Automation Developer

TEKsystems Global Services

WORKFORCE DEVELOPMENT

Lab page

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

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.

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:

Specifying 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"
```

POP QUIZ: DISCUSSION

How have you used handlers?

POP QUIZ: DISCUSSION

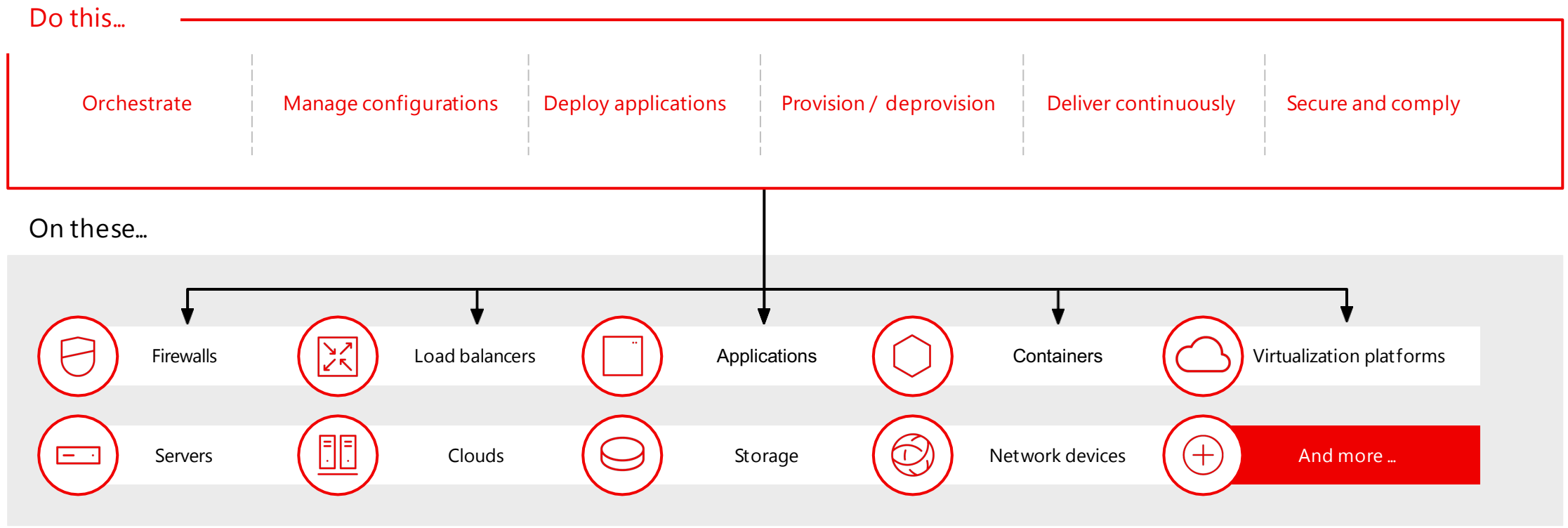
When should handlers be used?

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

Why The Ansible Automation Platform?

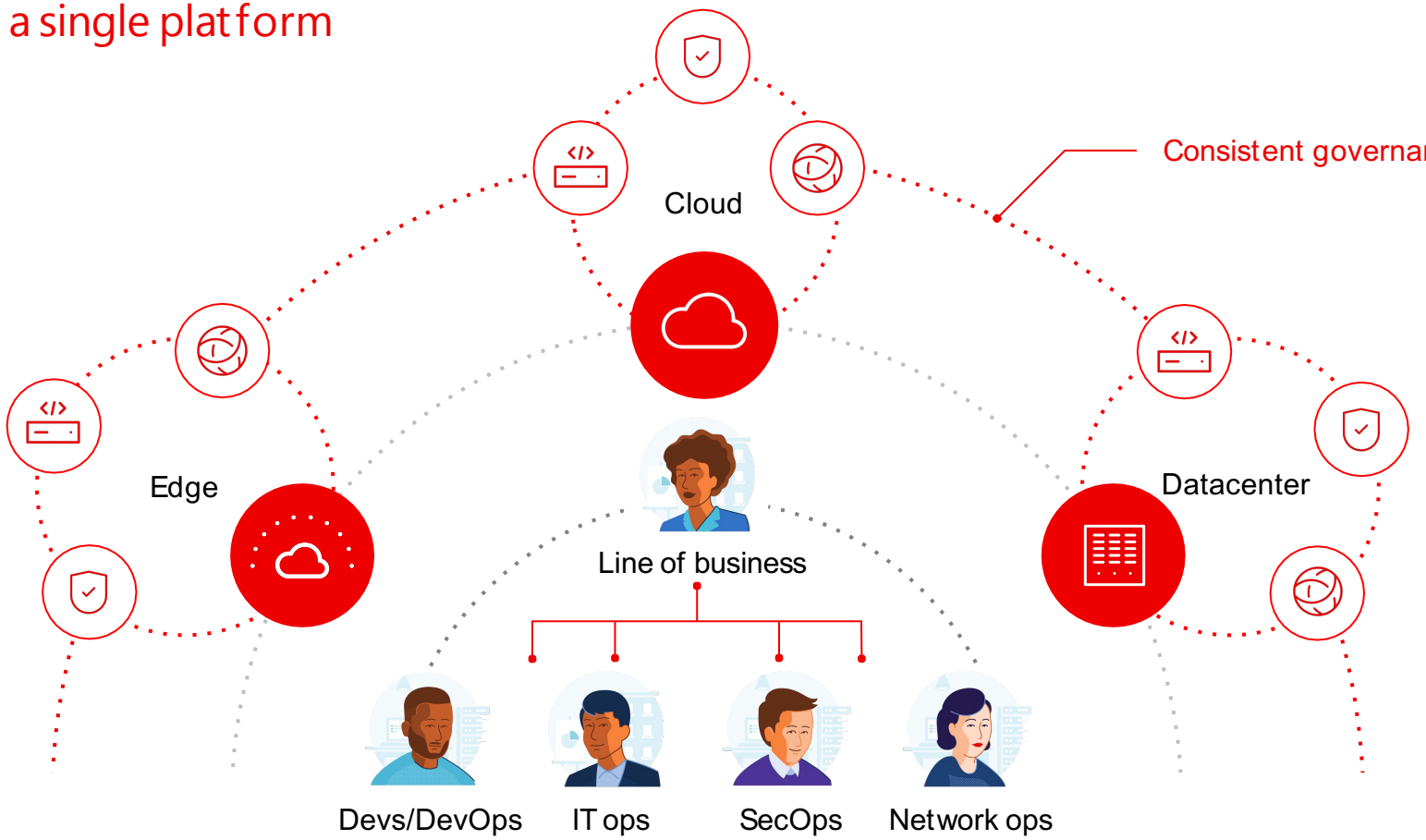
Automate the deployment and management of automation

Your entire IT footprint



Break Down Silos

Different teams a single platform



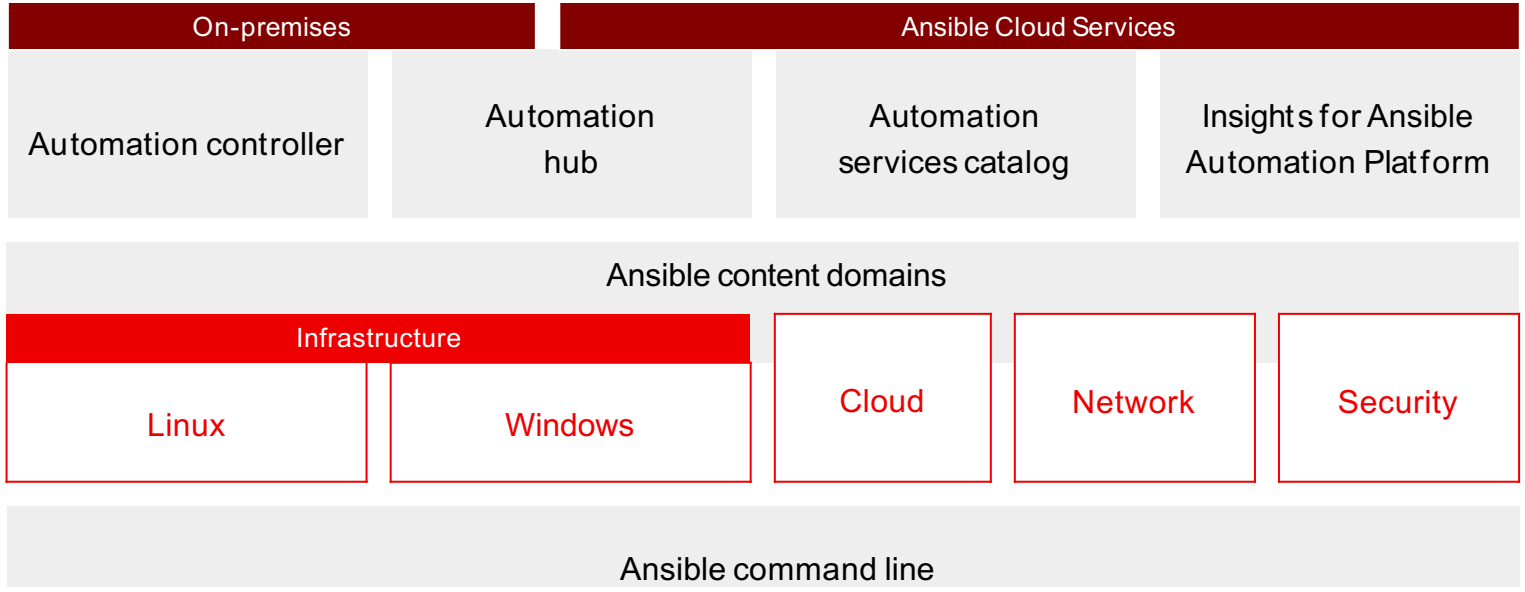
Red Hat
Ansible Automation Platform

Content creators

Operators

Domain experts

Users



Fueled by an open source community

Automation Platform Concepts

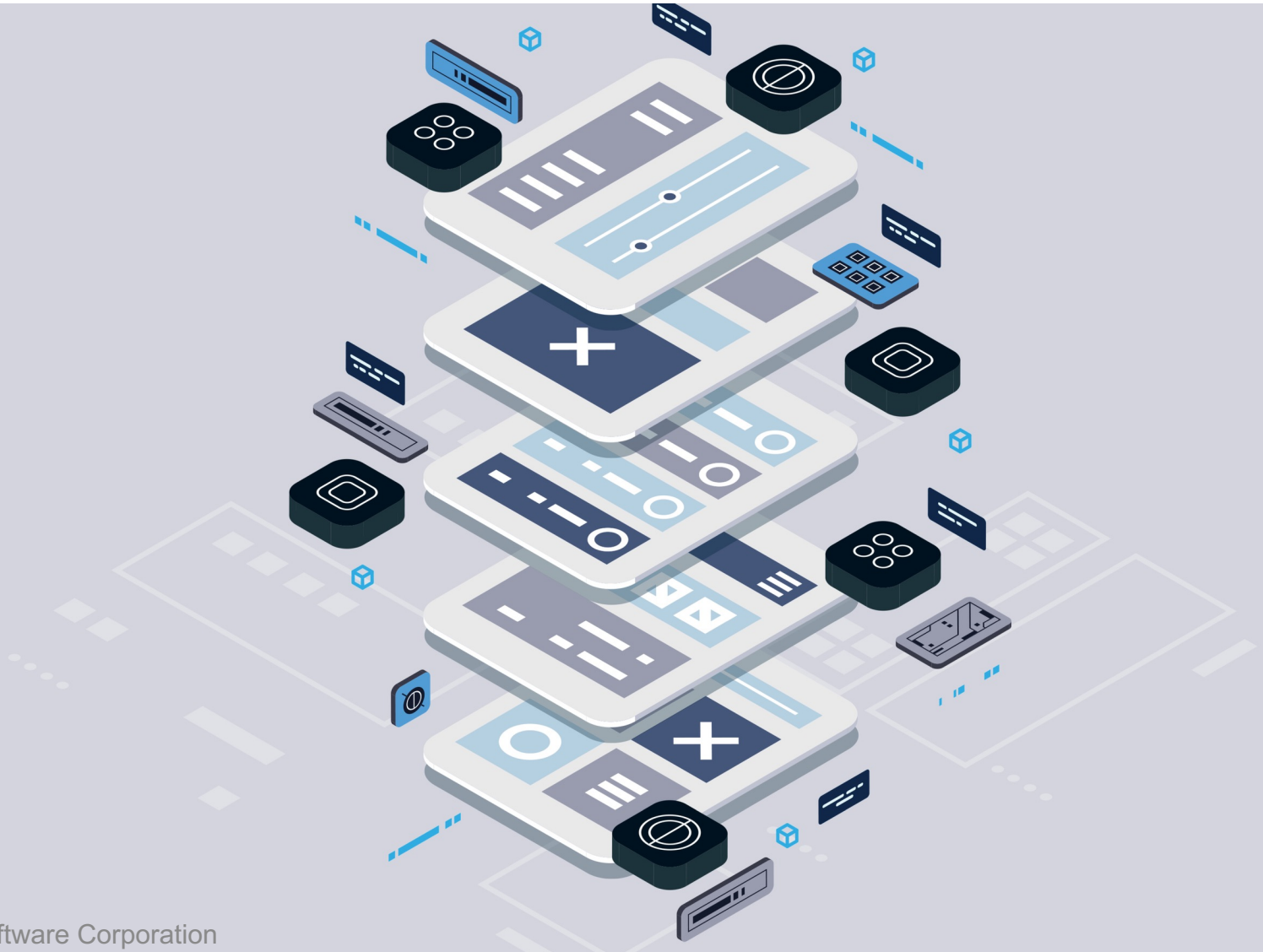
A control node is installed with Ansible and is used to run playbooks.

