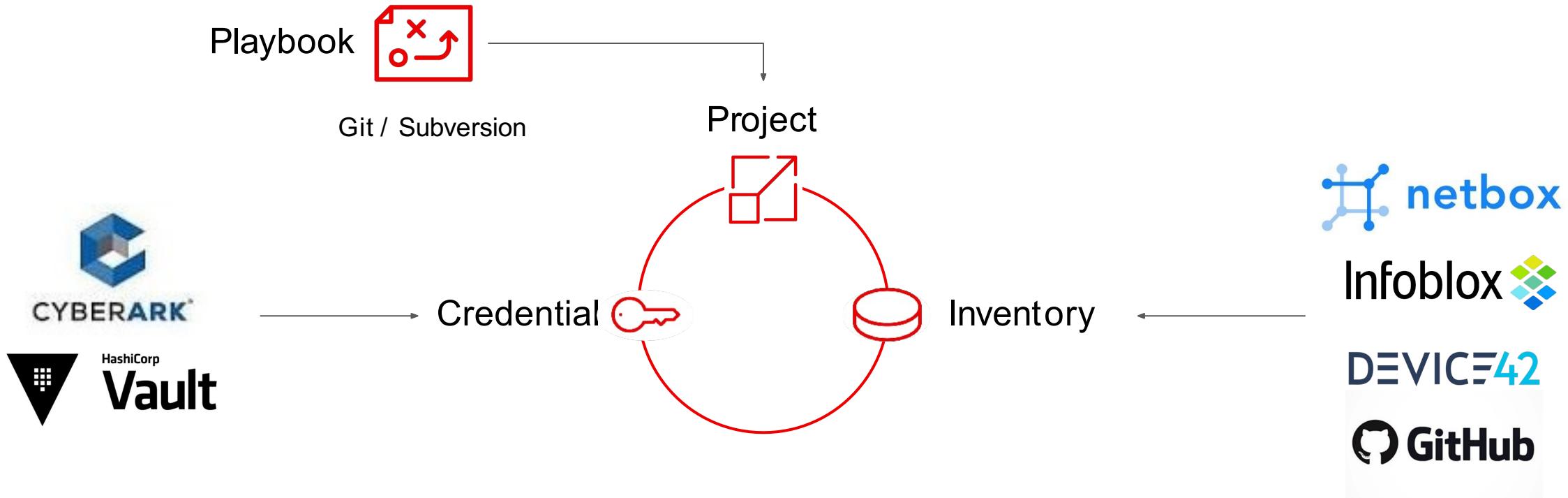
# Anatomy of An Automation Job



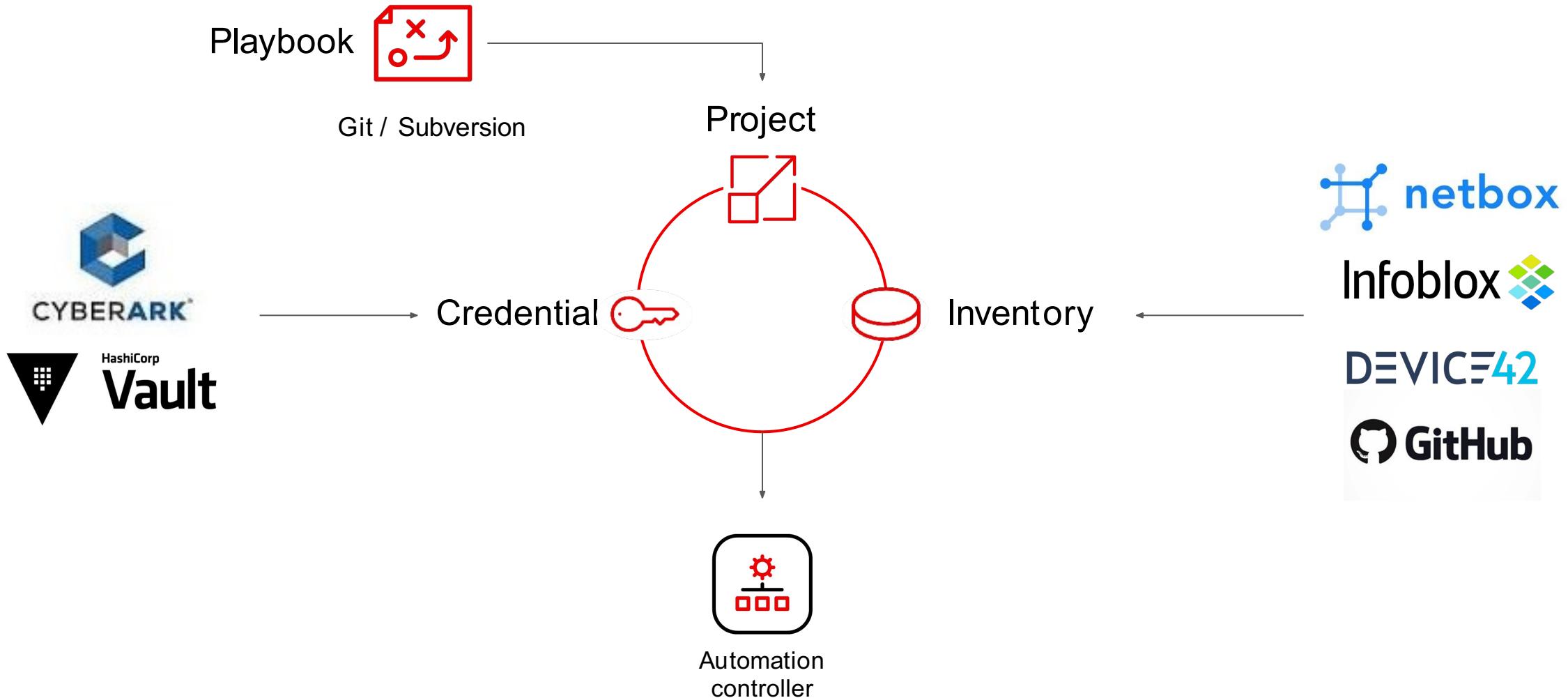
# Anatomy of An Automation Job



# 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. The bottom of the screen shows pagination information: 1 - 2 of 2 items, page 1 of 1 page.

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

# Team

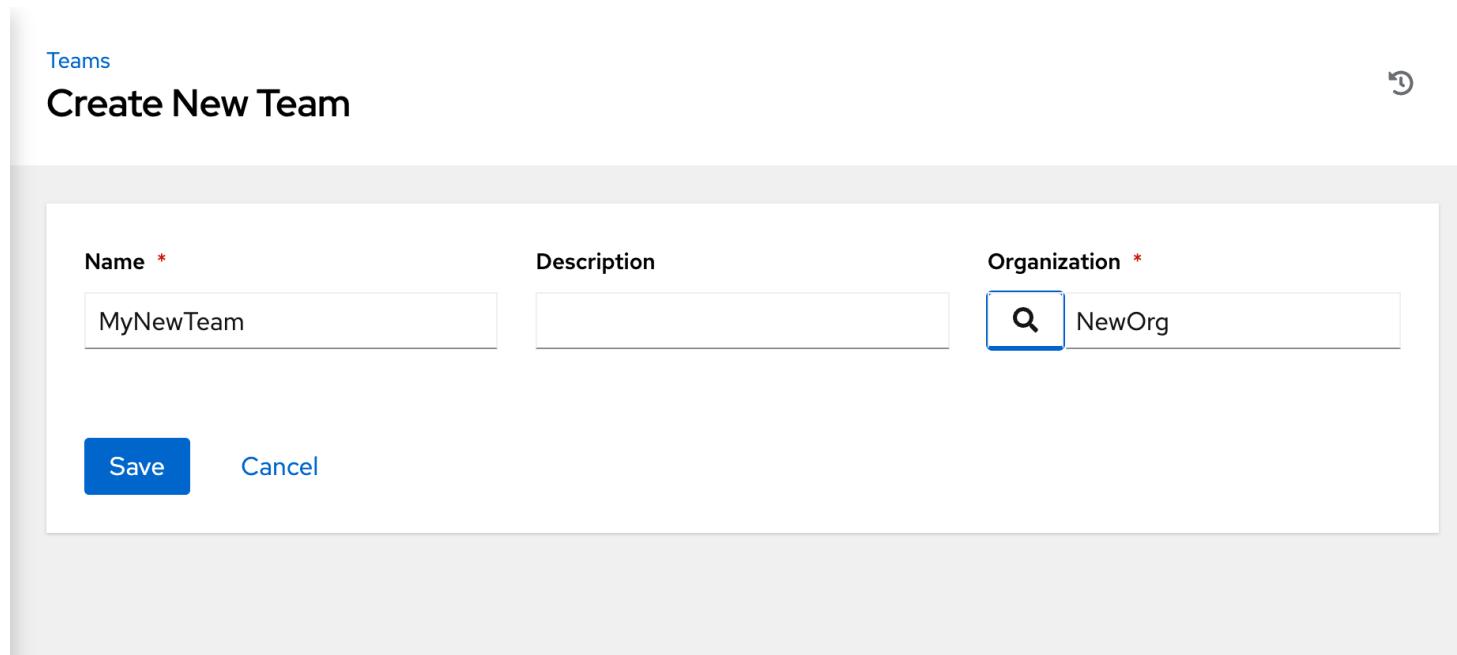
- You can apply permissions at the team level.

Teams

Create New Team

Save Cancel

| Name *    | Description | Organization * |
|-----------|-------------|----------------|
| MyNewTeam |             | NewOrg         |



# 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

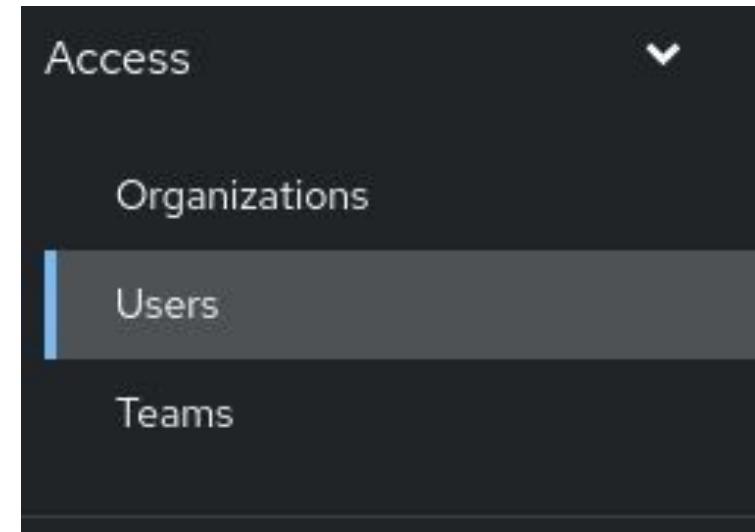
The screenshot shows a user creation interface. At the top, there are fields for First Name (User), Last Name (One), and Email (user1@domain.com). Below these are fields for Username (user1), Password, and Confirm Password, each with a visibility icon. A dropdown menu for User Type is open, showing options: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

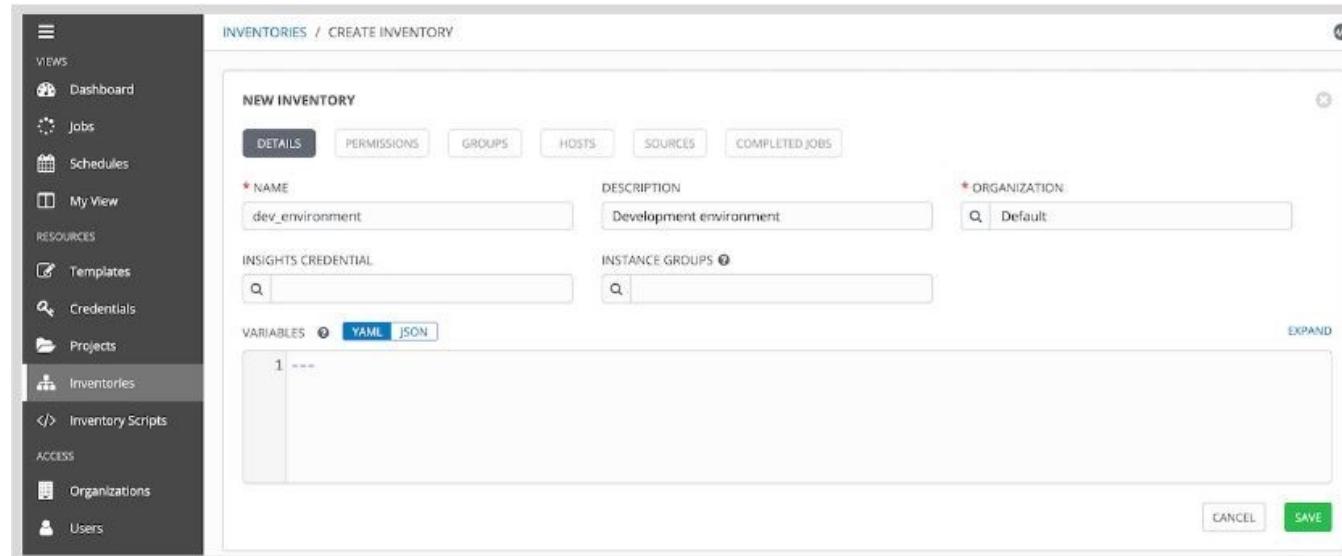
# User Management

- 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.