- Contains the Ansible Engine software, the playbook, and it's 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)

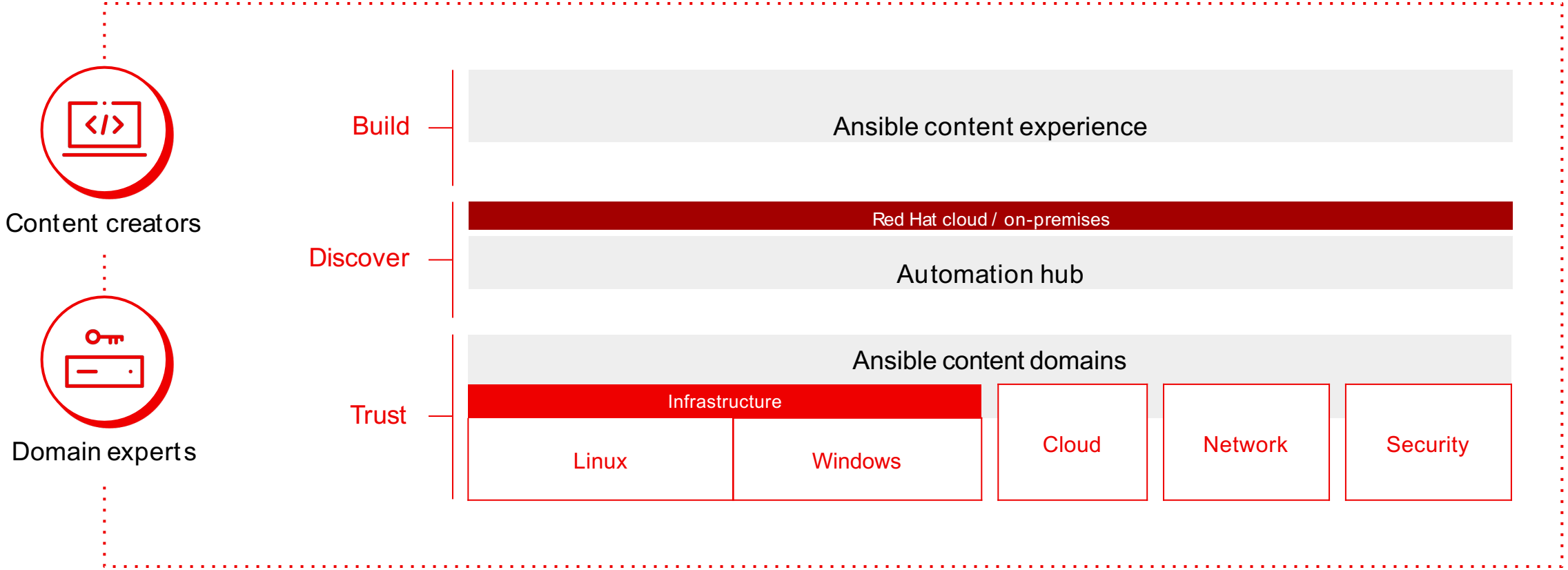
Ansible Automation Platform Infrastructure



Create Code

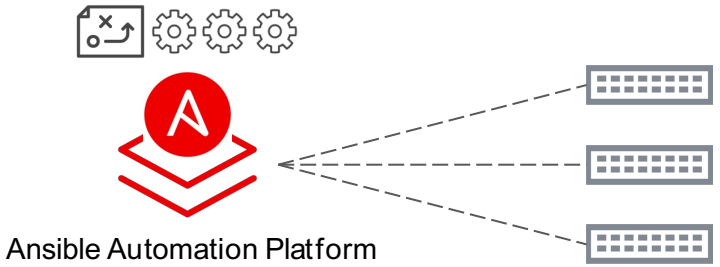
Create

The automation lifecycle



How Ansible Automation Works

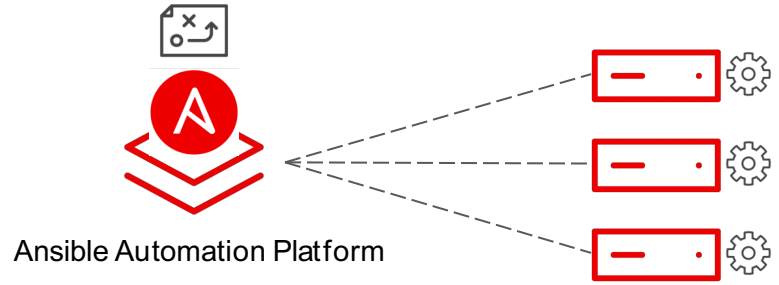
Module code is executed locally on the control node



Local Execution

Network Devices / API Endpoints

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



Remote Execution

Linux / Windows Hosts

Lab Environment

This class uses a single-node installation with an integrated database:

- Need a system installed with Red Hat Enterprise Linux (bare metal, virtual machine, or cloud instance)
  - Cloud VM
  - 2 vCPU / 4 GB RAM / at least 40GB of storage

Sign up for an evaluation of Automation Platform from Red Hat:

- https://www.redhat.com/en/technologies/management/ansible/try-it

Installer

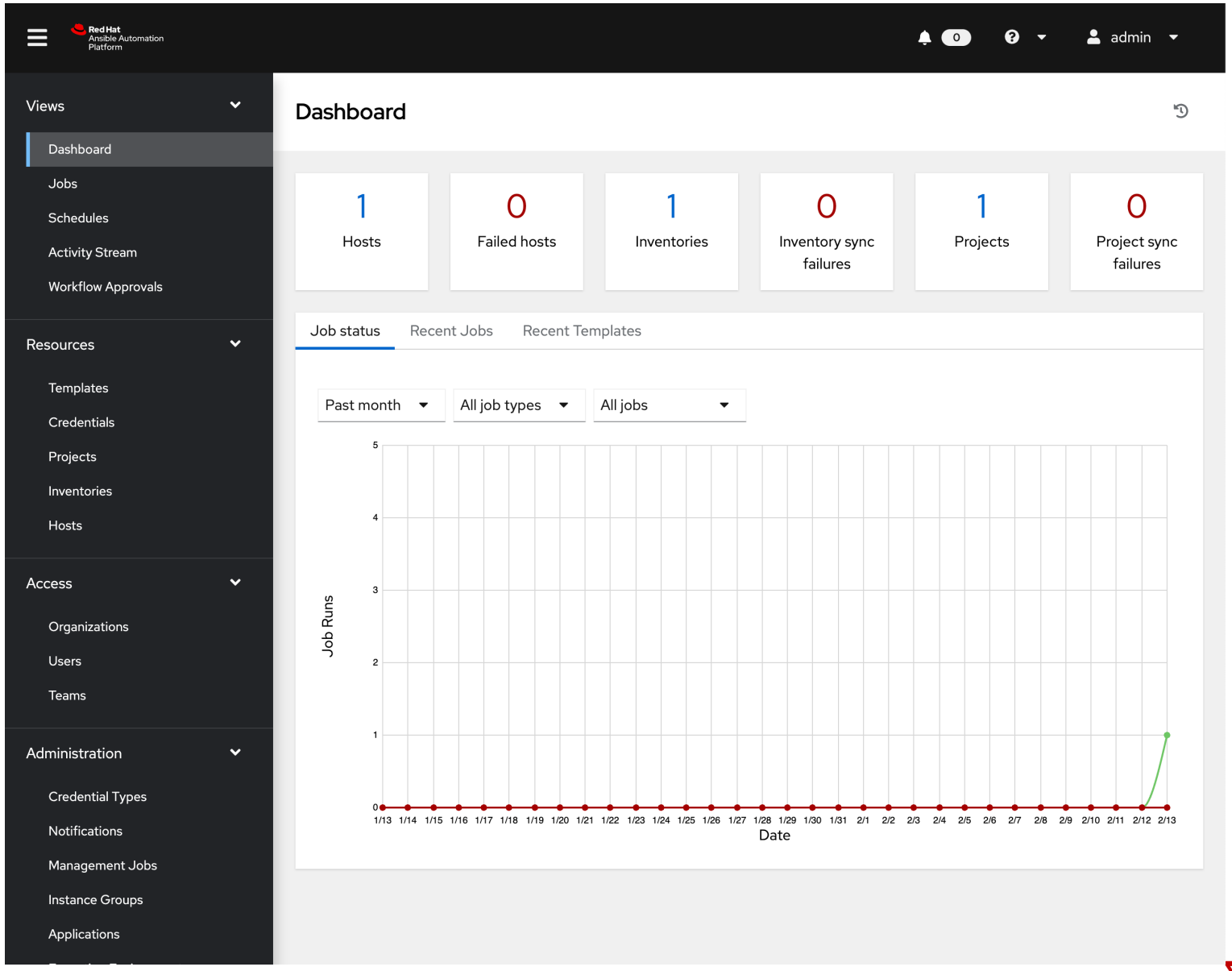
Two different installation packages are available for Automation Platform:

- Standard setup
  - https://access.redhat.com/downloads/content/480
  - Requires internet connectivity to download Automation Platform packages from repositories.