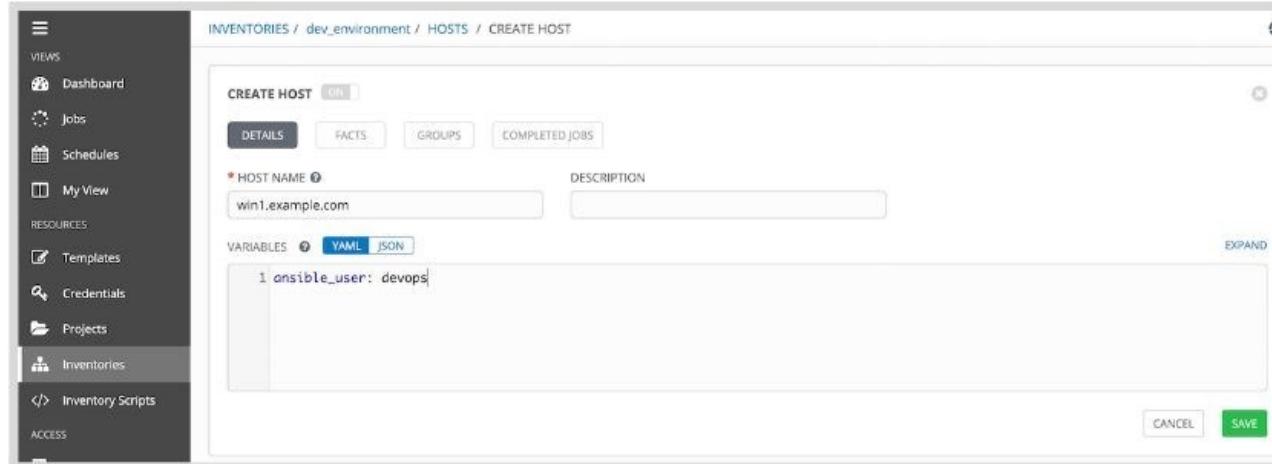
# Inventory

- 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**.

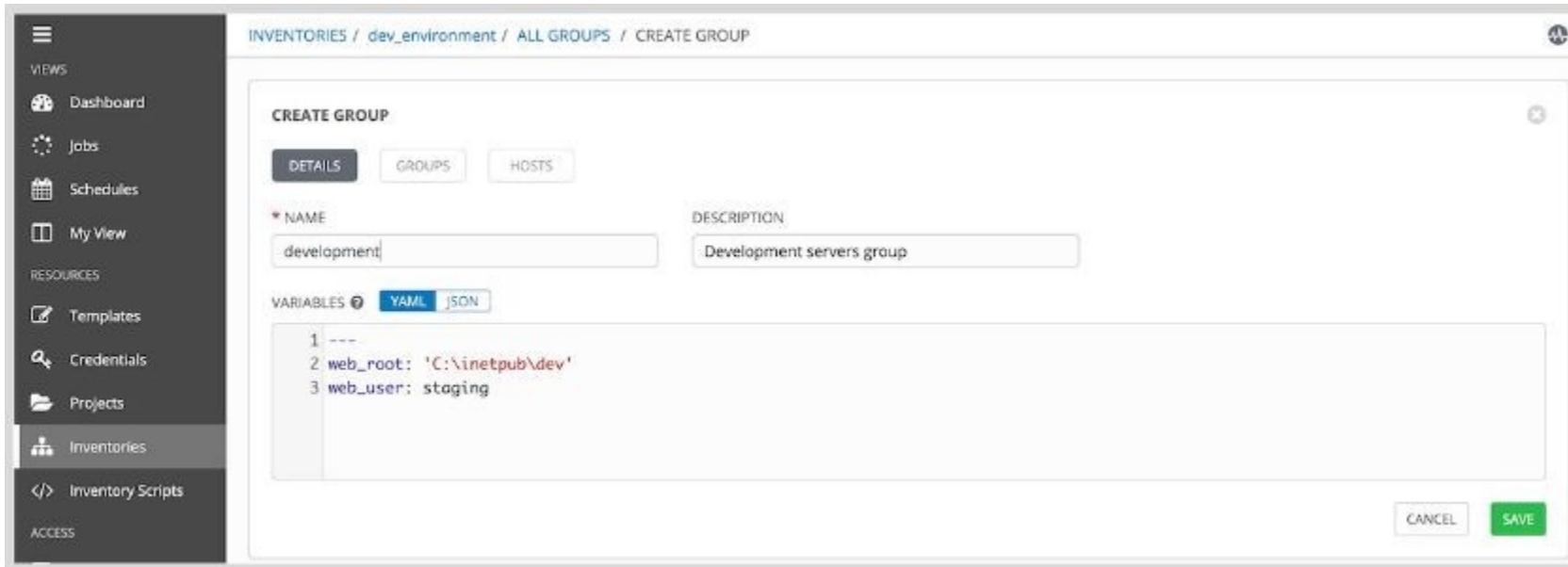


# Inventory Groups

- 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**.

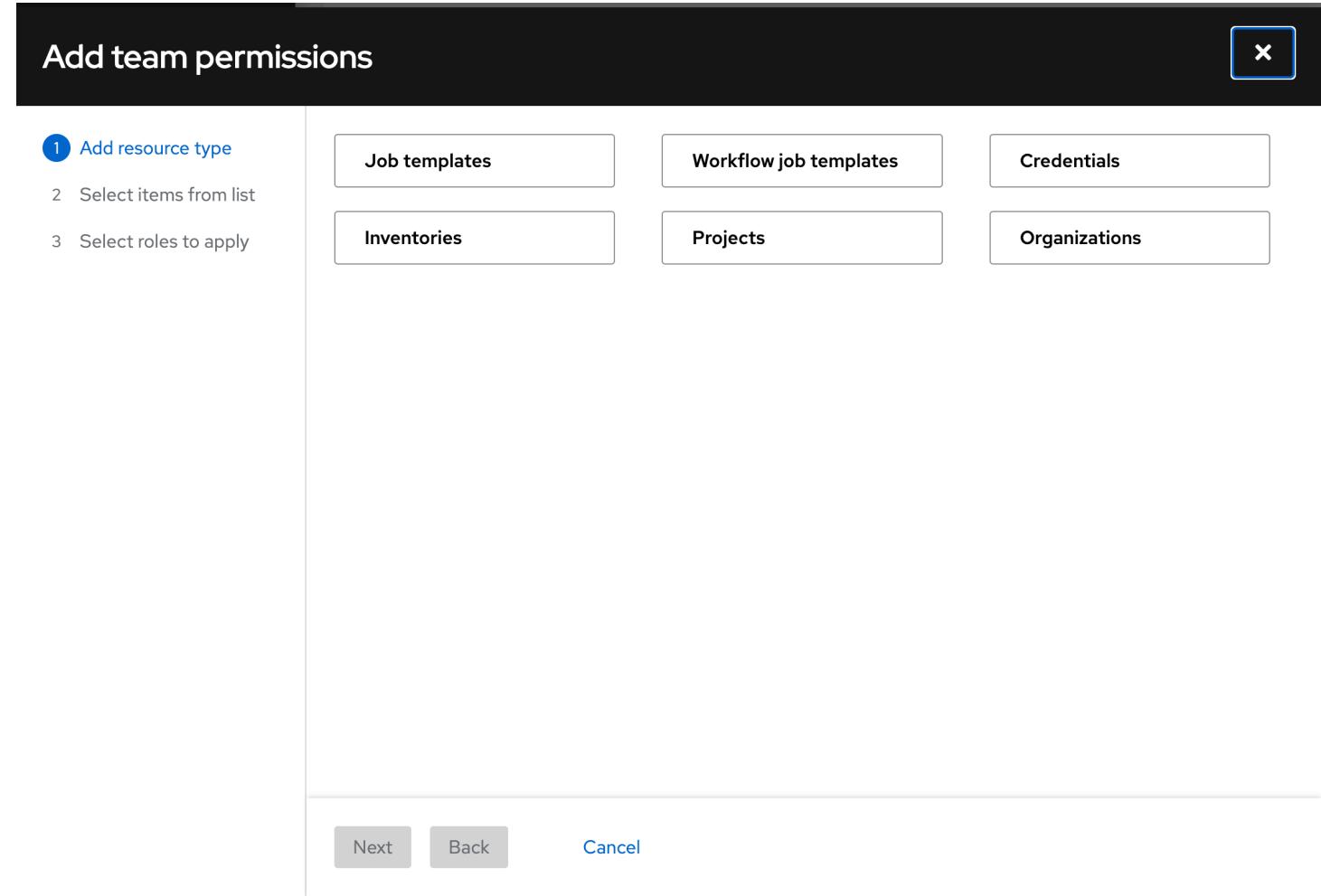


# Inventory Groups

- 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                                                                                                                                                                                                                                                                                |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Admin</b>  | 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> . |
| <b>Use</b>    | 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.                                                                                             |
| <b>Ad Hoc</b> | The inventory <b>Ad Hoc</b> role grants users the ability to use the inventory to execute ad hoc commands.                                                                                                                                                                                 |
| <b>Update</b> | The inventory <b>Update</b> role grants users the ability to update a dynamic inventory from its external data source.                                                                                                                                                                     |
| <b>Read</b>   | 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

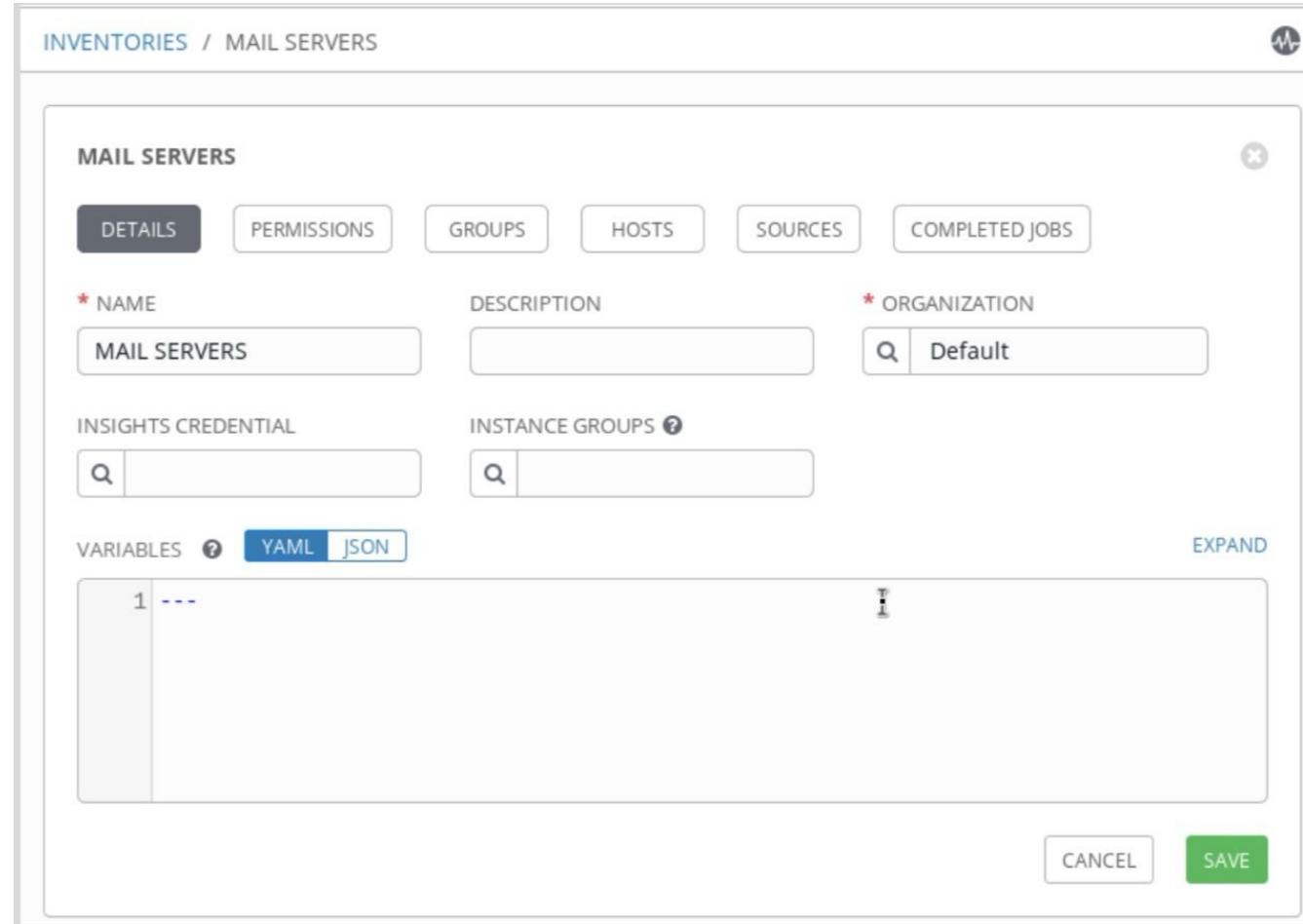
INSIGHTS CREDENTIAL    INSTANCE GROUPS

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.

The 'NAME' field is marked with a red asterisk (\*) and contains the value 'southeast'. The 'DESCRIPTION' field is empty.

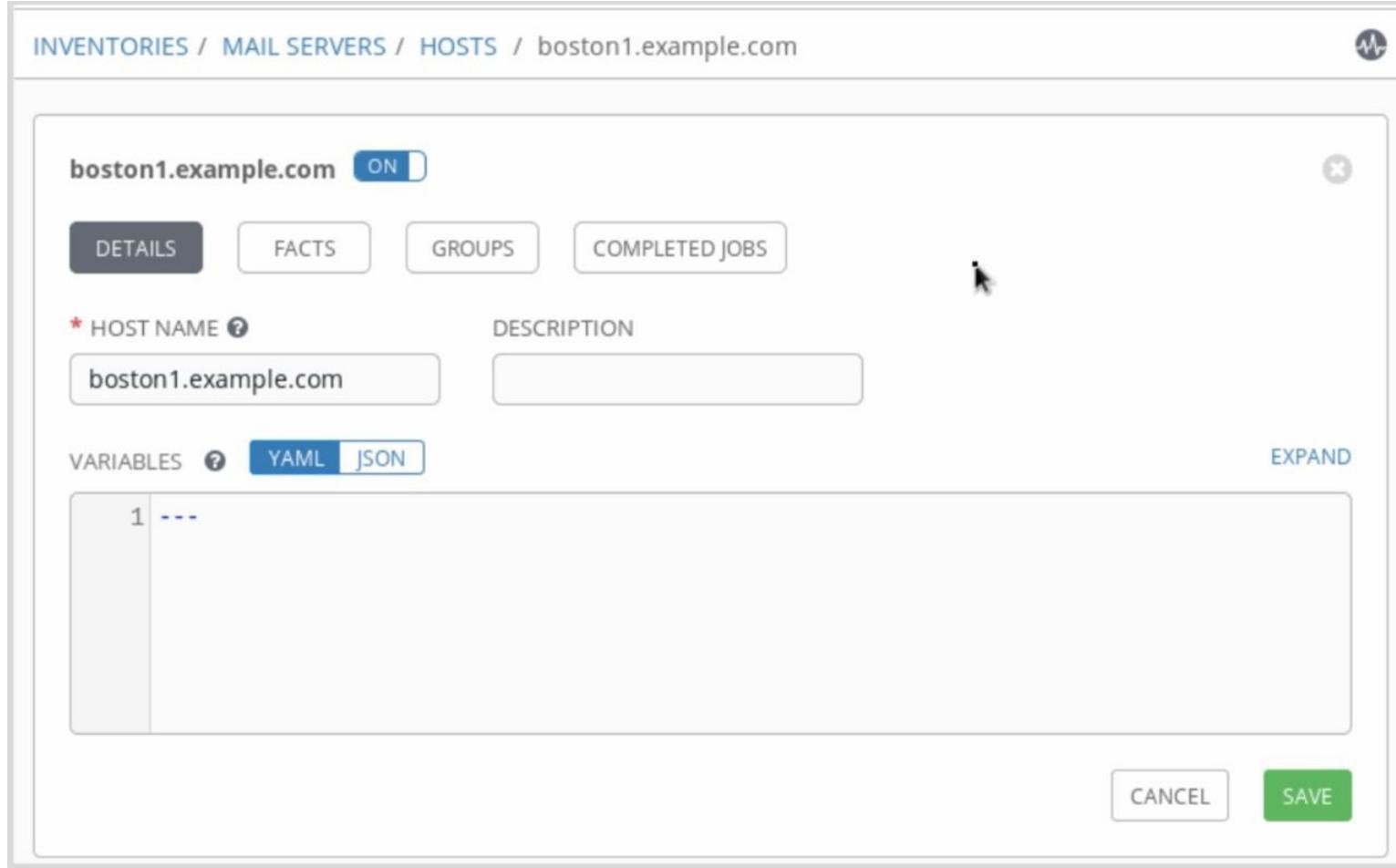
The 'VARIABLES' section shows two entries:

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

Below the variables, there are YAML and JSON tabs, with 'YAML' currently selected.