- Bundled installer
  - Download from same link as “Standard setup” but choose the packages with “Bundle” in the name.
  - Includes initial RPM packages for Automation Platform
  - May be installed on systems without internet access.

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

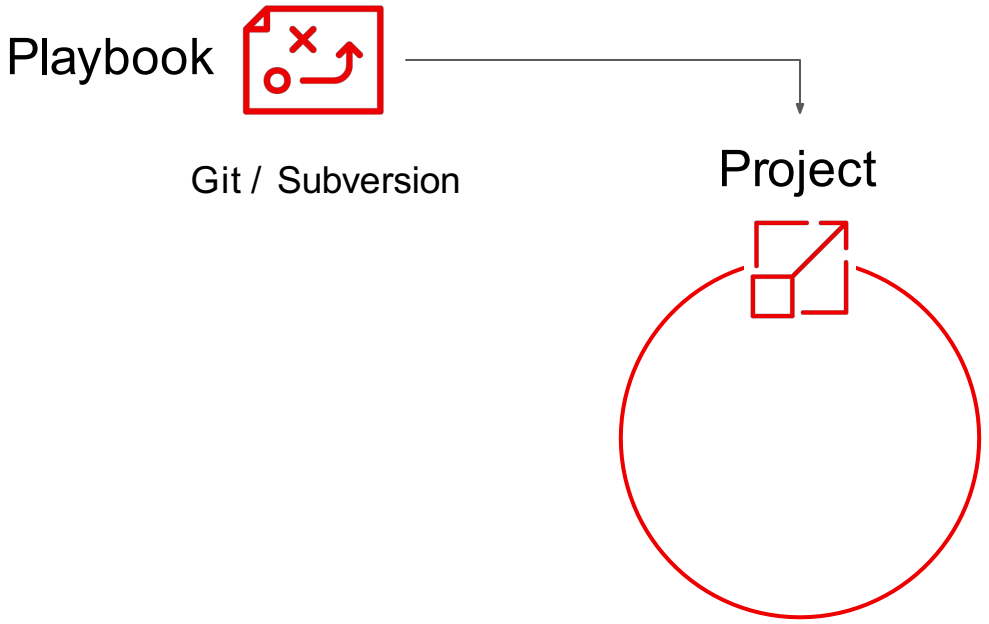
The dashboard contains links to common resources.

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.

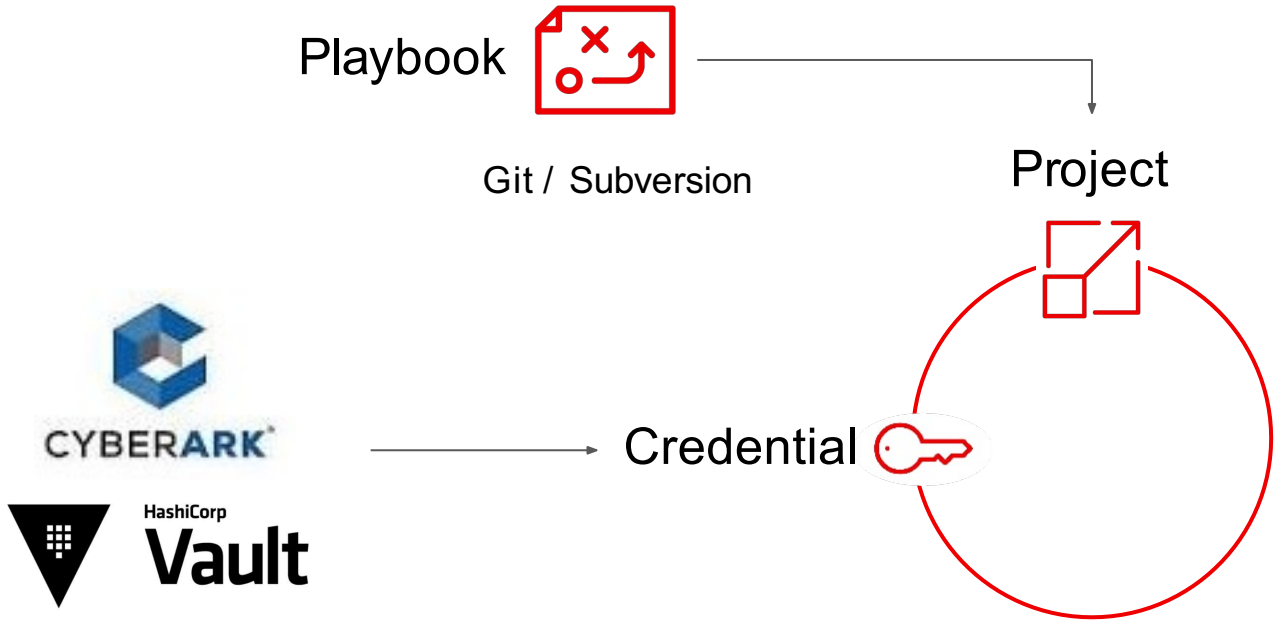
Lab: Install Ansible Automation Platform

Automation Controller

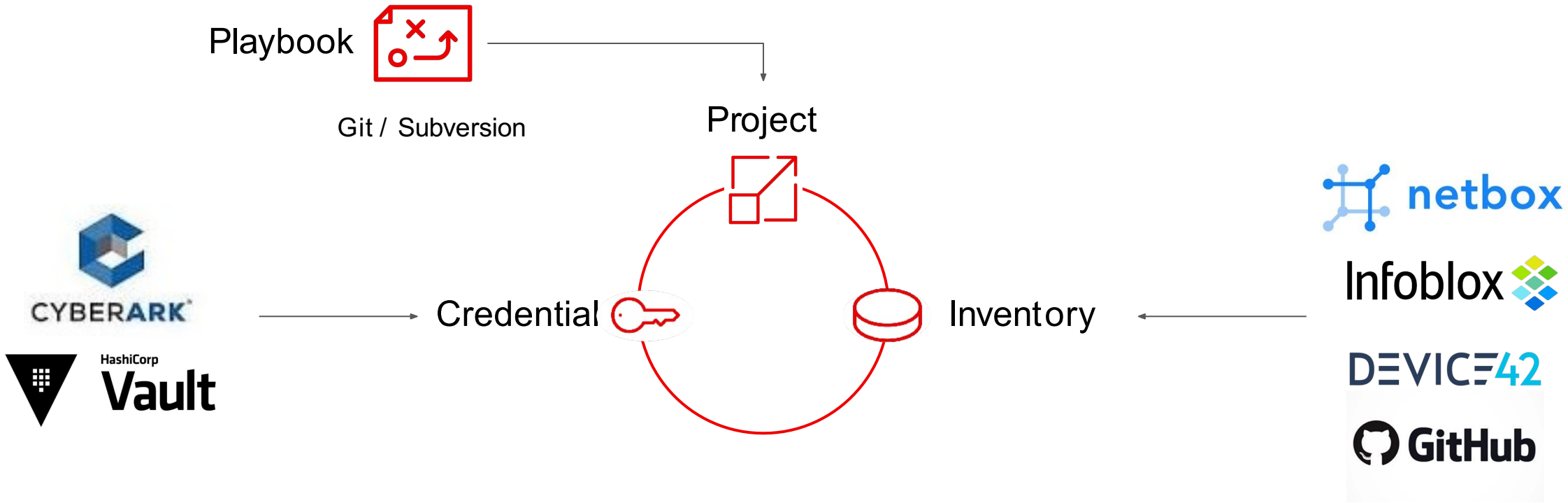
Anatomy of An Automation Job



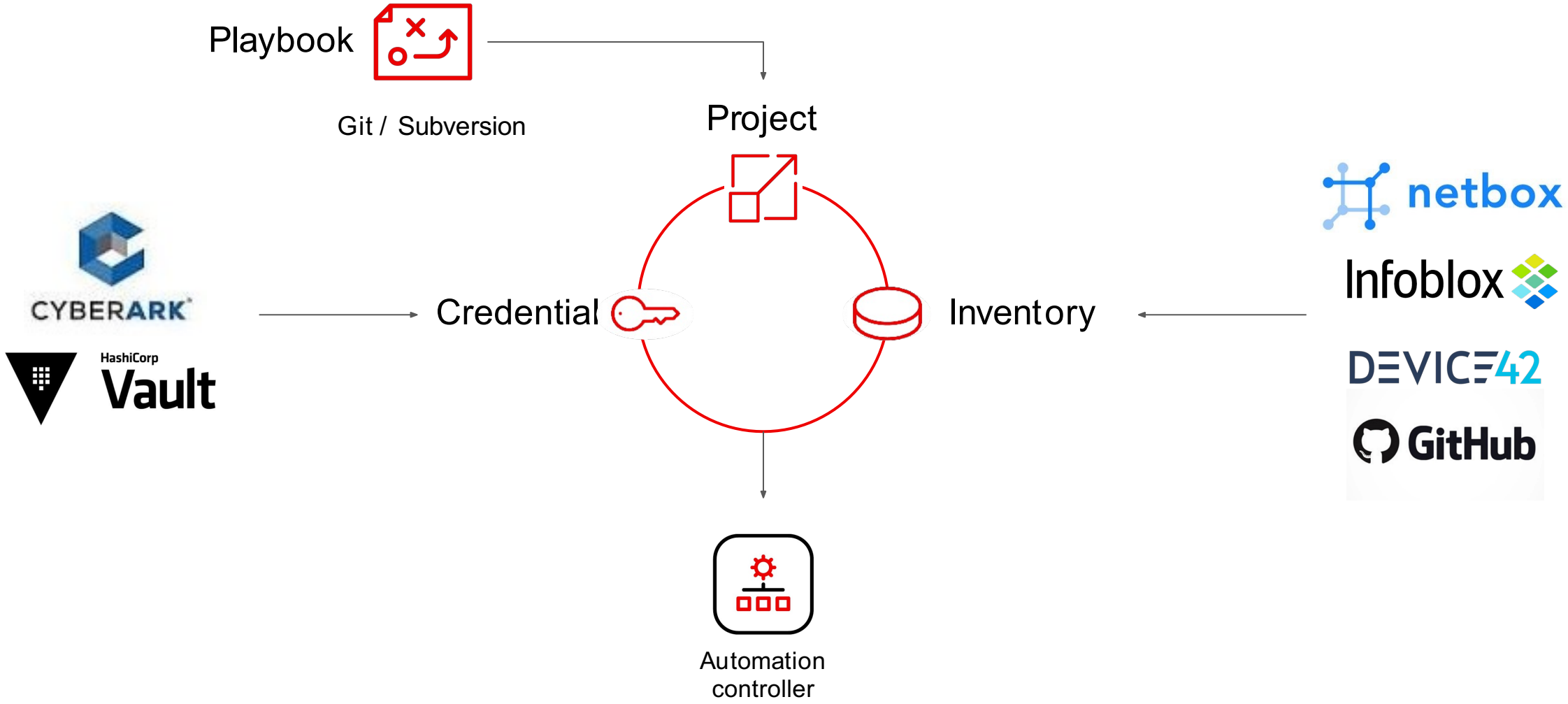
Anatomy of An Automation Job



Anatomy of An Automation Job

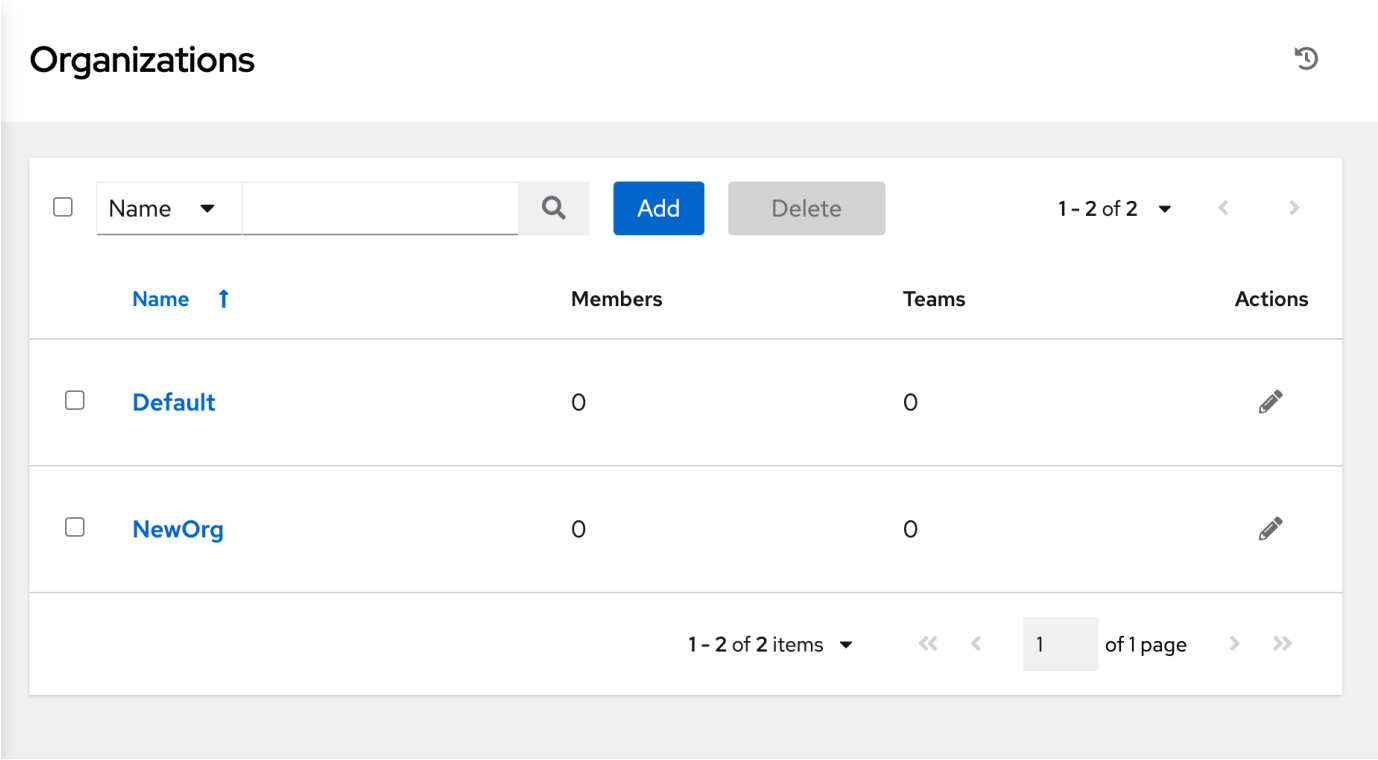


Anatomy of An Automation Job



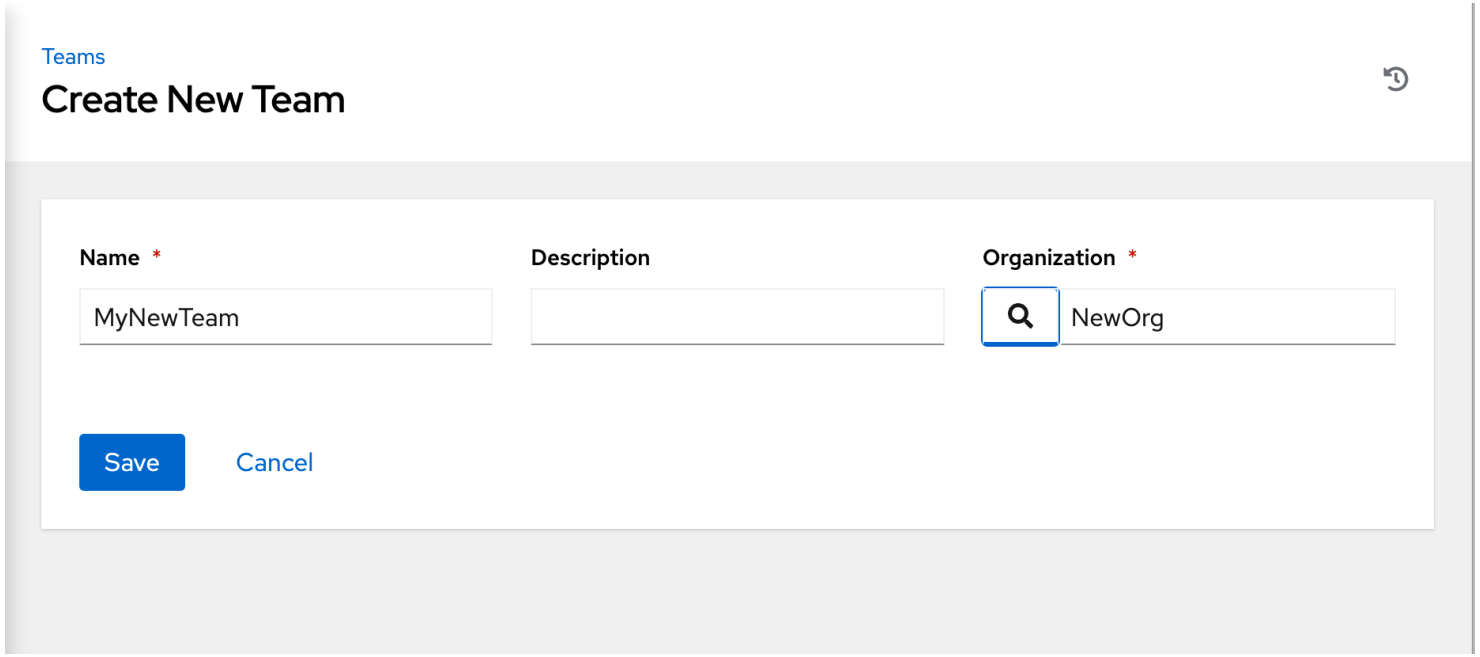
Organization

- 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.



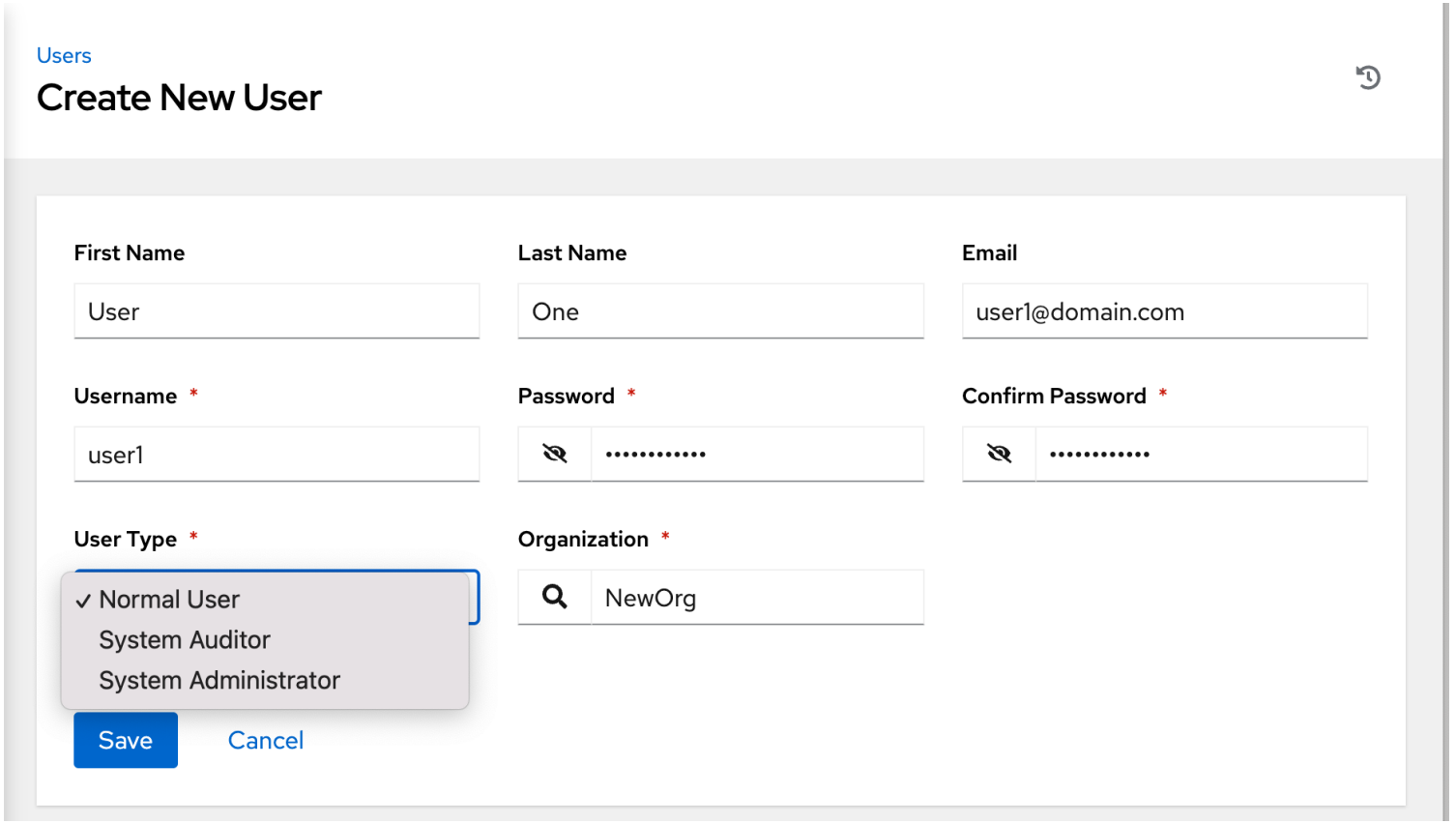
Team

- You can apply permissions at the team level.



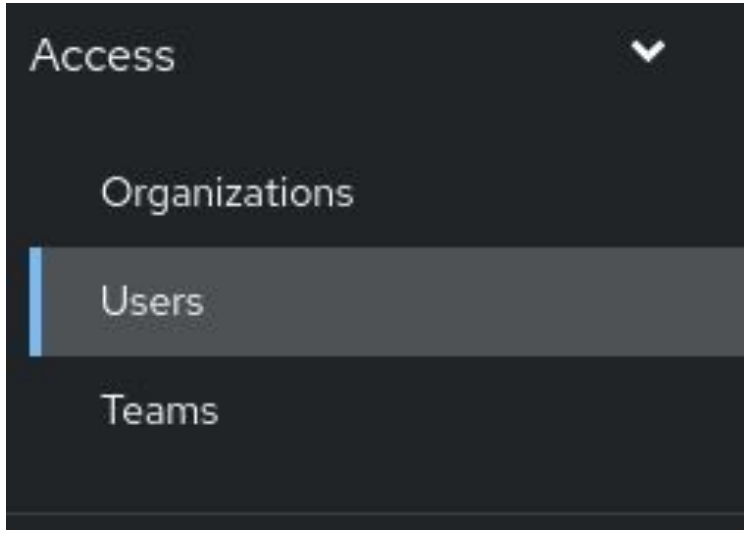
User Roles

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



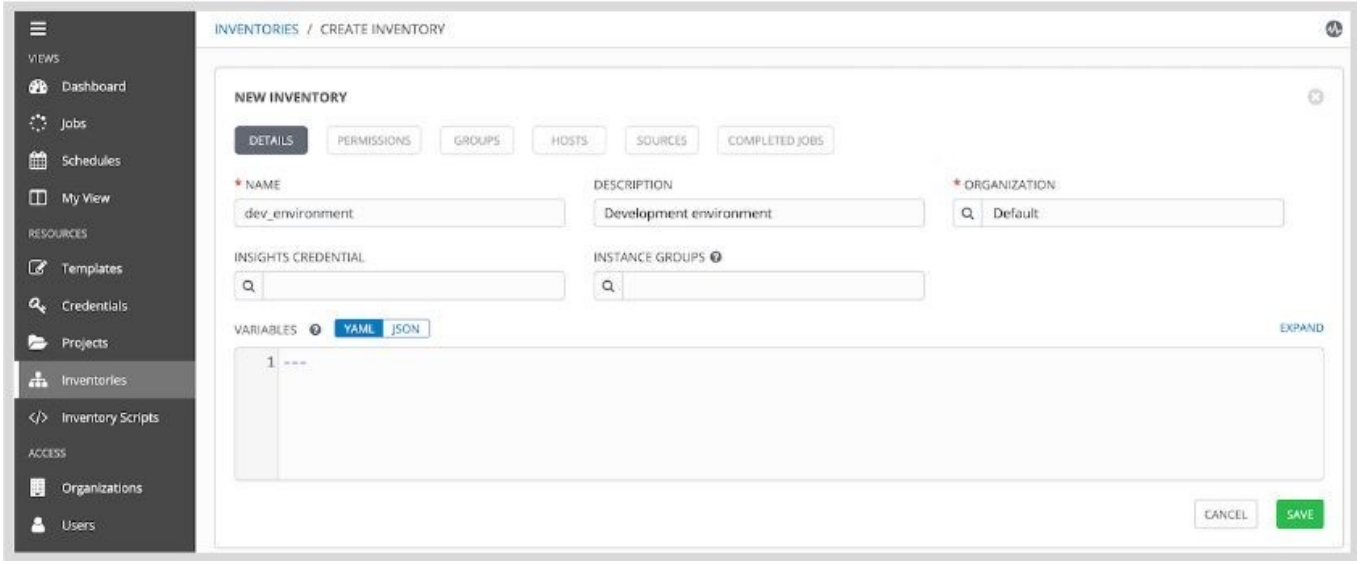
User Management

- 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.