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

# Inventory Host Variables

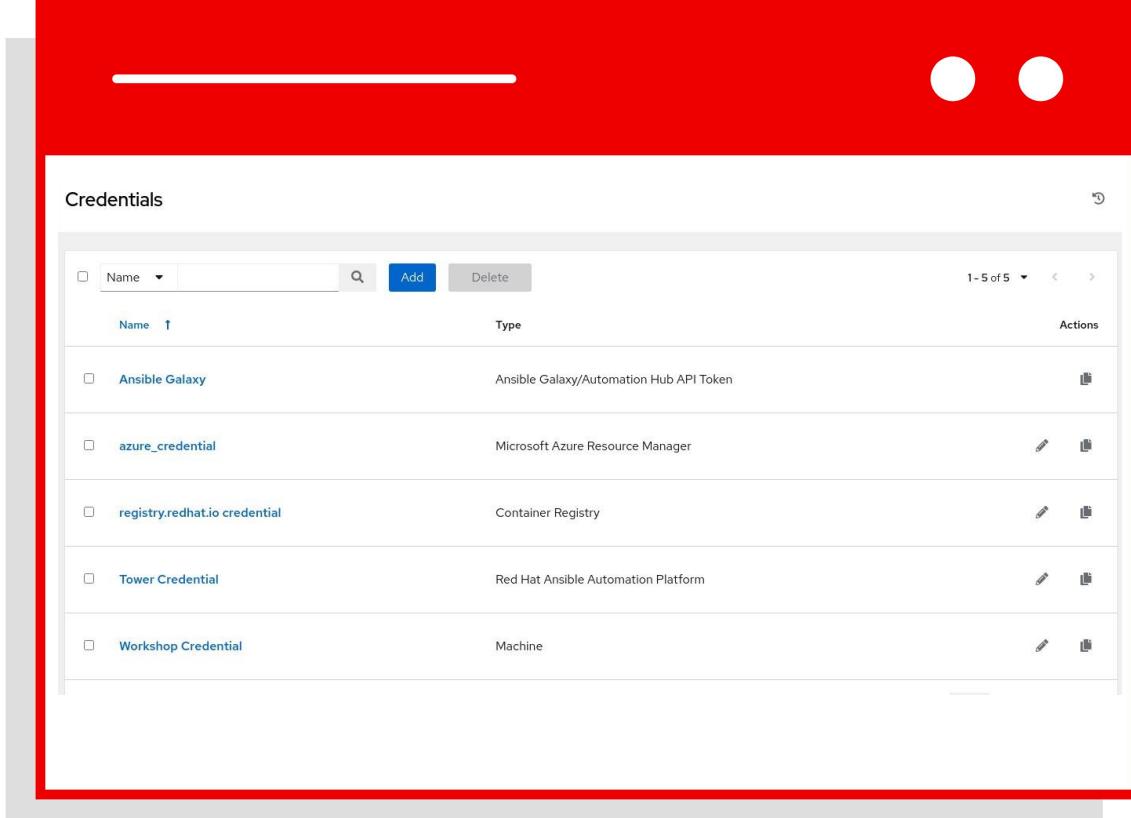


# Credentials

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

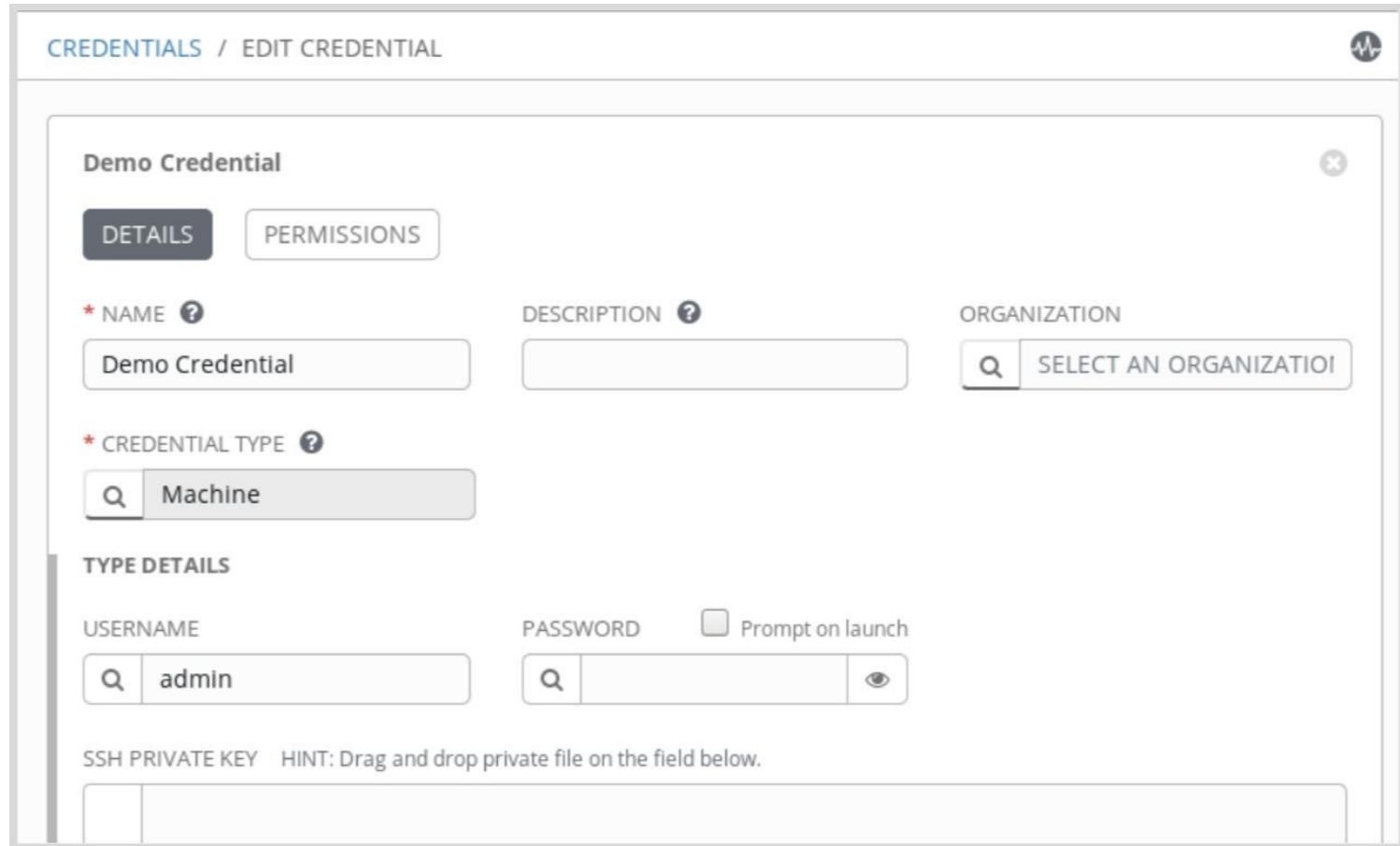
\* CREDENTIAL TYPE ?  
Machine

TYPE DETAILS

USERNAME  
admin

PASSWORD  
Prompt on launch

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

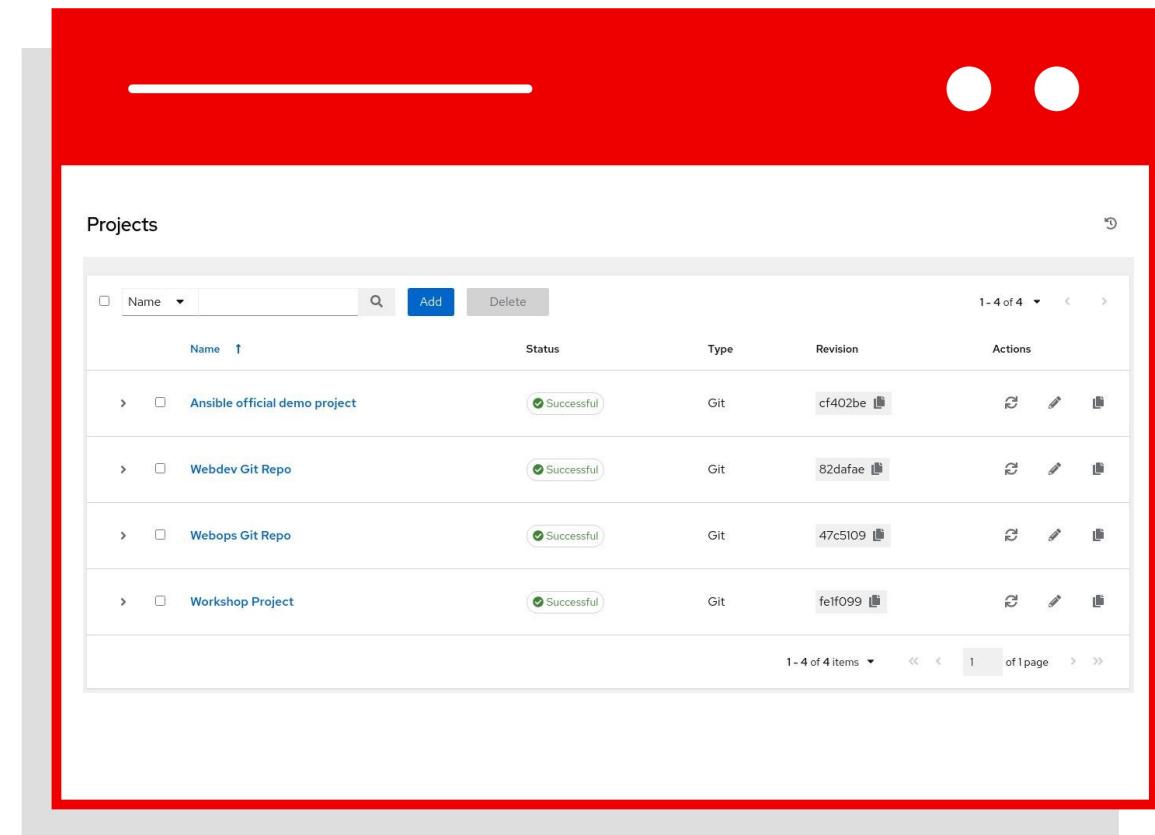


# Lab: AAP Inventories, credentials, and ad-hoc commands

# PROJECT

A project is a logical collection of Ansible Playbooks, represented in Ansible Automation Controller.

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.



The screenshot shows a table titled "Projects" with the following data:

| Name                          | Status     | Type | Revision | Actions                                                                                                                                                                 |
|-------------------------------|------------|------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ansible official demo project | Successful | Git  | cf402be  |   |
| Webdev Git Repo               | Successful | Git  | 82dafaef |   |
| Webops Git Repo               | Successful | Git  | 47c5109  |   |
| Workshop Project              | Successful | Git  | fef099   |   |

# PROJECT

- Under **Projects** in the left navigation bar, an example project named **Demo Project** is displayed.
- This project is configured to get Ansible project materials, including a playbook, from a public Git repository.
- You can also prepare a credential so you can access a private Git repository that needs authentication.

The screenshot shows the 'Demo Project' configuration page in the Ansible Tower web interface. The page has a header 'PROJECTS / Demo Project' and a title 'Demo Project'. It features several tabs: DETAILS (selected), PERMISSIONS, NOTIFICATIONS, JOB TEMPLATES, and SCHEDULES. The 'DETAILS' tab contains fields for 'NAME' (Demo Project), 'DESCRIPTION' (empty), 'ORGANIZATION' (Default), 'SCM TYPE' (Git), 'SCM URL' (https://github.com/ansible/ansible-tower.git), 'SCM BRANCH/TAG/COMMIT' (empty), 'SCM CREDENTIAL' (empty), 'SCM UPDATE OPTIONS' (CLEAN, DELETE ON UPDATE unchecked, UPDATE REVISION ON LAUNCH checked), and 'CACHE TIMEOUT (SECONDS)' (0). At the bottom right are 'CANCEL' and 'SAVE' buttons.

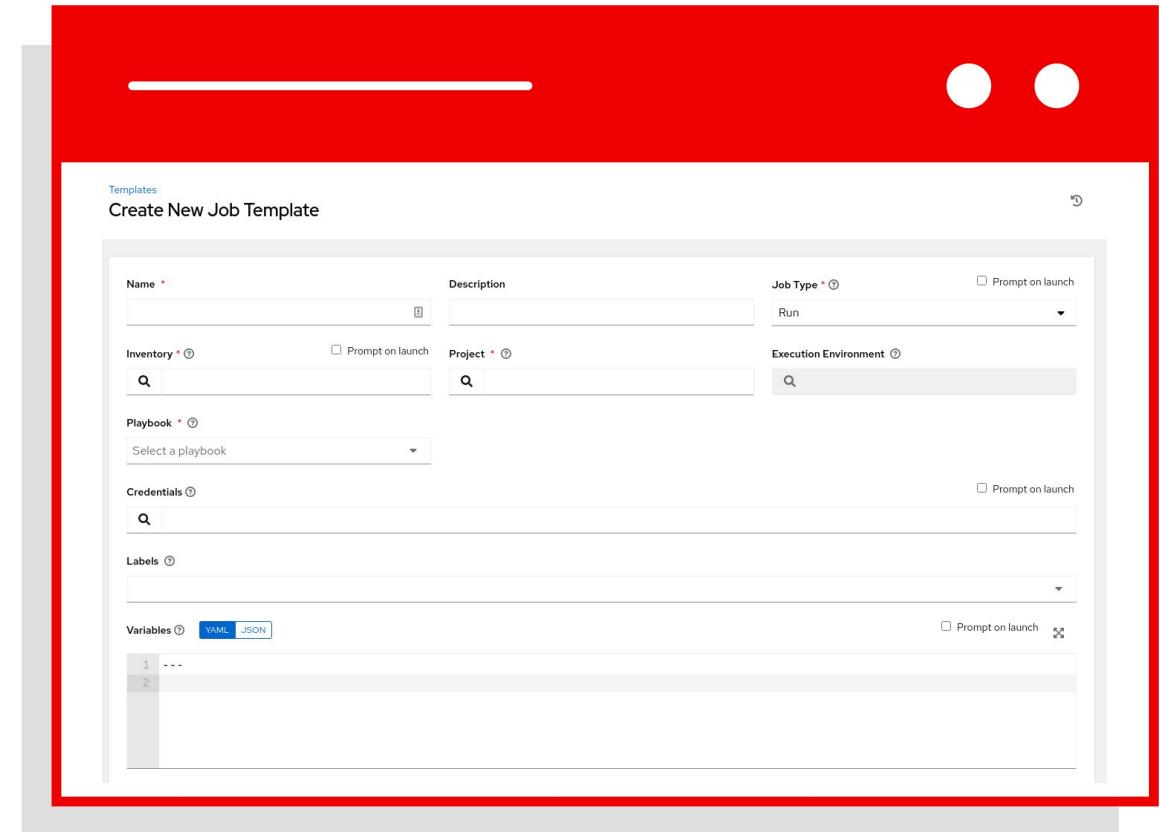
# JOB TEMPLATES

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



# JOB TEMPLATES

- Under **Templates** in the left navigation bar, an example template called **Demo Job Template** is displayed.
- This job template runs the `hello_world.yml` playbook from **Demo Project** on the hosts in **Demo Inventory**, using **Demo Credential** to authenticate access.
- This initial job template can be used to test Ansible Tower.

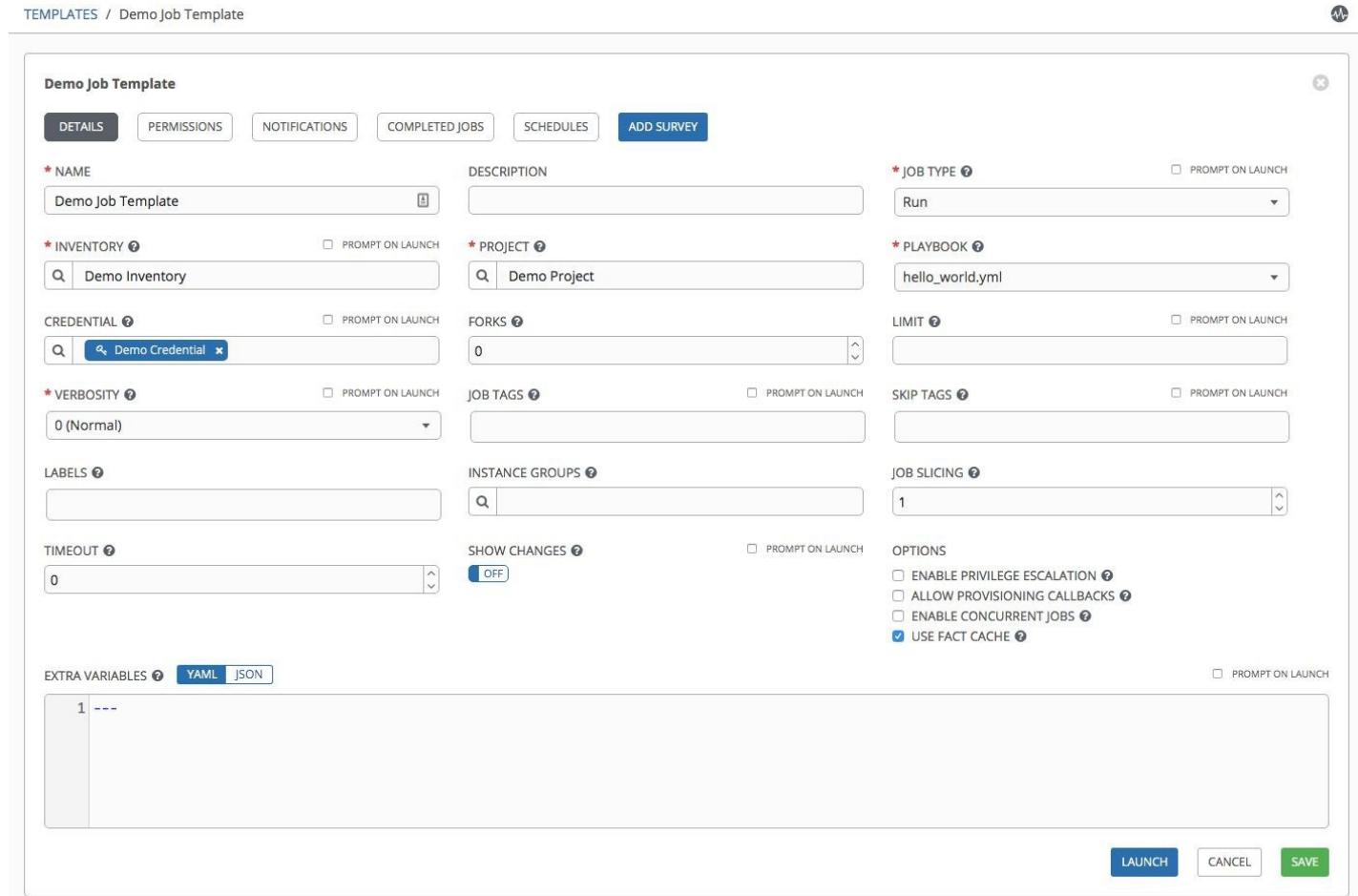
TEMPLATES / Demo Job Template

**Demo Job Template**

DETAILS    PERMISSIONS    NOTIFICATIONS    COMPLETED JOBS    SCHEDULES    ADD SURVEY

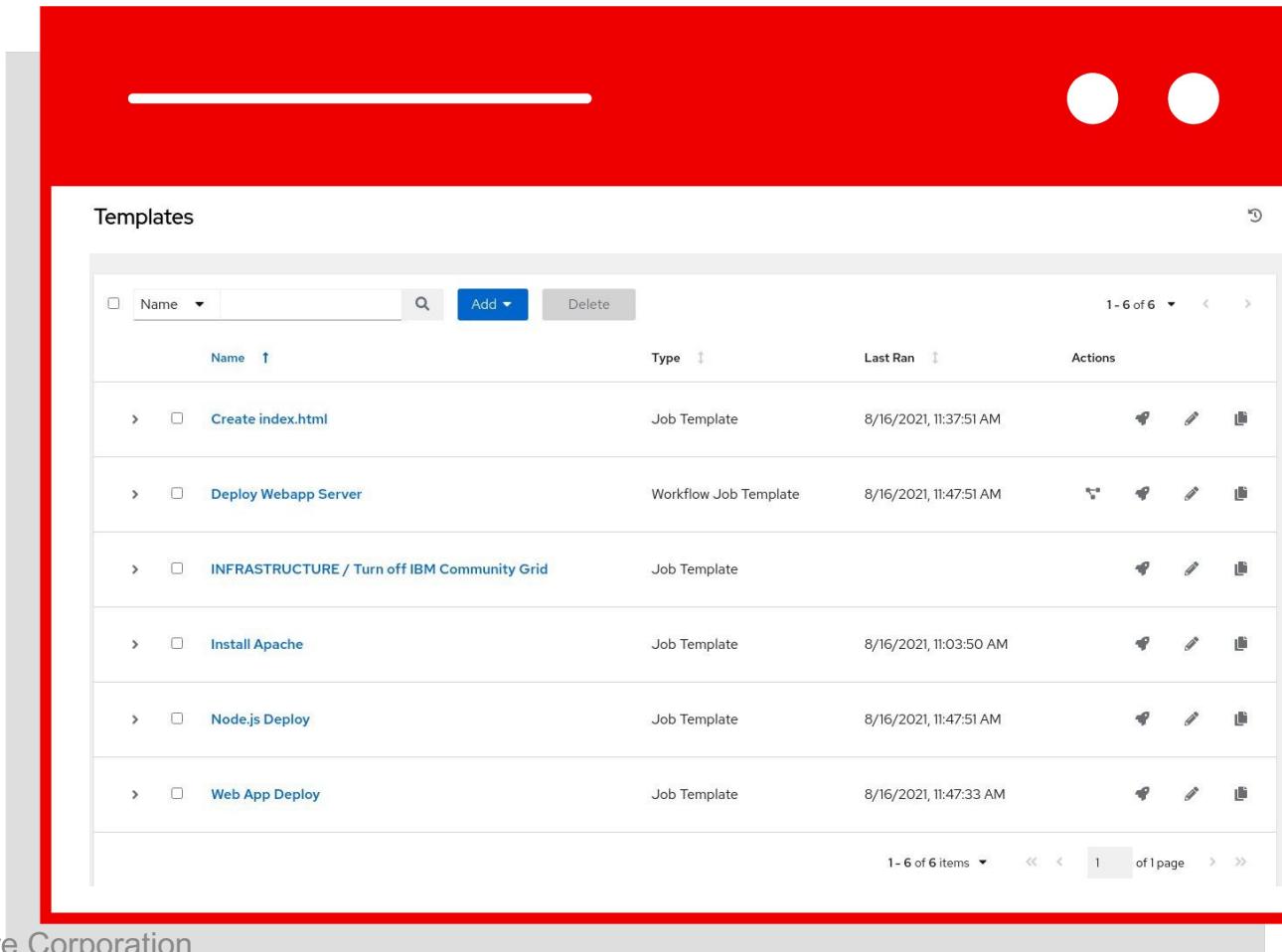
|                                       |                           |                               |
|---------------------------------------|---------------------------|-------------------------------|
| * NAME<br>Demo Job Template           | DESCRIPTION               | * JOB TYPE<br>Run             |
| * INVENTORY<br>Demo Inventory         | * PROJECT<br>Demo Project | * PLAYBOOK<br>hello_world.yml |
| CREDENTIAL<br>Demo Credential         | FORKS<br>0                | LIMIT                         |
| * VERBOSITY<br>0 (Normal)             | JOB TAGS                  | SKIP TAGS                     |
| LABELS                                | INSTANCE GROUPS           | JOB SLICING<br>1              |
| TIMEOUT<br>0                          | SHOW CHANGES<br>OFF       | OPTIONS                       |
| EXTRA VARIABLES<br>YAML JSON<br>1 --- |                           |                               |

LAUNCH    CANCEL    SAVE



# EXPANDING JOB TEMPLATES

Job Templates can be found and created by clicking the **Templates** button under the *Resources* section on the left menu.



The screenshot shows a user interface for managing job templates. A red box highlights the central content area. At the top, there's a search bar with a dropdown for 'Name', a 'Search' button, and an 'Add' button. To the right of the search bar are two circular icons. Below the search bar is a table titled 'Templates'. The table has columns for 'Name', 'Type', 'Last Ran', and 'Actions'. The data in the table is as follows:

| Name                                         | Type                  | Last Ran               | Actions               |
|----------------------------------------------|-----------------------|------------------------|-----------------------|
| Create index.html                            | Job Template          | 8/16/2021, 11:37:51 AM | Edit, Delete, Preview |
| Deploy Webapp Server                         | Workflow Job Template | 8/16/2021, 11:47:51 AM | Edit, Delete, Preview |
| INFRASTRUCTURE / Turn off IBM Community Grid | Job Template          |                        | Edit, Delete, Preview |
| Install Apache                               | Job Template          | 8/16/2021, 11:03:50 AM | Edit, Delete, Preview |
| Node.js Deploy                               | Job Template          | 8/16/2021, 11:47:51 AM | Edit, Delete, Preview |
| Web App Deploy                               | Job Template          | 8/16/2021, 11:47:33 AM | Edit, Delete, Preview |

At the bottom of the table, it says '1 - 6 of 6 items' and '1 of 1 page'.

# EXECUTING AN EXISTING JOB

Job Templates can be launched by clicking the **rocketship button** for the corresponding Job Template

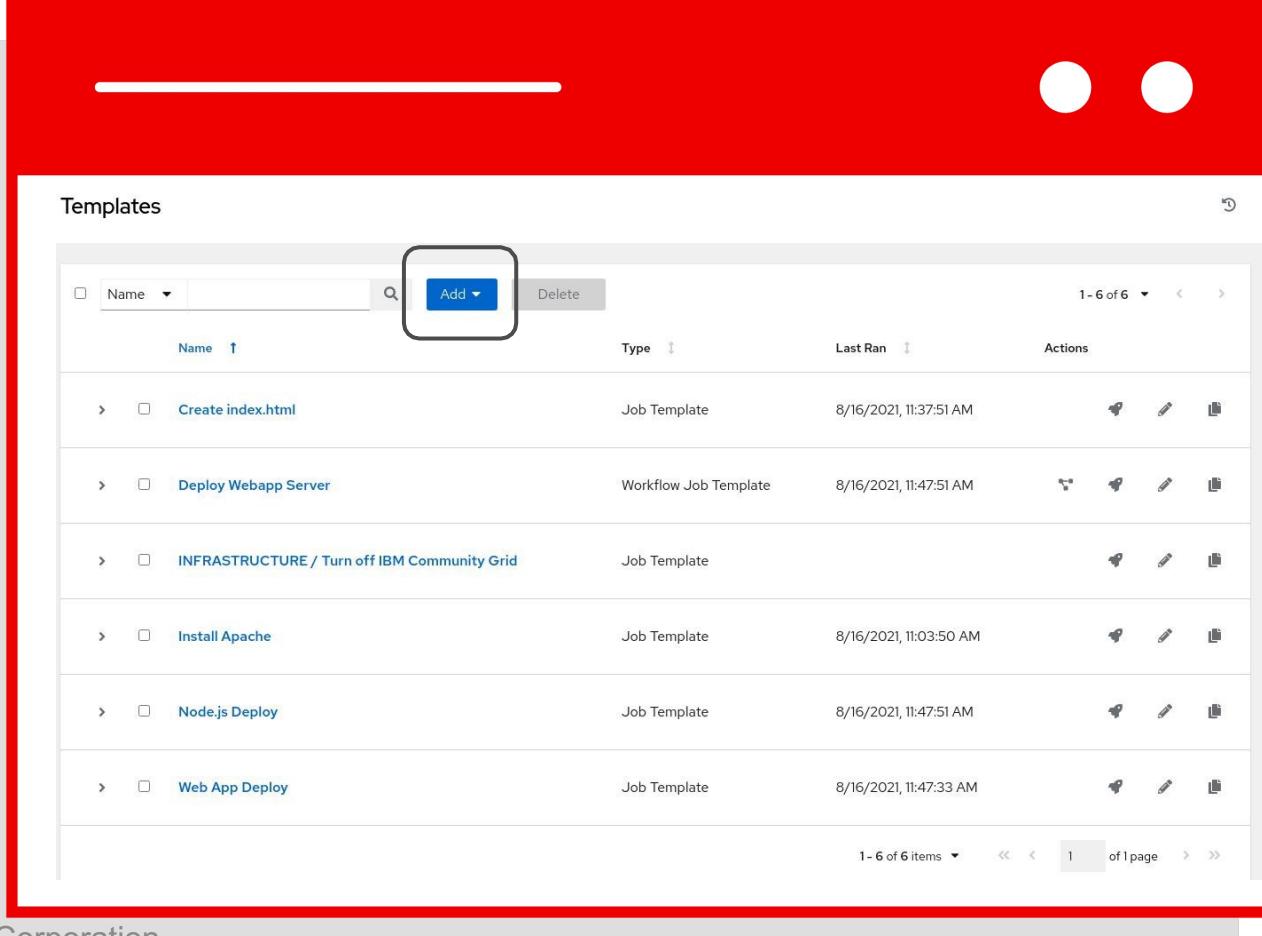


The screenshot shows a software interface titled "Templates". At the top, there is a search bar, an "Add" button, and a "Delete" button. Below the search bar, there is a table header with columns: "Name", "Type", "Last Ran", and "Actions". The table contains six rows of data. The first five rows are highlighted with a red box around their "Actions" column. The first row's "Actions" column is also highlighted with a red circle. The data in the table is as follows:

| Name                                         | Type                  | Last Ran               | Actions   |
|----------------------------------------------|-----------------------|------------------------|-----------|
| Create index.html                            | Job Template          | 8/16/2021, 11:37:51 AM | [Actions] |
| Deploy Webapp Server                         | Workflow Job Template | 8/16/2021, 11:47:51 AM | [Actions] |
| INFRASTRUCTURE / Turn off IBM Community Grid | Job Template          |                        | [Actions] |
| Install Apache                               | Job Template          | 8/16/2021, 11:03:50 AM | [Actions] |
| Node.js Deploy                               | Job Template          | 8/16/2021, 11:47:51 AM | [Actions] |
| Web App Deploy                               | Job Template          | 8/16/2021, 11:47:33 AM | [Actions] |

# CREATING A NEW JOB TEMPLATE (1/2)

New Job Templates can be created by clicking the **Add button**



The screenshot shows a software application window titled "Templates". At the top, there is a search bar, a "Delete" button, and a "1-6 of 6" status indicator. Below the header is a table with columns: "Name", "Type", "Last Ran", and "Actions". The table contains six rows, each representing a job template. The first row is "Create index.html" (Job Template, last ran 8/16/2021, 11:37:51 AM). The second row is "Deploy Webapp Server" (Workflow Job Template, last ran 8/16/2021, 11:47:51 AM). The third row is "INFRASTRUCTURE / Turn off IBM Community Grid" (Job Template, last ran 8/16/2021, 11:47:51 AM). The fourth row is "Install Apache" (Job Template, last ran 8/16/2021, 11:03:50 AM). The fifth row is "Node.js Deploy" (Job Template, last ran 8/16/2021, 11:47:51 AM). The sixth row is "Web App Deploy" (Job Template, last ran 8/16/2021, 11:47:33 AM). Each row has a "Details" icon in the "Actions" column. A large red box highlights the "Add" button at the top of the table area.

| Name                                         | Type                  | Last Ran               | Actions |
|----------------------------------------------|-----------------------|------------------------|---------|
| Create index.html                            | Job Template          | 8/16/2021, 11:37:51 AM |         |
| Deploy Webapp Server                         | Workflow Job Template | 8/16/2021, 11:47:51 AM |         |
| INFRASTRUCTURE / Turn off IBM Community Grid | Job Template          | 8/16/2021, 11:47:51 AM |         |
| Install Apache                               | Job Template          | 8/16/2021, 11:03:50 AM |         |
| Node.js Deploy                               | Job Template          | 8/16/2021, 11:47:51 AM |         |
| Web App Deploy                               | Job Template          | 8/16/2021, 11:47:33 AM |         |

## CREATING A NEW JOB TEMPLATE (2/2)

This **New Job Template** window is where the inventory, project and credential are assigned. The red asterisk \* means the field is required .

The screenshot shows the 'Create New Job Template' dialog box. The form includes fields for Name, Description, Job Type (set to Run), Inventory, Project, Execution Environment, Playbook (with a dropdown menu showing 'Select a playbook'), Credentials, Labels, and Variables (set to YAML). The 'Inventory', 'Project', and 'Playbook' fields are marked with a red asterisk, indicating they are required.