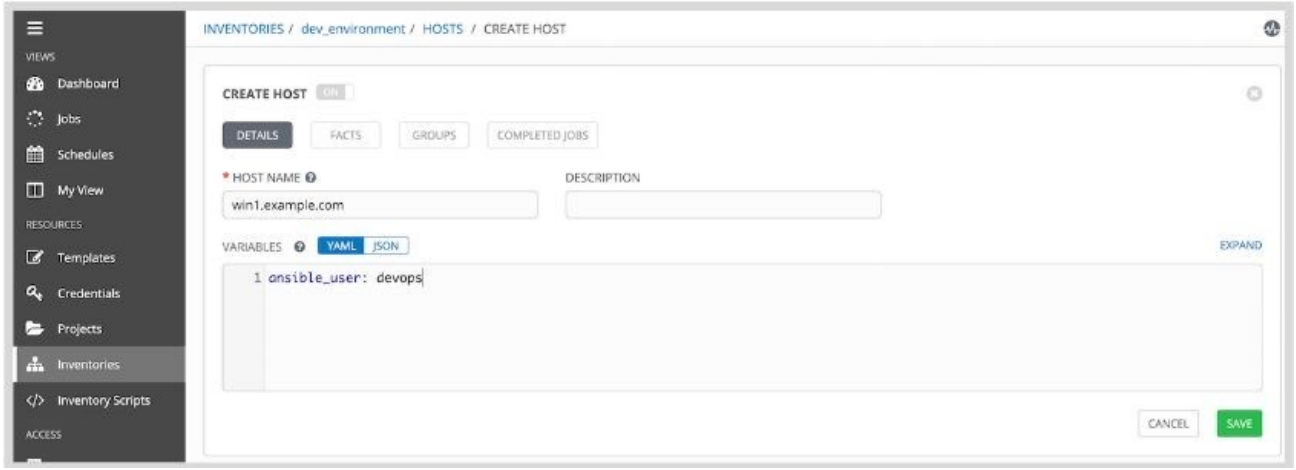
Inventory

- 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**.

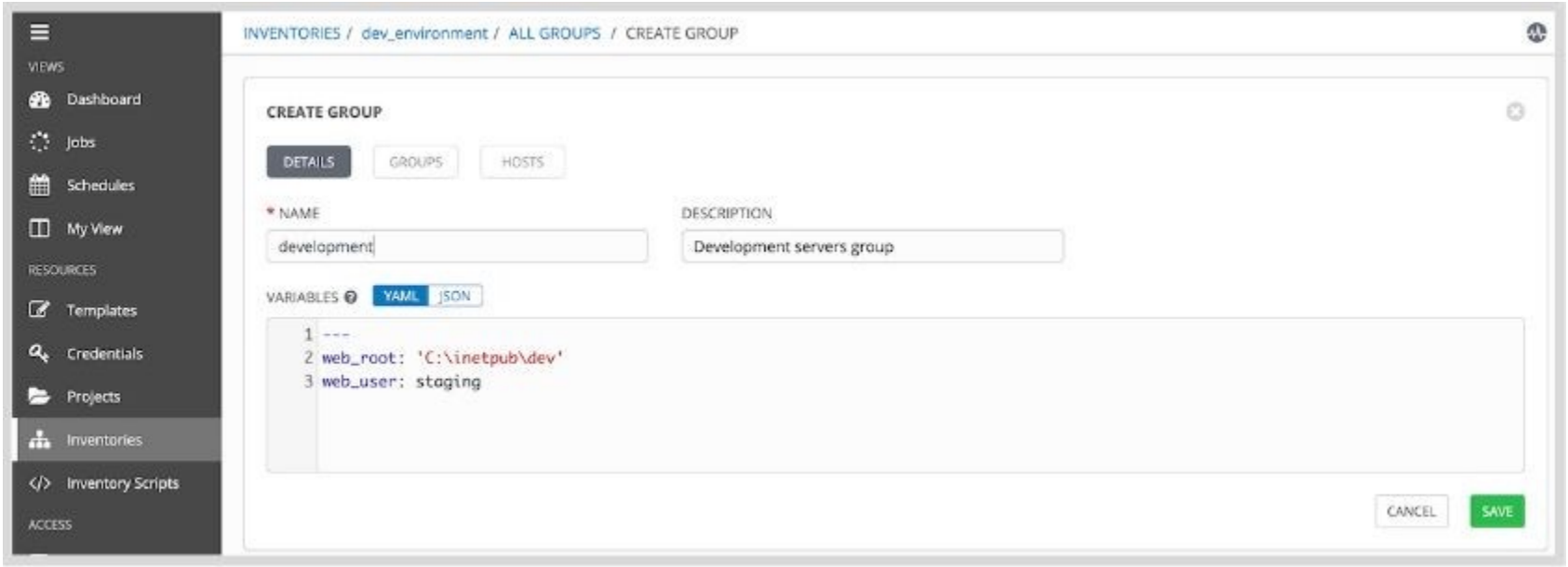


Inventory Groups

- 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**.

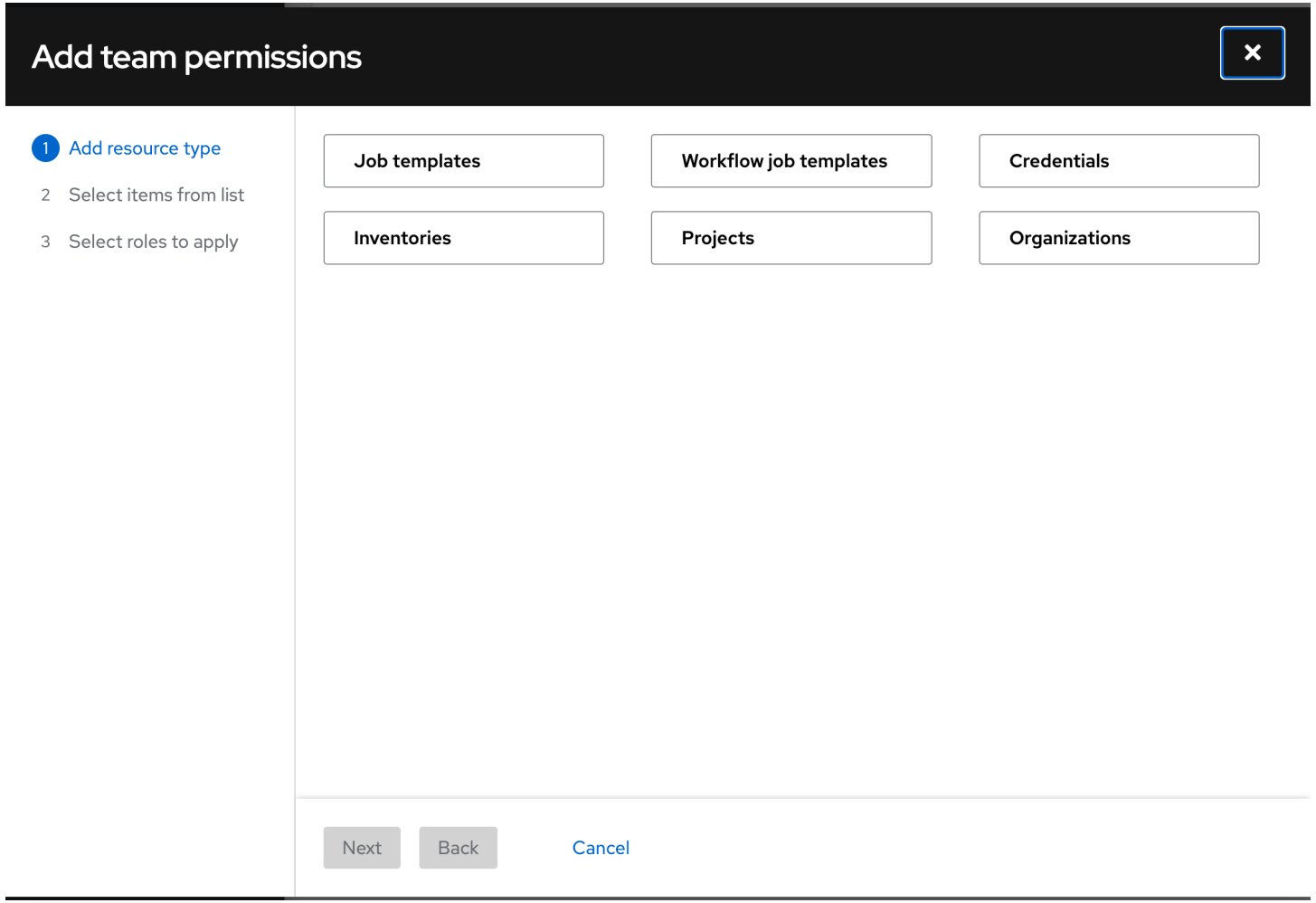


Inventory Groups

- 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

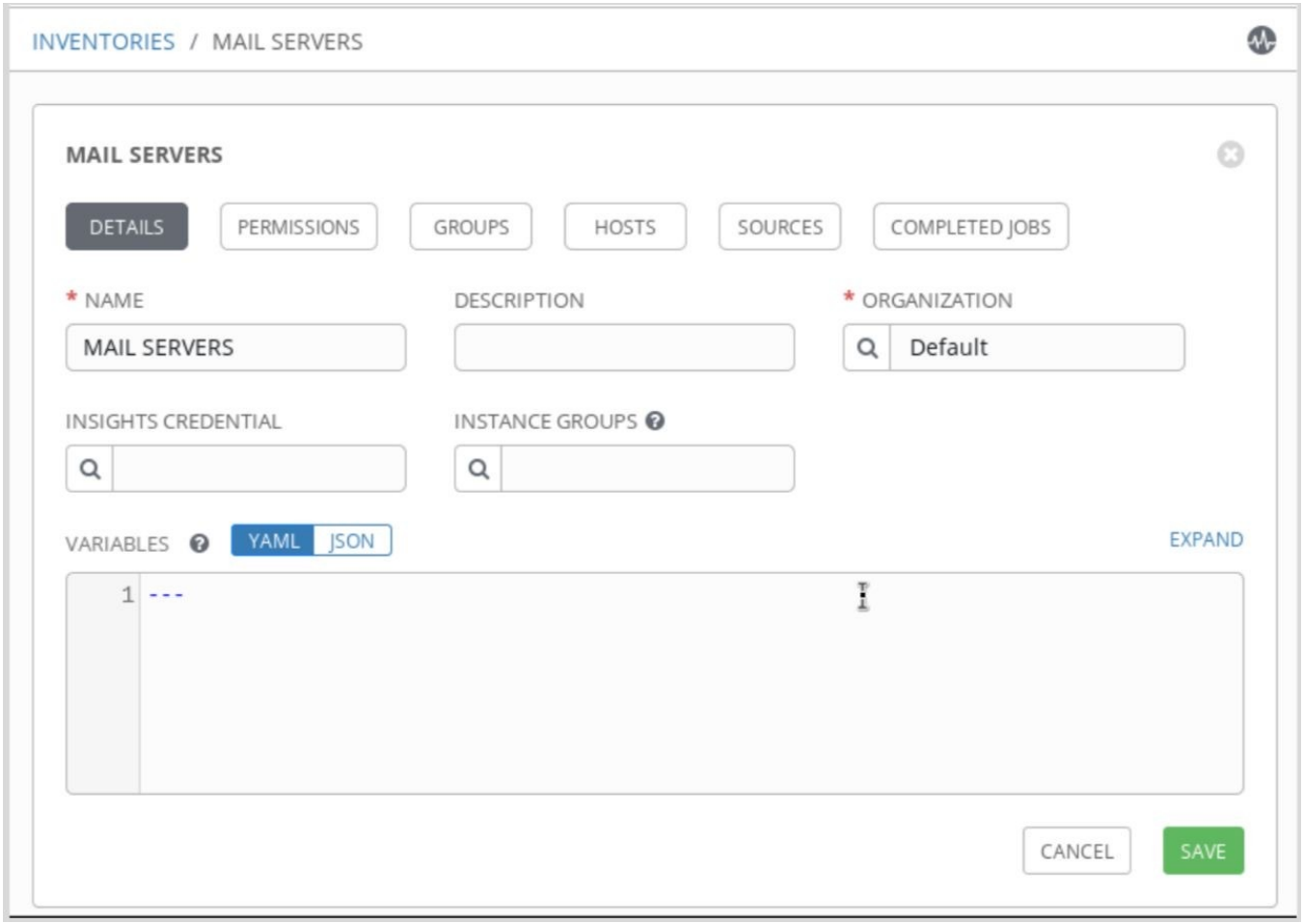
Role	Description
Admin	The inventory **Admin** role grants users full permissions over an inventory. These permissions include deletion and modification of the inventory. In addition, this role also grants permissions associated with the inventory roles **Use**, **Ad Hoc**, and **Update**.
Use	The inventory **Use** 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 **Ad Hoc** role grants users the ability to use the inventory to execute ad hoc commands.
Update	The inventory **Update** role grants users the ability to update a dynamic inventory from its external data source.
Read	The inventory **Read** role grants users the ability to view the contents of an inventory.