Templates

Create New Job Template

Name \*

Description

Job Type \* ⓘ

Run

Inventory \* ⓘ

Project \* ⓘ

Execution Environment ⓘ

Playbook \* ⓘ

Select a playbook

Credentials ⓘ

Labels ⓘ

Variables ⓘ

YAML JSON

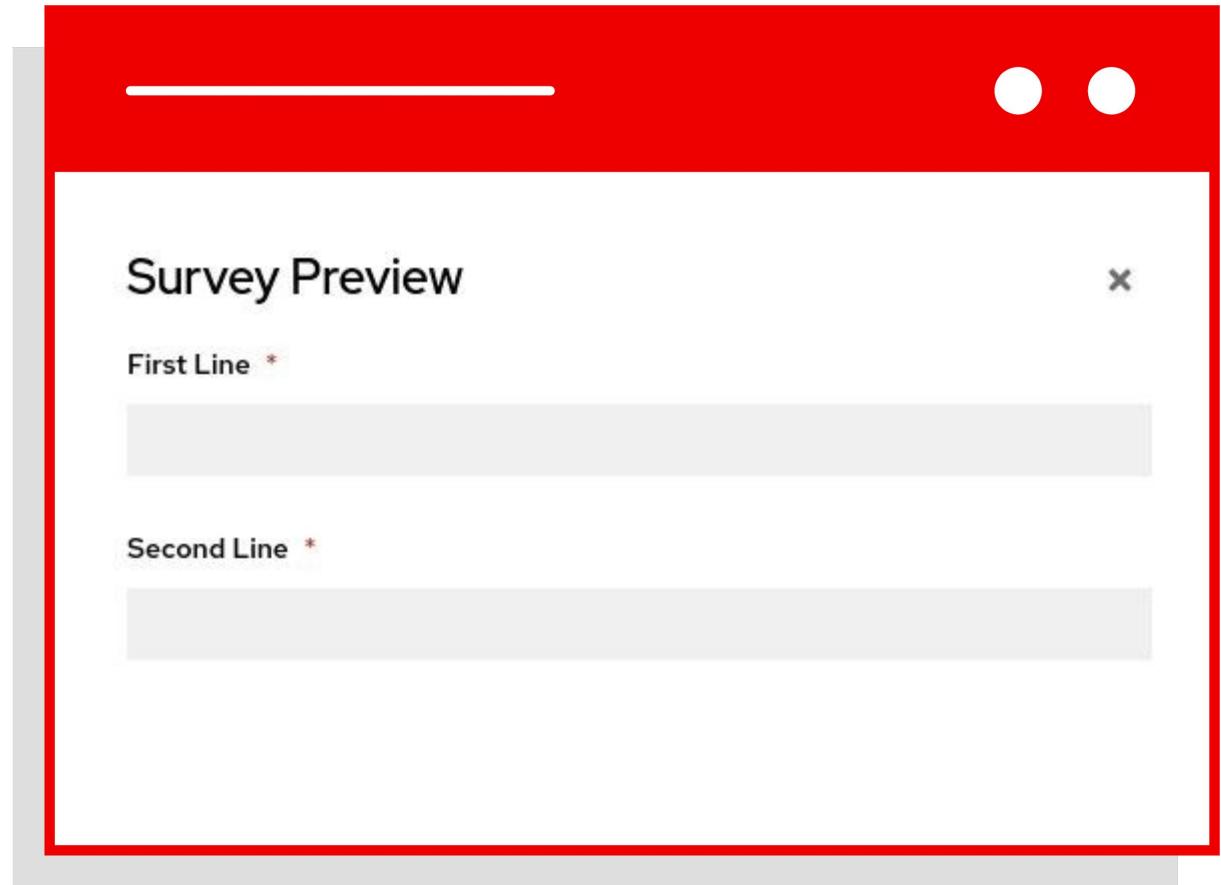
1 ---  
2

# SURVEYS

Controller surveys allow you to configure how a job runs via a series of questions, making it simple to customize your jobs in a user-friendly way.

An Ansible Controller survey is a simple question-and-answer form that allows users to customize their job runs.

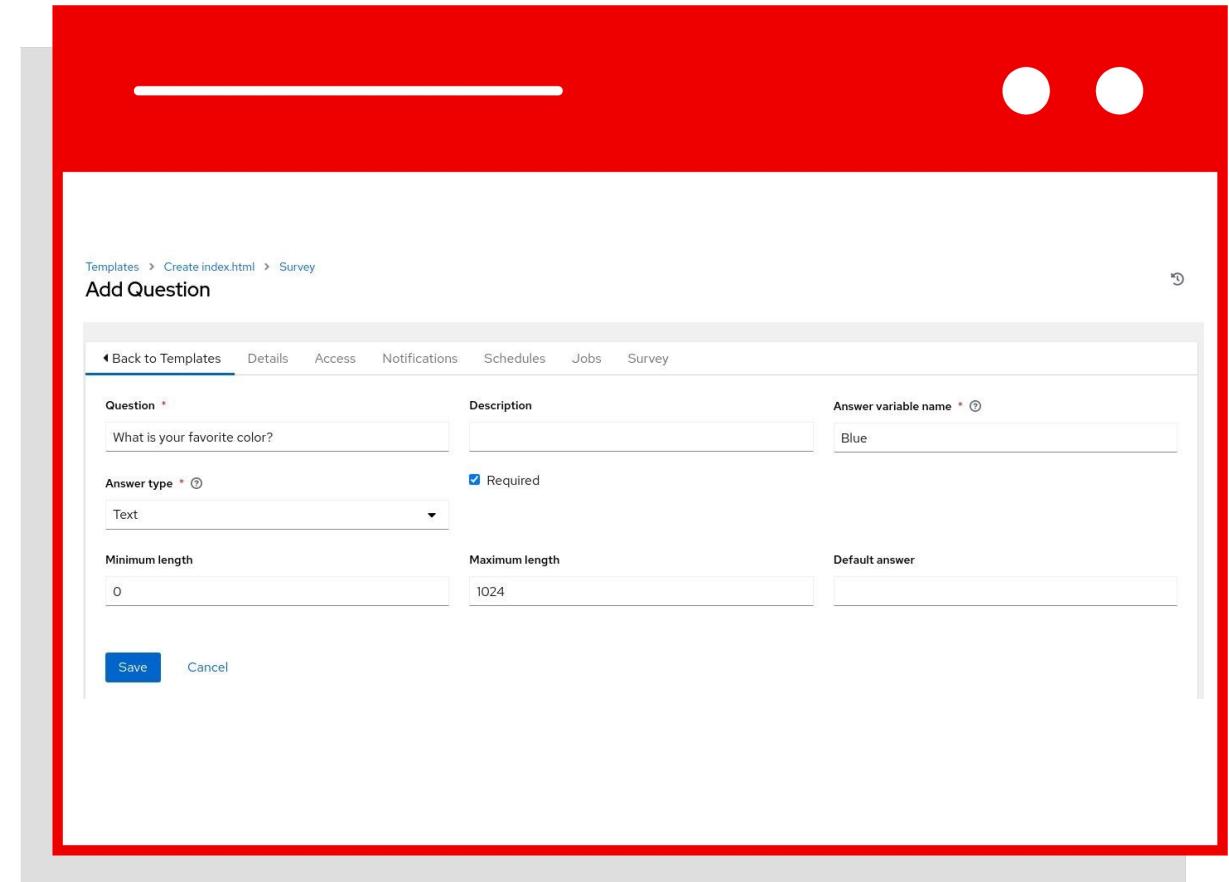
Combine that with Controller's role-based access control, and you can build simple, easy self-service for your users.



# CREATING A SURVEY (1/2)

Once a Job Template is saved, the Survey menu will have an **Add Button**

Click the button to open the Add Survey window.



## CREATING A SURVEY (2/2)

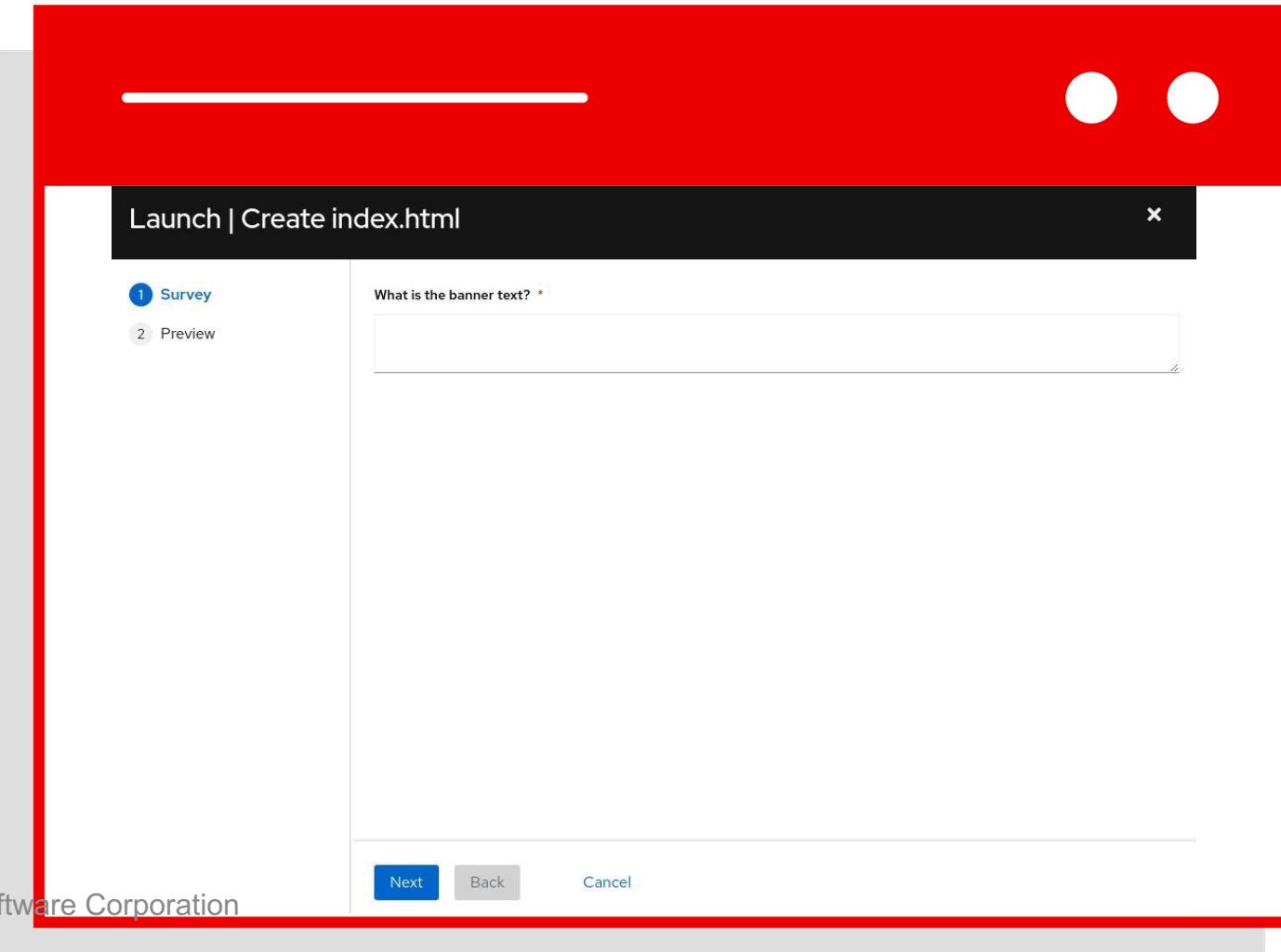
The Add Survey window allows the Job Template to prompt users for one or more questions. The answers provided become variables for use in the Ansible Playbook.

This screenshot shows the 'Add Question' form within a 'Survey' tab of a job template configuration. The form includes fields for 'Question' (What is the banner text?), 'Description' (empty), 'Answer variable name' (net\_banner), 'Answer type' (Textarea, selected), 'Required' (checked), 'Minimum length' (0), 'Maximum length' (1024), and 'Default answer' (empty). Buttons at the bottom include 'Save' and 'Cancel'.

This screenshot shows the 'Survey' configuration interface for the job template. It lists the question 'What is the banner text?' with its details: Type (textarea), Default value (empty), and a preview section showing the question text.

# USING A SURVEY

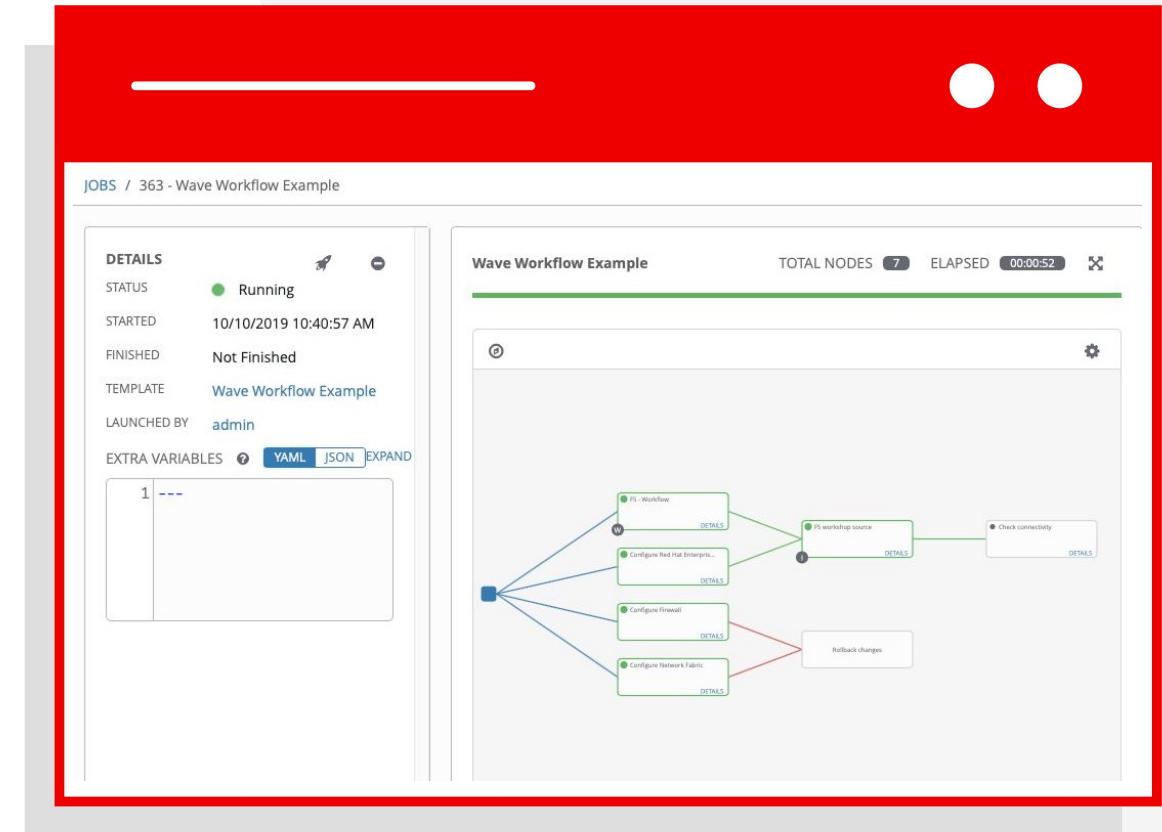
When launching a job, the user will now be prompted with the Survey. The user can be required to fill out the Survey before the Job Template will execute.



# WORKFLOWS

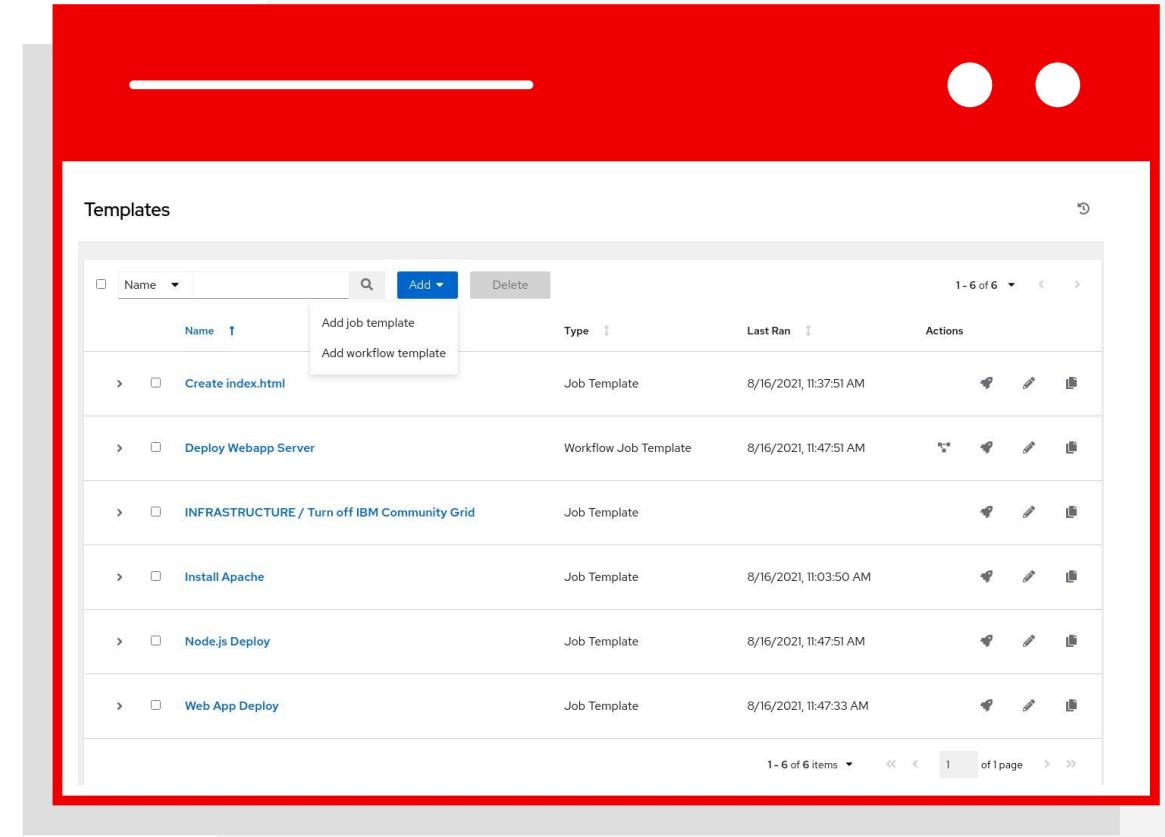
Combine automation to create something bigger

- ▶ Workflows enable the creation of powerful holistic automation, chaining together multiple pieces of automation and events.
- ▶ Simple logic inside these workflows can trigger automation depending on the success or failure of previous steps.



# ADDING A NEW TEMPLATE

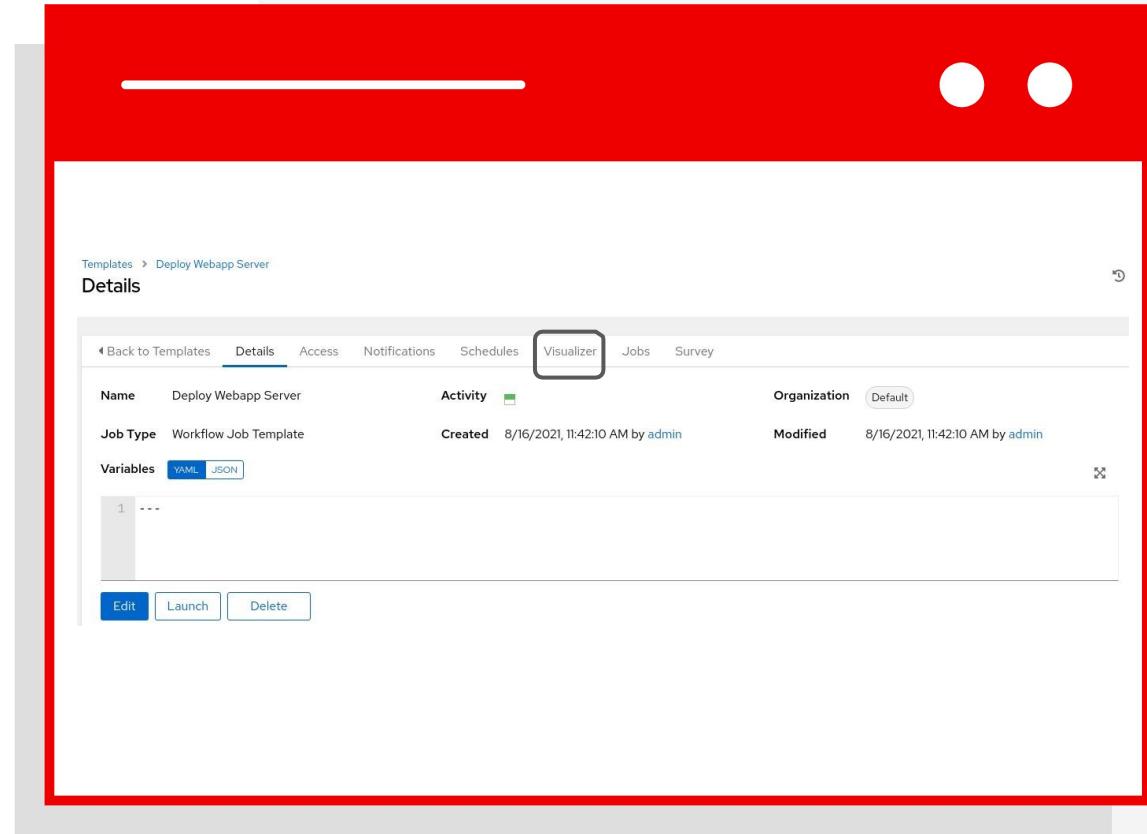
- To add a new **Workflow** click on the **Add** button.  
This time select the **Add workflow template**



# CREATING THE WORKFLOW

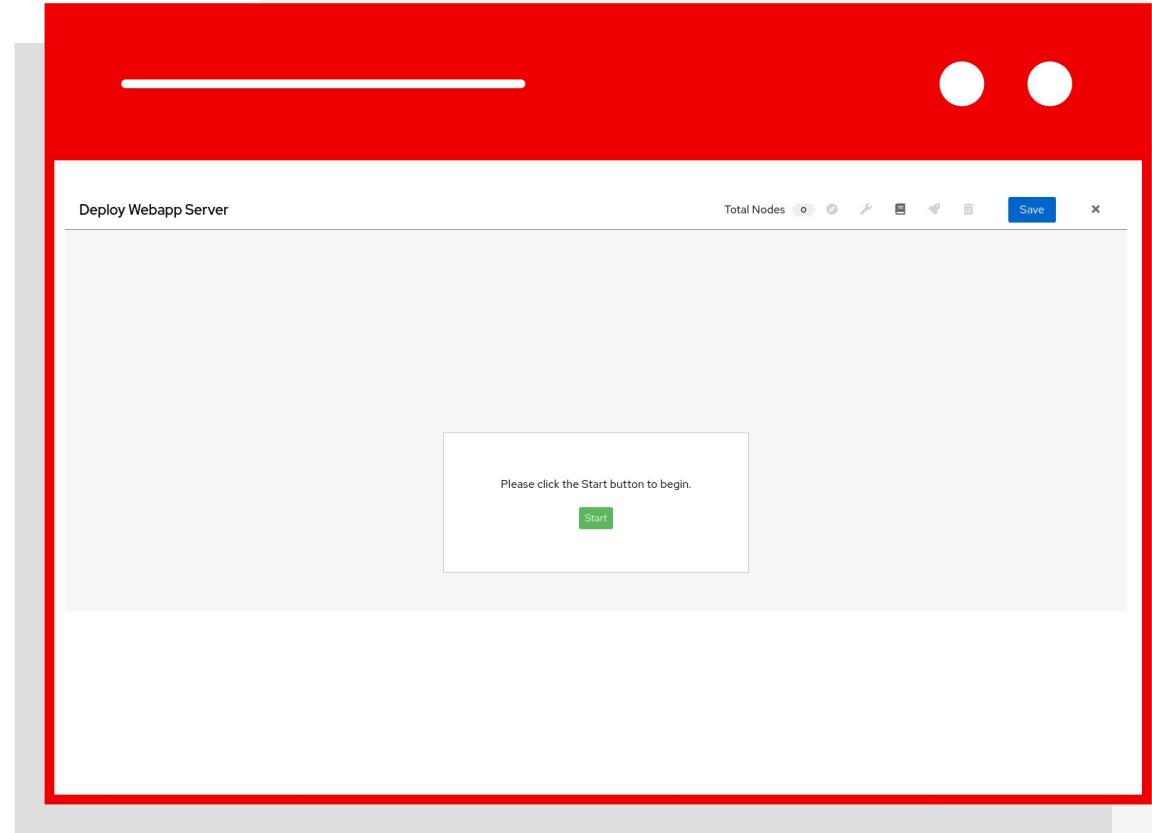
- Fill out the required parameters and click **Save**.

As soon as the Workflow Template is saved the Workflow Visualizer will open.