Inventory Variables

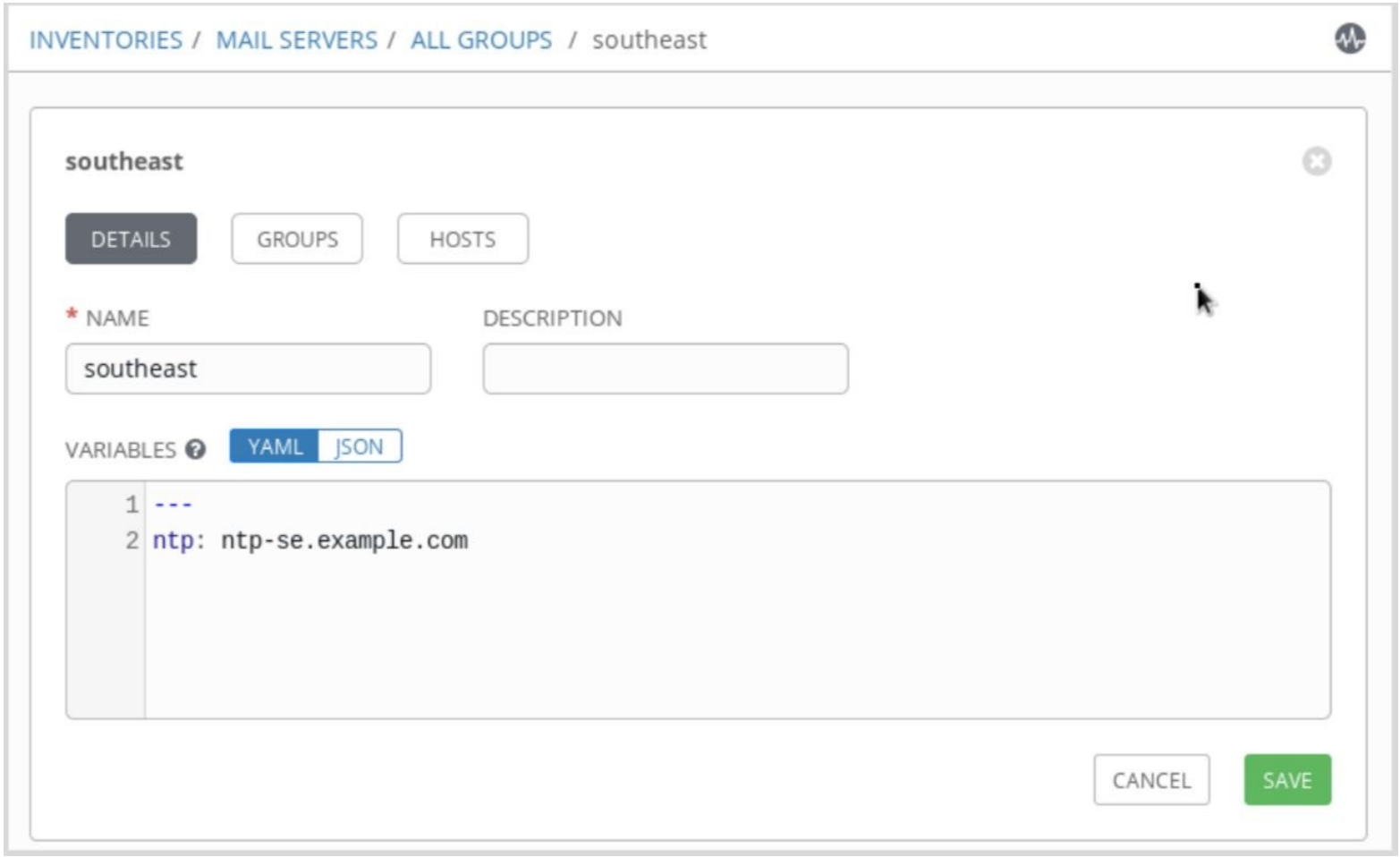
When you manage a static inventory in the Ansible Platform web UI, you may define inventory variables directly in the inventory objects.

- 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`**.

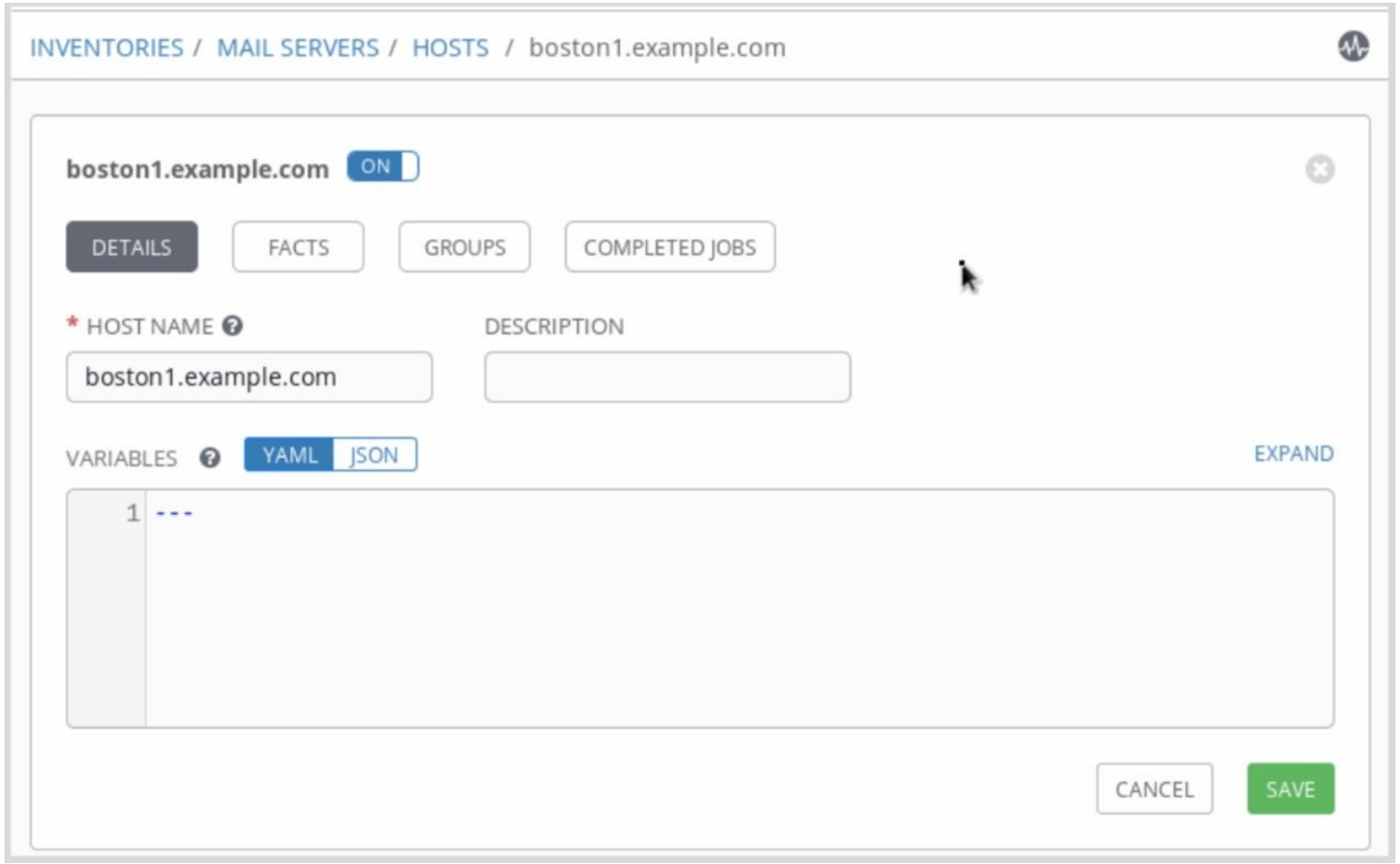
Inventory Variables



Inventory Group Variables



Inventory Host Variables

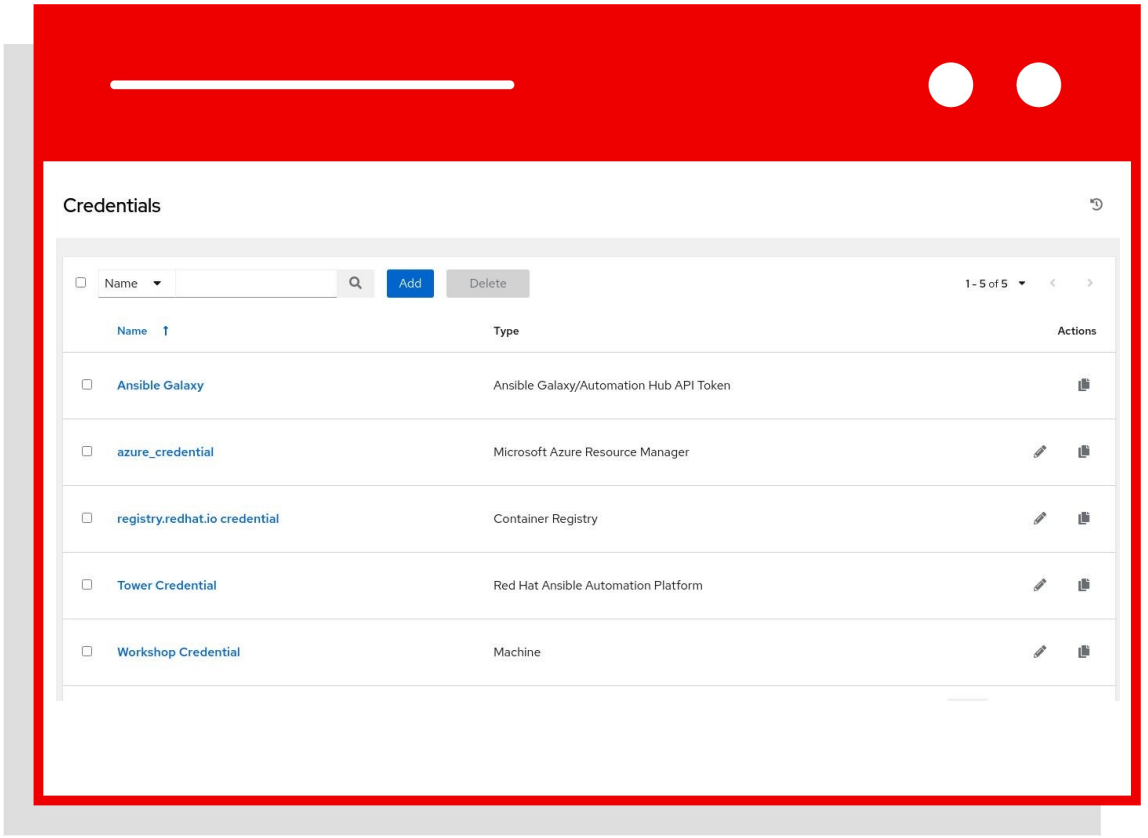


Credentials

Credentials are utilized by Automation Controller for authentication with various external resources:

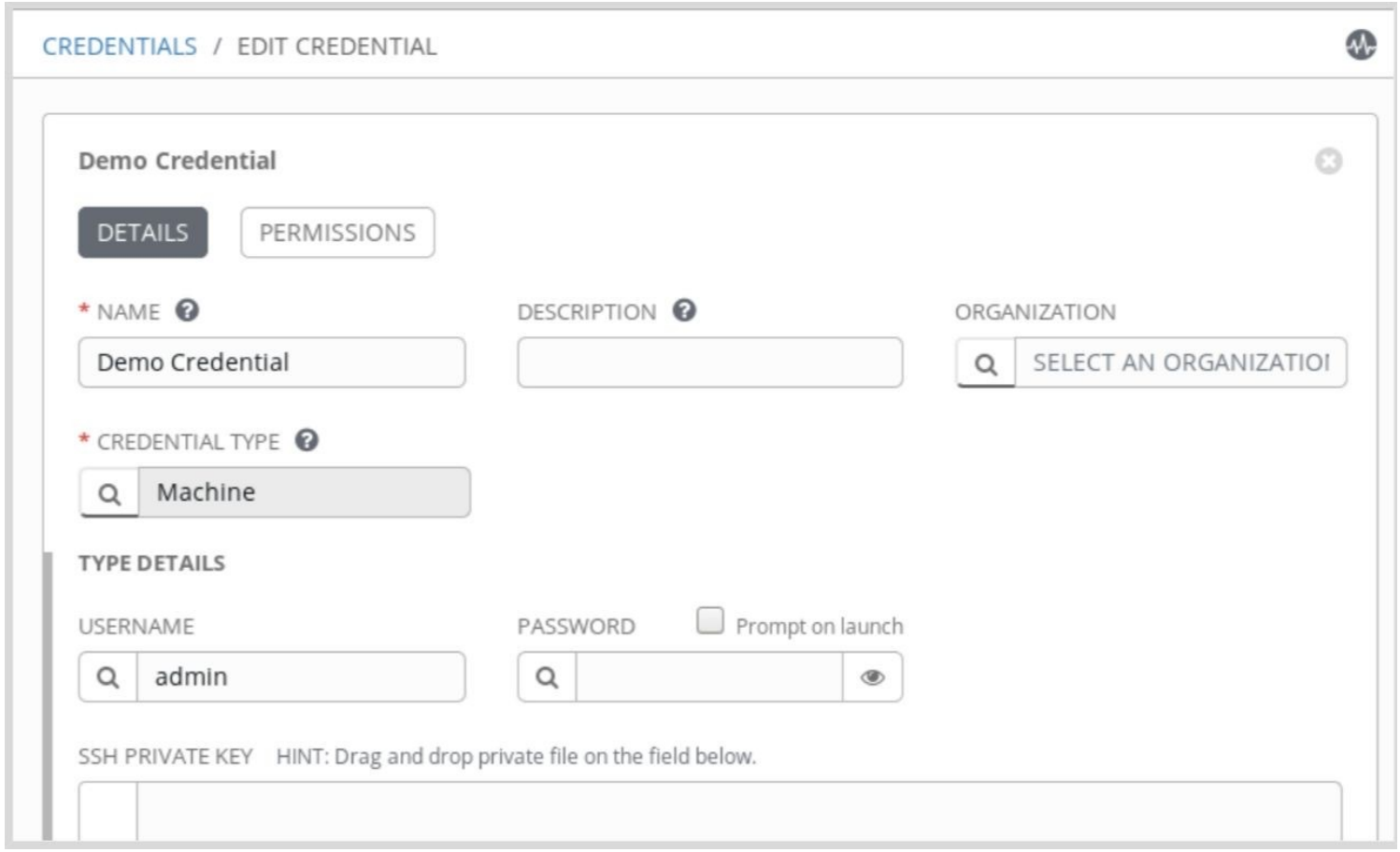
- Connecting to remote machines to run jobs
- Syncing with inventory sources
- Importing project content from version control systems
- Connecting to and managing network devices

Centralized management of various credentials allows end users to leverage a secret without ever exposing that secret to them.