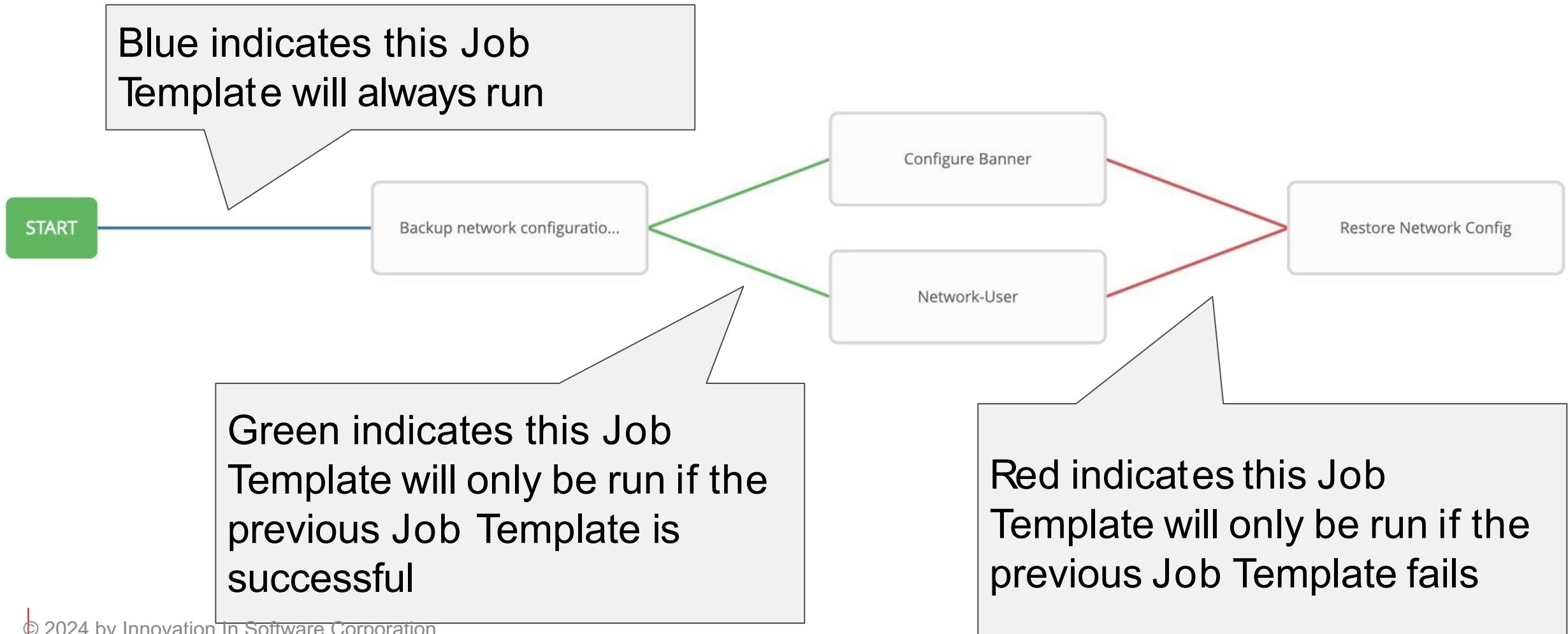
# WORKFLOW VISUALIZER

- ▶ The Workflow Visualizer will start as a blank canvas.
- ▶ Click the green Start button to start building the workflow.



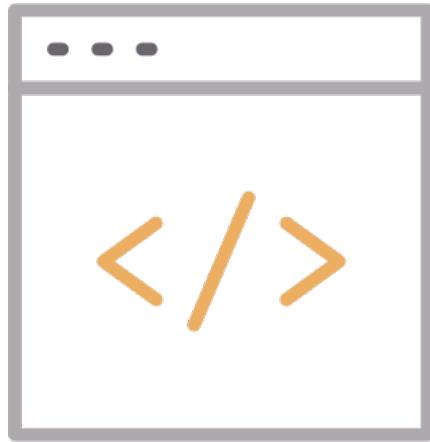
# VISUALIZING A WORKFLOW

Workflows can branch out, or converge in.



# Lab: AAP Projects and job templates

# Developing a module

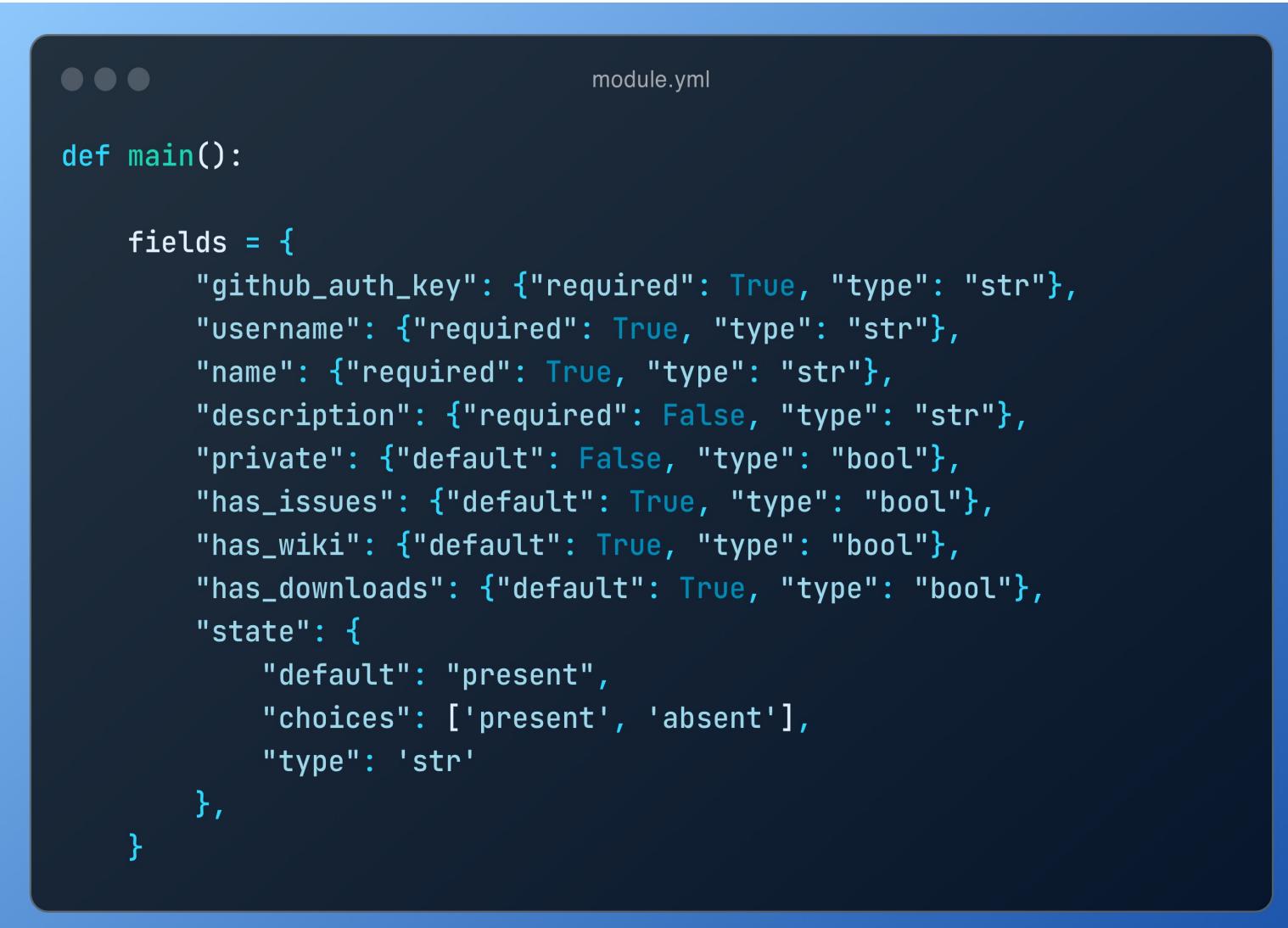


A module is a reusable, standalone script that Ansible runs on your behalf, either locally or remotely. Modules interact with your local machine, an API, or a remote system to perform specific tasks like changing a database password or spinning up a cloud instance.

```
- name: Install ntp
 package:
 name:
 - ntp
 - nslookup
 state: present
 tags: utils
```

# Developing a module

A module provides a defined interface, accepts arguments, and returns information to Ansible by printing a JSON string to stdout before exiting.



The image shows a terminal window with a dark background and light-colored text. The title bar of the window says "module.yml". The code inside the window is a Python function named "main" which defines a dictionary "fields" containing various parameters with their types and default values. The code uses standard Python syntax with indentation and curly braces for dictionaries.

```
def main():

 fields = {
 "github_auth_key": {"required": True, "type": "str"},
 "username": {"required": True, "type": "str"},
 "name": {"required": True, "type": "str"},
 "description": {"required": False, "type": "str"},
 "private": {"default": False, "type": "bool"},
 "has_issues": {"default": True, "type": "bool"},
 "has_wiki": {"default": True, "type": "bool"},
 "has_downloads": {"default": True, "type": "bool"},
 "state": {
 "default": "present",
 "choices": ['present', 'absent'],
 "type": 'str'
 },
 }
```

# Developing a module



To create a module:

1. Create a library directory in your workspace. Your test play should live in the same directory.
2. Create your new module file:
  - `library/my_test.py`.
3. Start writing your module.

# Developing a module

All modules require documentation.  
Python docstrings should include  
examples of how to use the module.

```
module.yml

#!/usr/bin/python3

from __future__ import (absolute_import, division,
print_function)
__metaclass__ = type

DOCUMENTATION = r'''

module: github_repo

short_description: This module manages GitHub repositories
...

EXAMPLES = r'''
- name: Create a github Repo
 github_repo:
 github_auth_key: "..."
 name: "Hello-World"
 description: "This is your first repository"
 private: yes
 has_issues: no
 has_wiki: no
 has_downloads: no
 register: result

- name: Delete that repo
 github_repo:
 github_auth_key: "..."
 name: "Hello-World"
 state: absent
 register: result
'''
```

# Developing a module

This code defines a function that creates a new GitHub repository using the GitHub API. It extracts the GitHub authentication key from the provided data, sends a POST request to the /user/repos endpoint with the repository details.

The function returns different outcomes based on the response: success with changes (201), no changes due to an existing repository (422), or a default response indicating an error with the status code and response metadata for debugging.

```
module.yml

from ansible.module_utils.basic import *
import requests

api_url = "https://api.github.com"

def github_repo_present(data):

 api_key = data['github_auth_key']

 del data['state']
 del data['github_auth_key']

 headers = {
 "Authorization": "token {}".format(api_key)
 }
 url = "{}{}".format(api_url, '/user/repos')
 result = requests.post(url, json.dumps(data), headers=headers)

 if result.status_code == 201:
 return False, True, result.json()
 if result.status_code == 422:
 return False, False, result.json()

 # default: something went wrong
 meta = {"status": result.status_code, 'response': result.json()}
 return True, False, meta
```

# Developing a module

This function deletes a specified GitHub repository by interacting with the GitHub API. It constructs the request URL using the repository's name and the user's username, authenticates using the provided GitHub token, and sends a DELETE request.

The function returns success with changes (204), no changes if the repository does not exist (404), or a default response indicating an error with the status code and response metadata for debugging.

```
module.yml

def github_repo_absent(data=None):
 headers = {
 "Authorization": "token {}".format(data['github_auth_key'])
 }
 url = "{}/repos/{}/{}/{}" . format(api_url, data['username'],
data['name'])
 result = requests.delete(url, headers=headers)

 if result.status_code == 204:
 return False, True, {"status": "SUCCESS"}
 if result.status_code == 404:
 result = {"status": result.status_code, "data":
result.json()}
 return False, False, result
 else:
 result = {"status": result.status_code, "data":
result.json()}
 return True, False, result
```

# Developing a module

The main function serves as the entry point for an Ansible module to manage GitHub repositories. It defines a schema for input fields using fields, including parameters like `github_auth_key`, `username`, `name`, and other repository properties.

Based on the `state` parameter, it maps to either `github_repo_present` or `github_repo_absent` to create or delete a repository. The function processes the action, and based on the outcome, it either exits successfully with the result or fails with an error message, providing the necessary feedback to Ansible.

The `if __name__ == '__main__':` condition ensures that the code inside the block is executed only when the script is run directly. If the module is imported elsewhere, this block will not run.

It allows the module to define functions or classes that can be imported without executing the script's main logic.

```
module.yml

def main():

 fields = {
 "github_auth_key": {"required": True, "type": "str"},
 "username": {"required": True, "type": "str"},
 "name": {"required": True, "type": "str"},
 "description": {"required": False, "type": "str"},
 "private": {"default": False, "type": "bool"},
 "has_issues": {"default": True, "type": "bool"},
 "has_wiki": {"default": True, "type": "bool"},
 "has_downloads": {"default": True, "type": "bool"},
 "state": {
 "default": "present",
 "choices": ['present', 'absent'],
 "type": 'str'
 },
 }

 choice_map = {
 "present": github_repo_present,
 "absent": github_repo_absent,
 }

 module = AnsibleModule(argument_spec=fields)
 is_error, has_changed, result = choice_map.get(
 module.params['state'])(module.params)

 if not is_error:
 module.exit_json(changed=has_changed, meta=result)
 else:
 module.fail_json(msg="Error deleting repo", meta=result)

if __name__ == '__main__':
 main()
```

# Lab: Write a module