Credentials

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

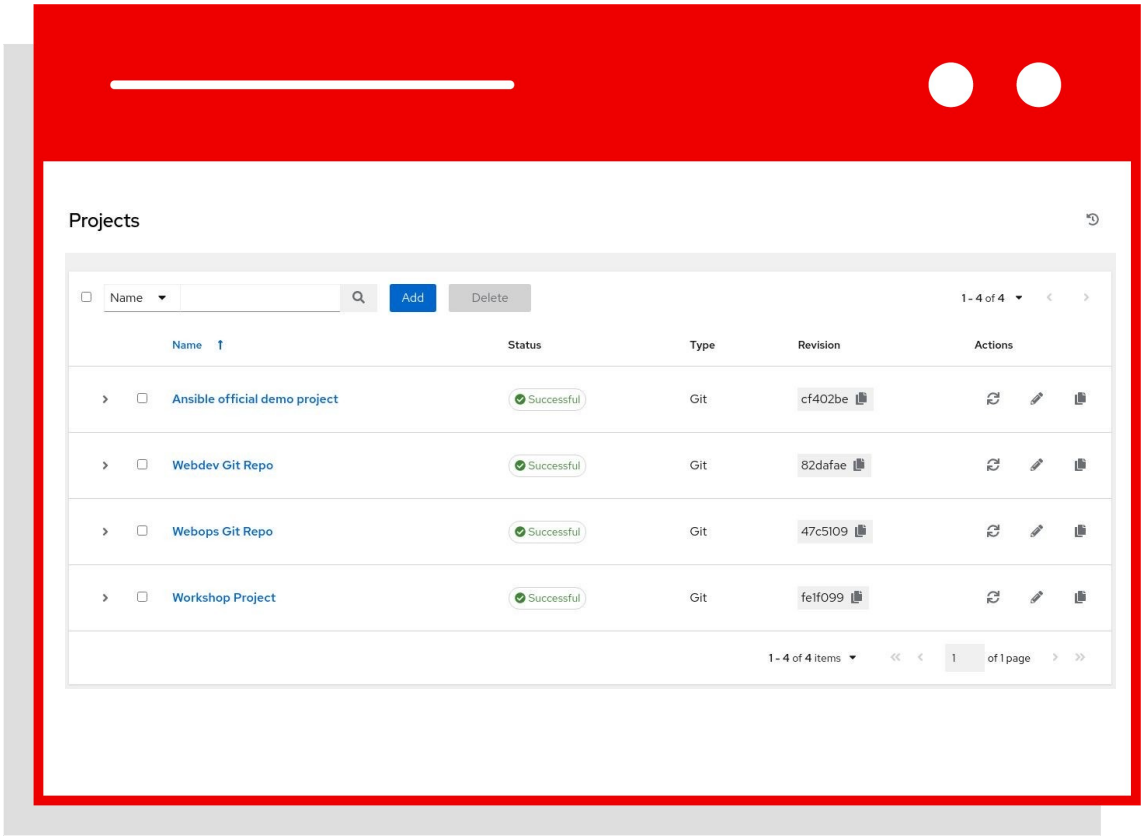


Lab: AAP Inventories, credentials, and ad-hoc commands

PROJECT

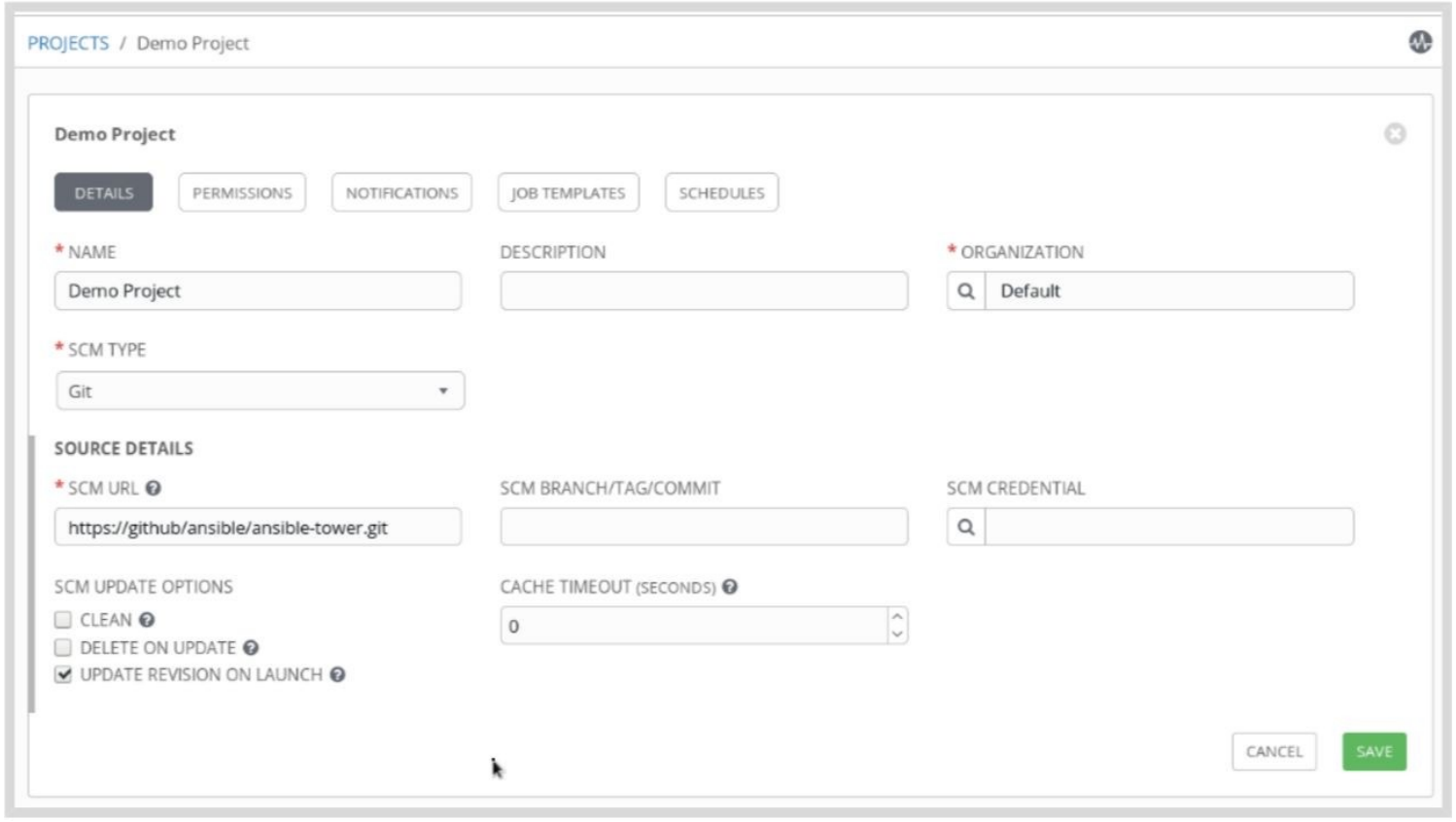
A project is a logical collection of Ansible Playbooks, represented in Ansible Automation Controller.

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.



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.



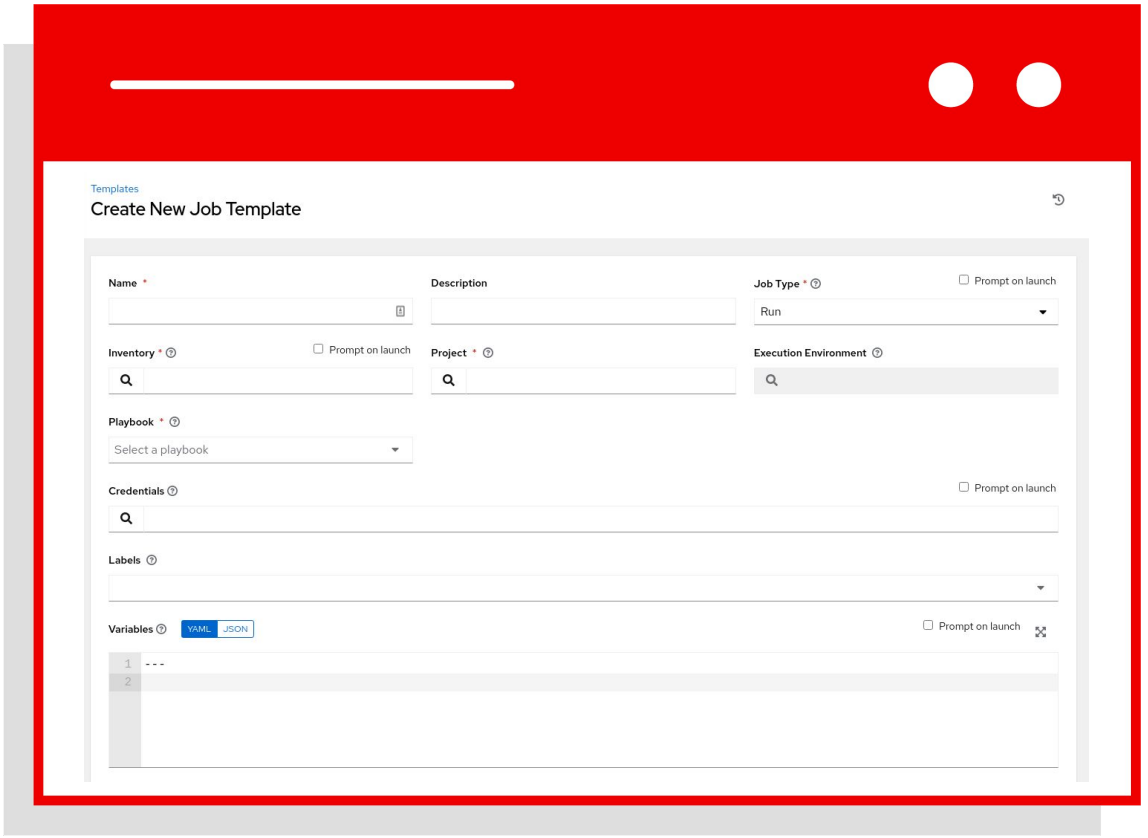
JOB TEMPLATES

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.

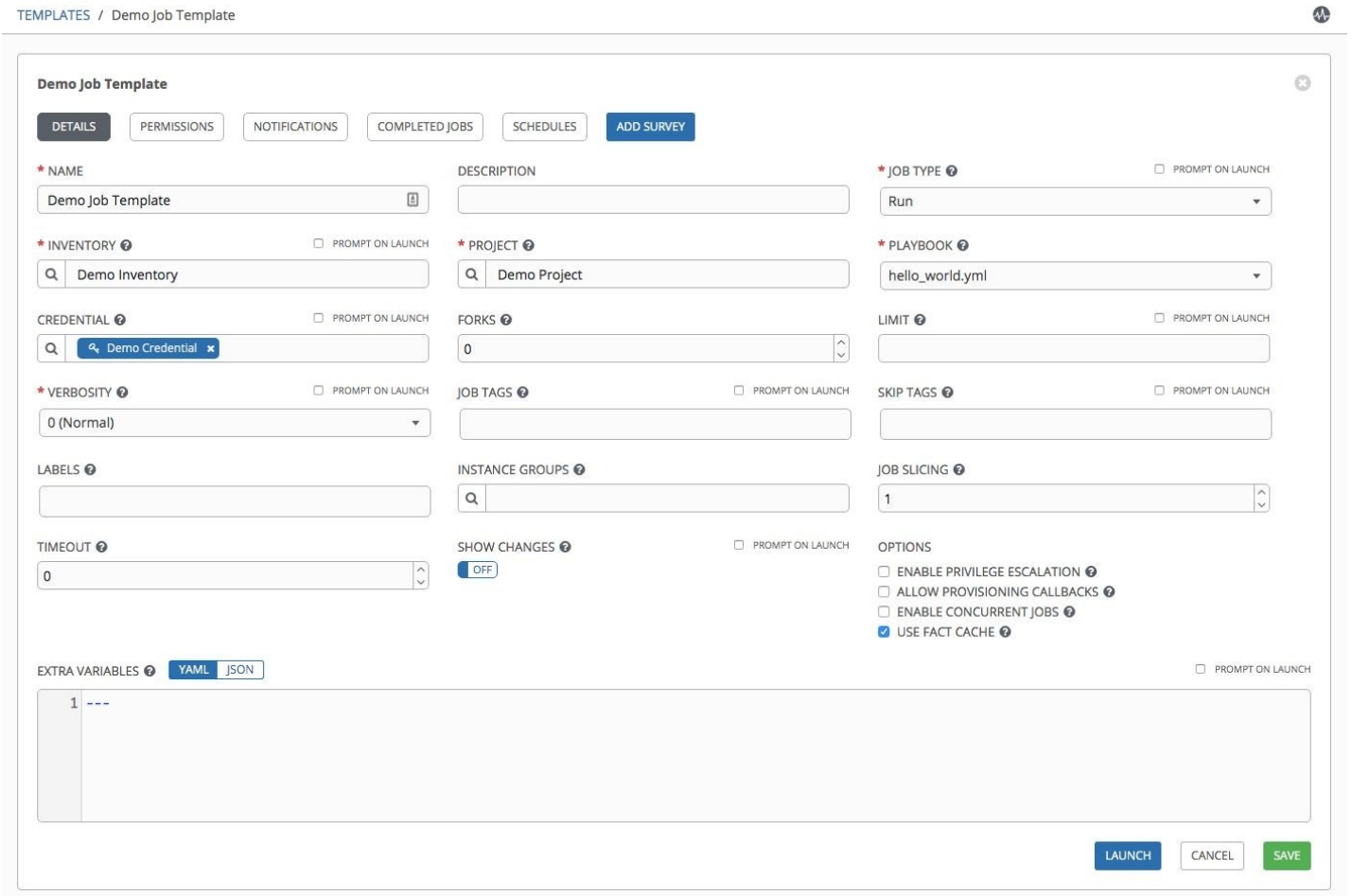
A **Job Template** requires:

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



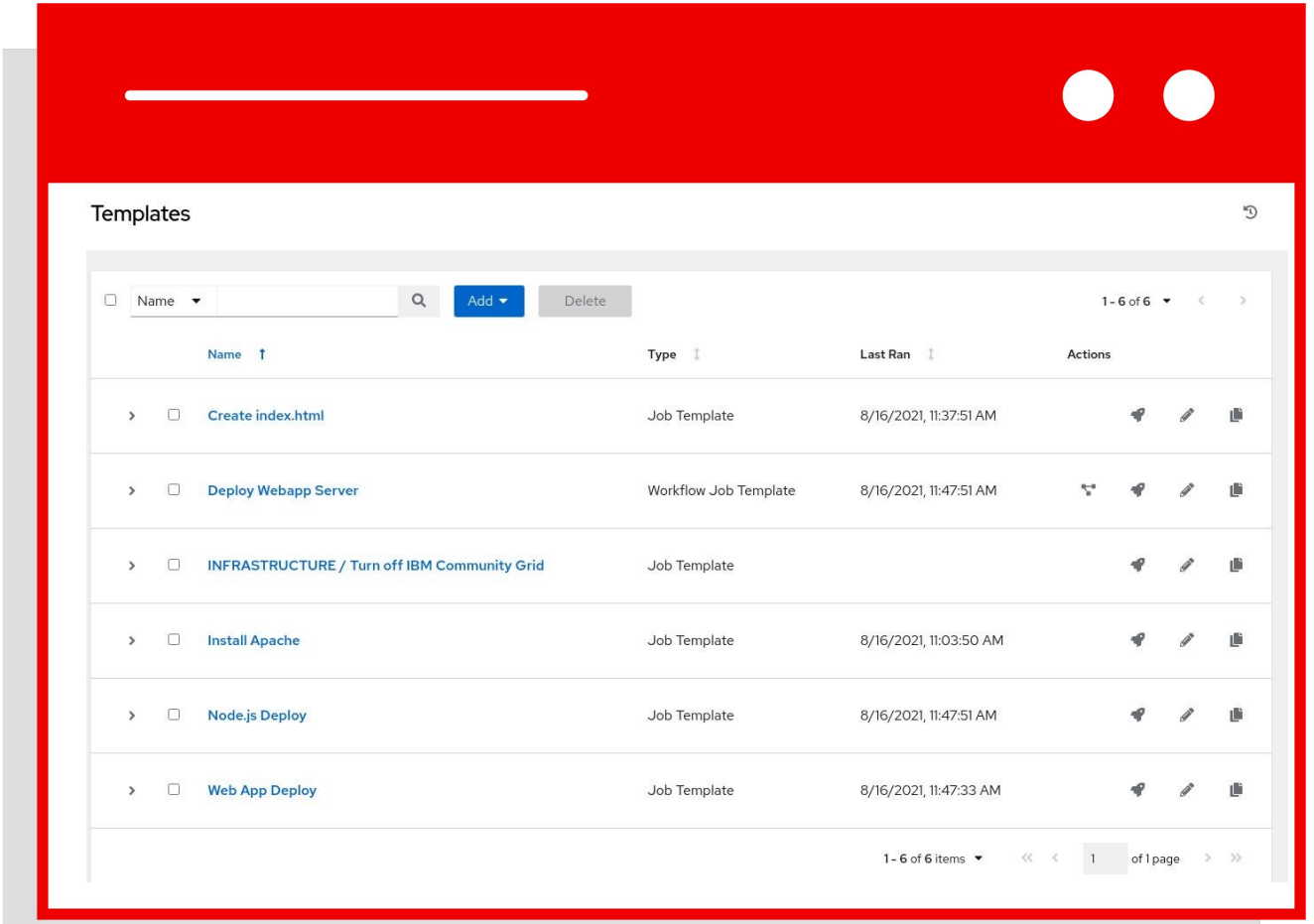
JOB TEMPLATES

- 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.



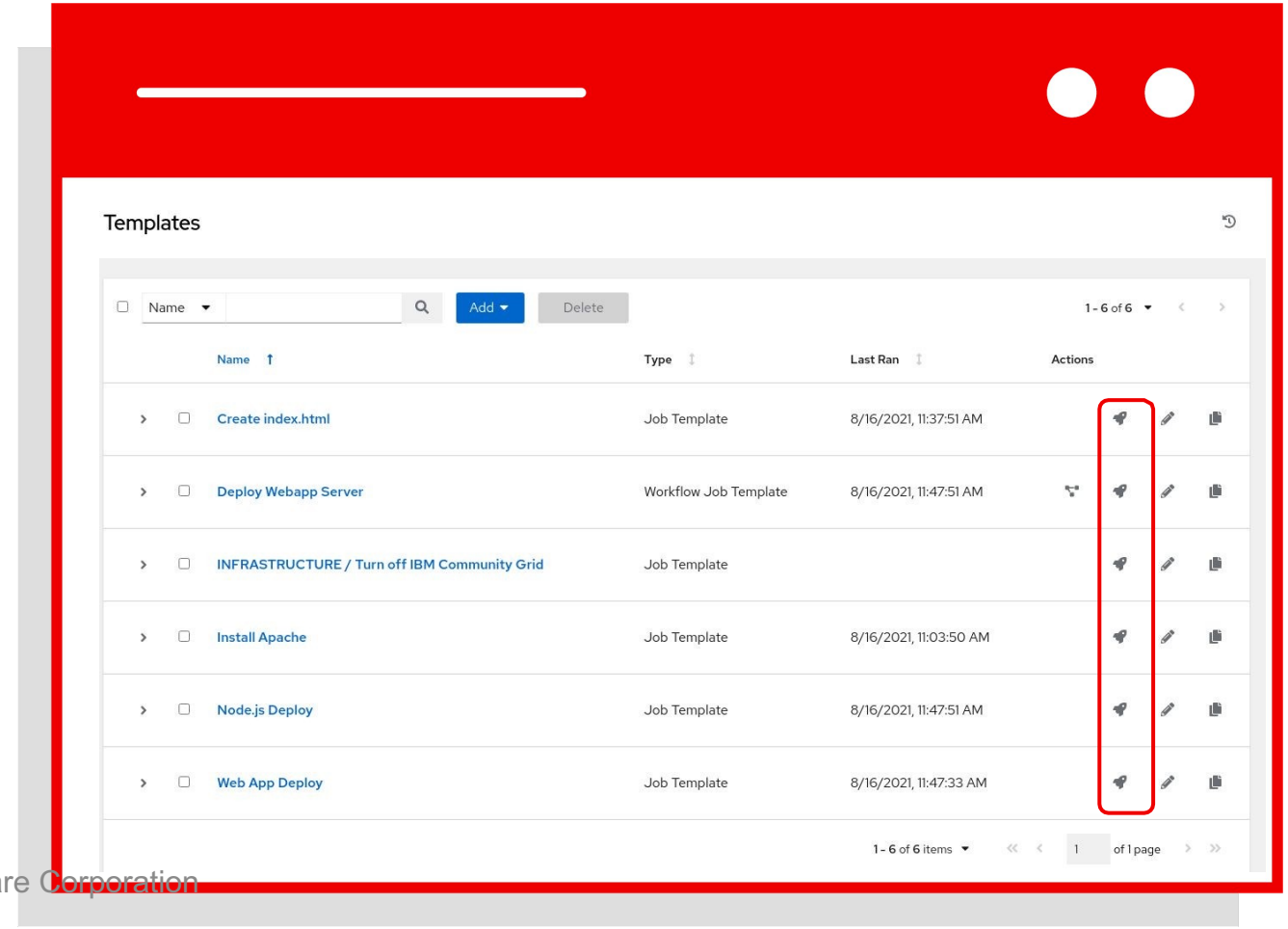
EXPANDING JOB TEMPLATES

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



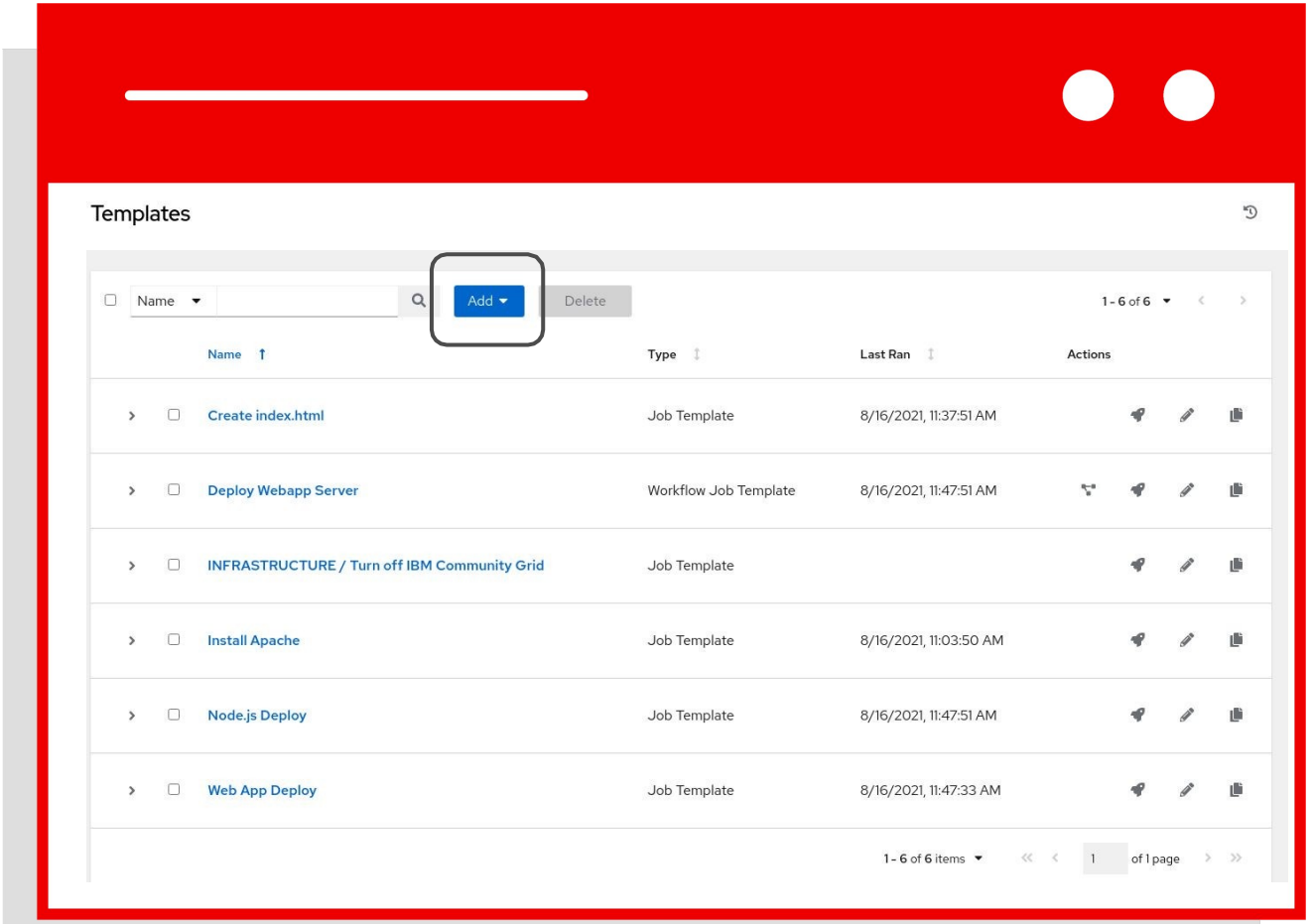
EXECUTING AN EXISTING JOB

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



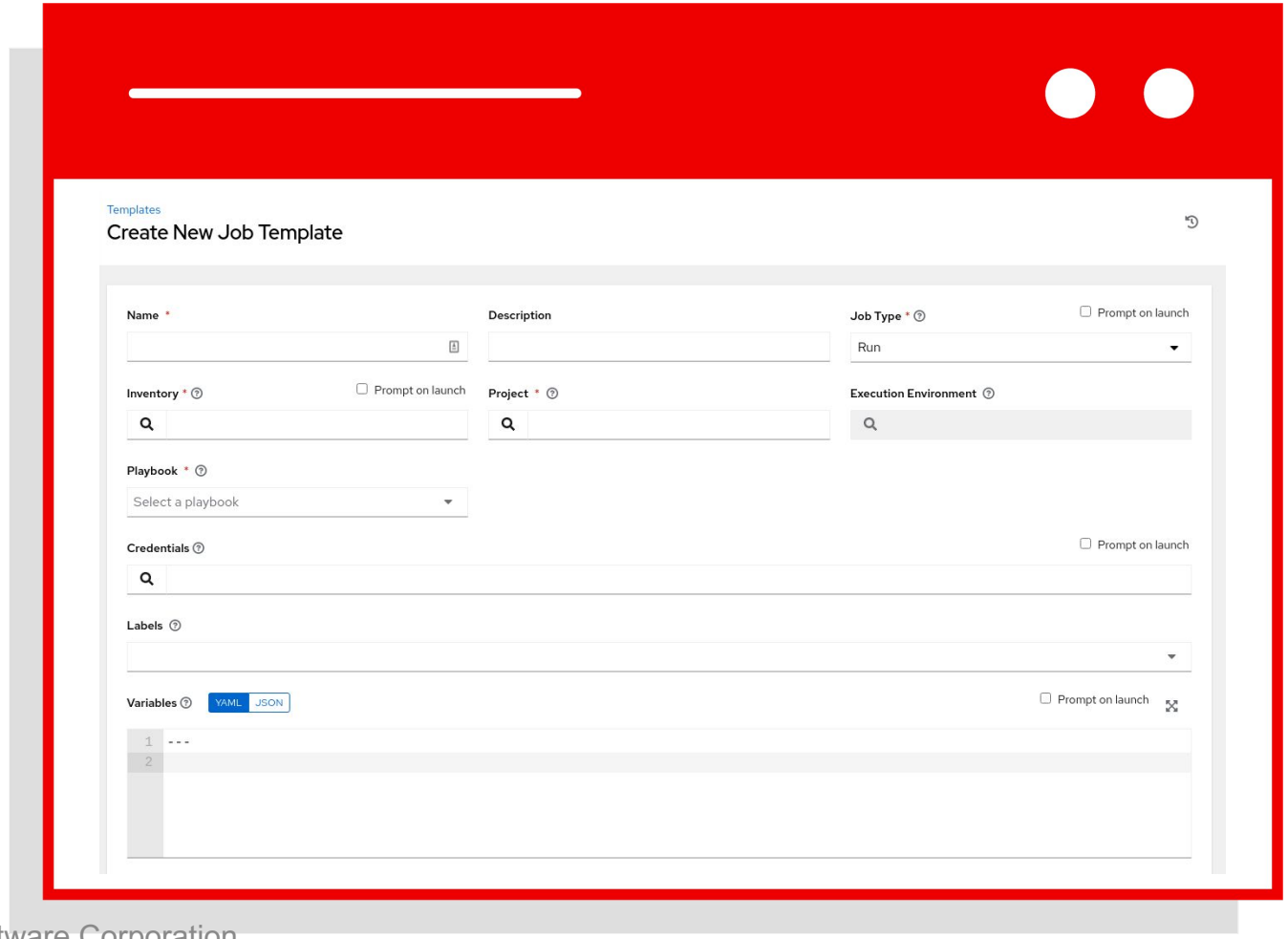
CREATING A NEW JOB TEMPLATE (1/2)

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



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 .

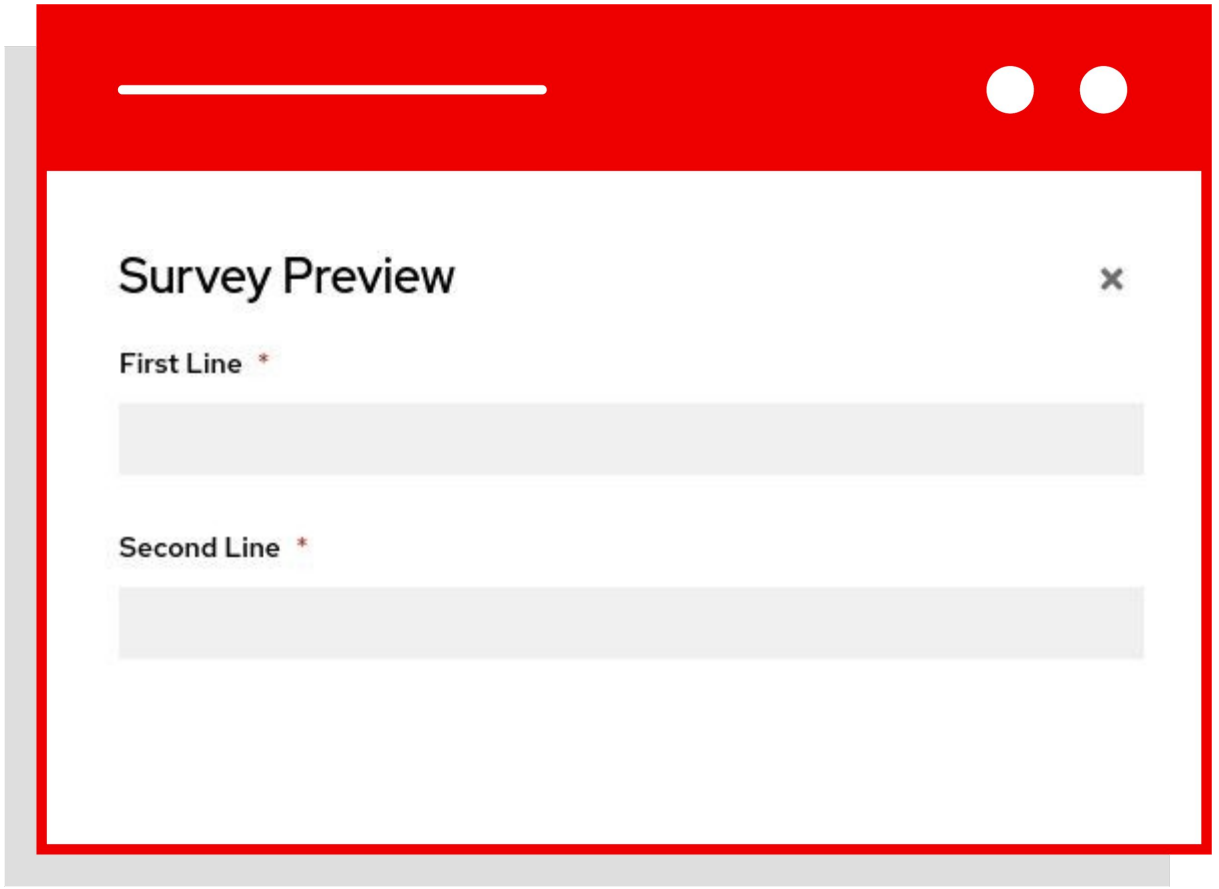


SURVEYS

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.

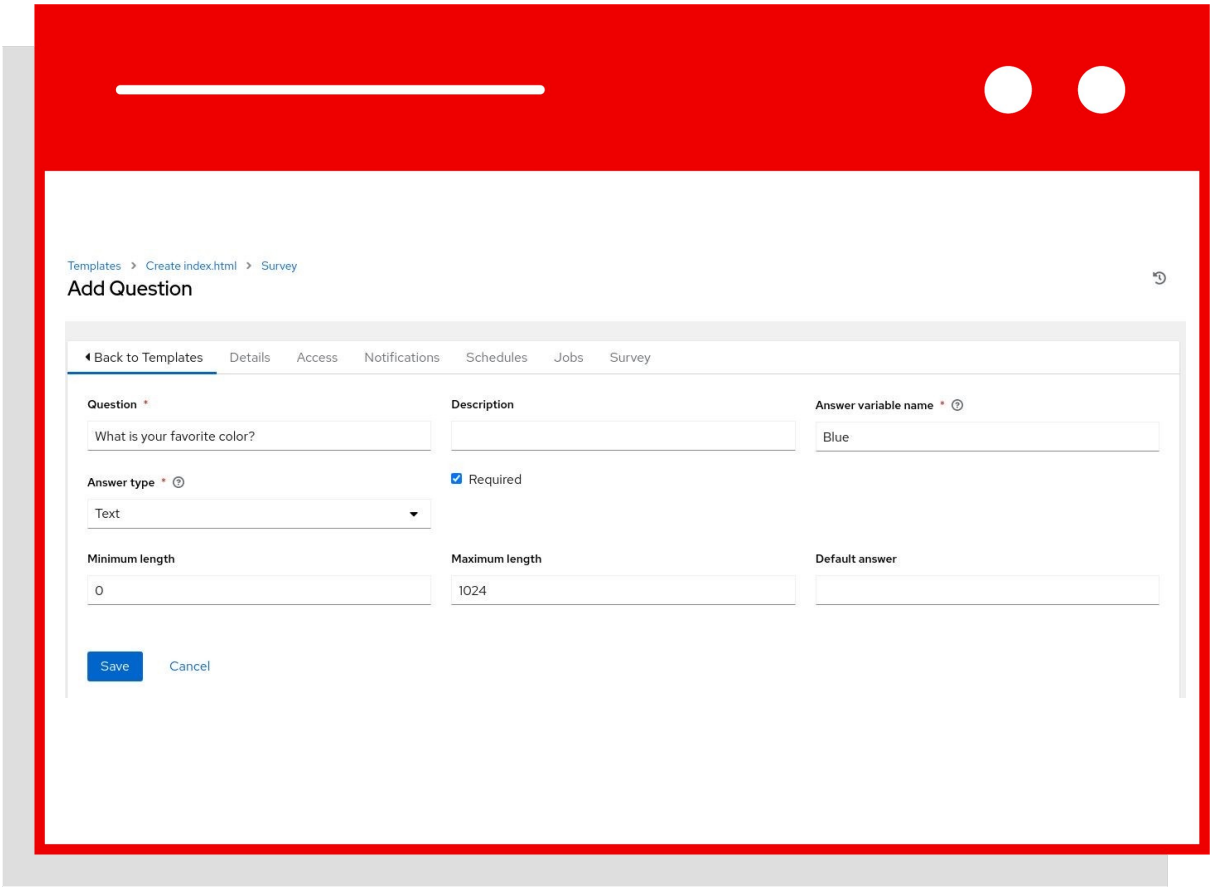
Combine that with Controller's role-based access control, and you can build simple, easy self-service for your users.



CREATING A SURVEY (1/2)

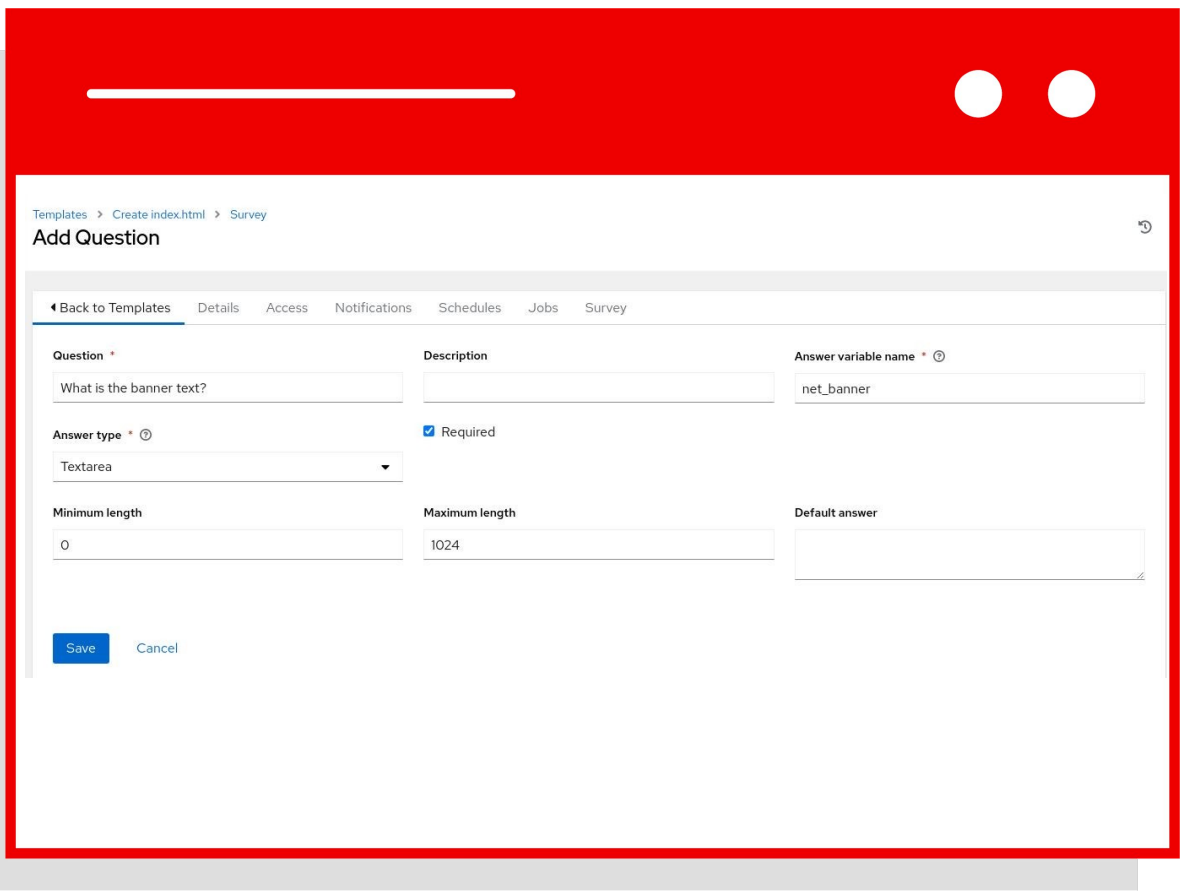
Once a Job Template is saved, the Survey menu will have an **Add Button**

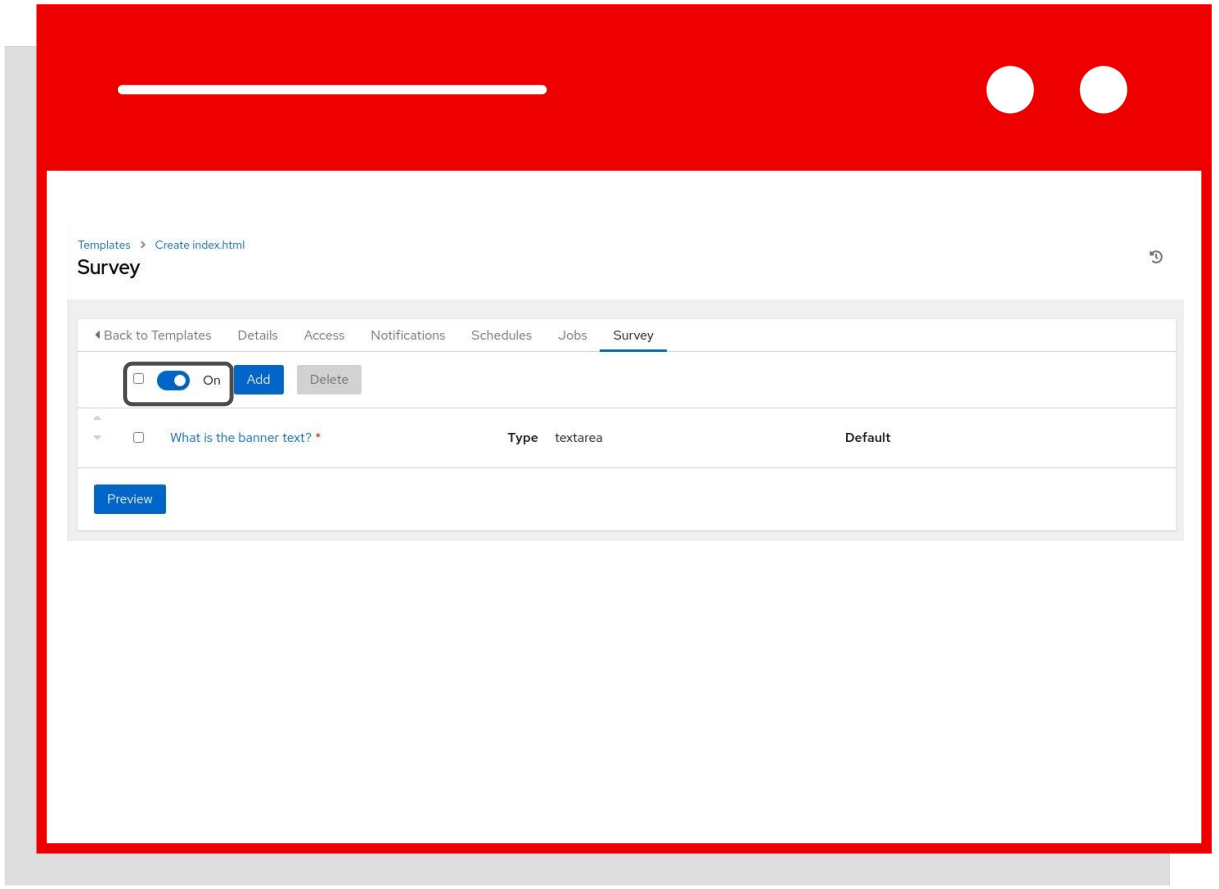
Click the button to open the Add Survey window.



CREATING A SURVEY (2/2)

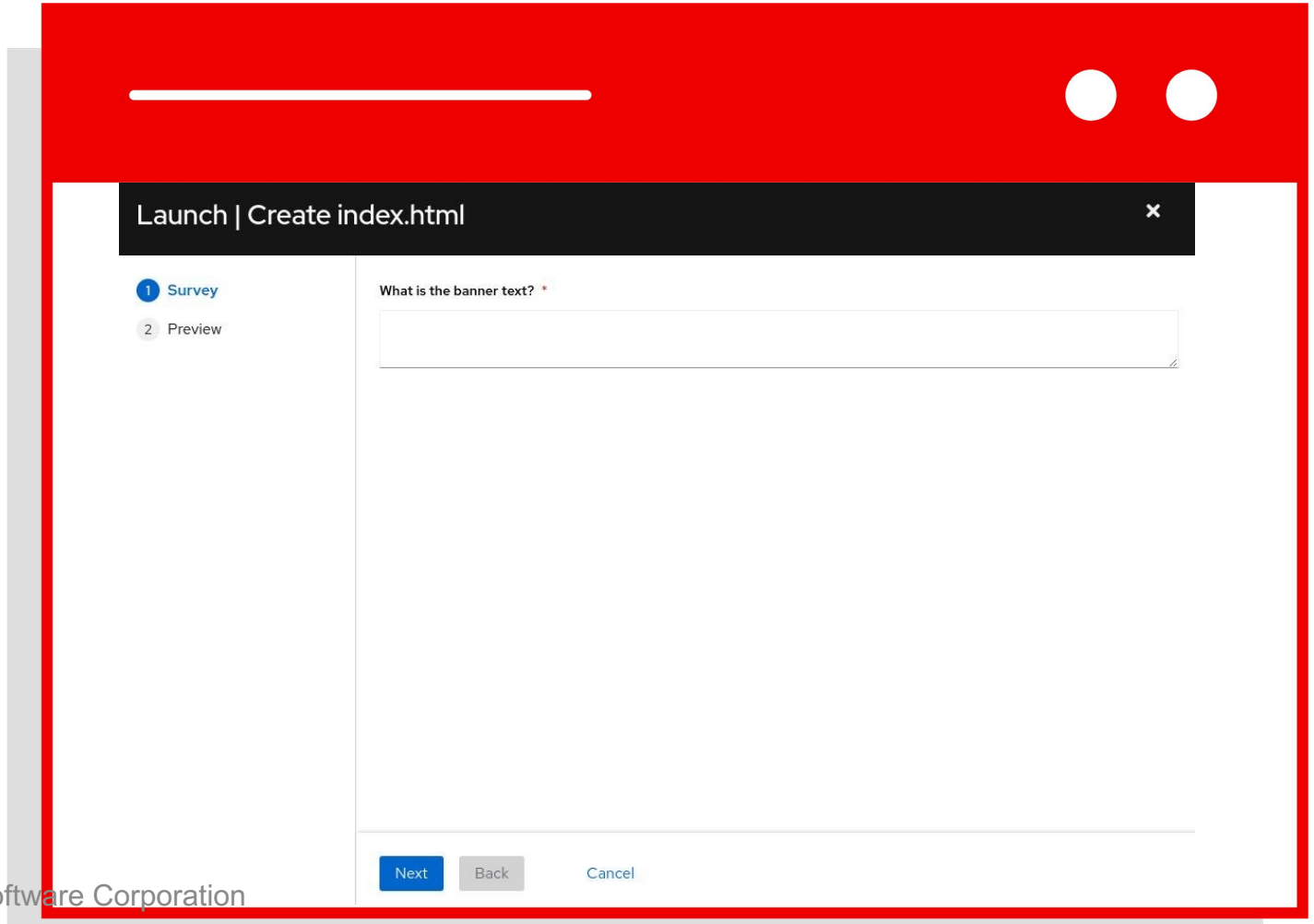
The Add Survey window allows the Job Template to prompt users for one or more questions. The answers provided become variables for use in the Ansible Playbook.





USING A SURVEY

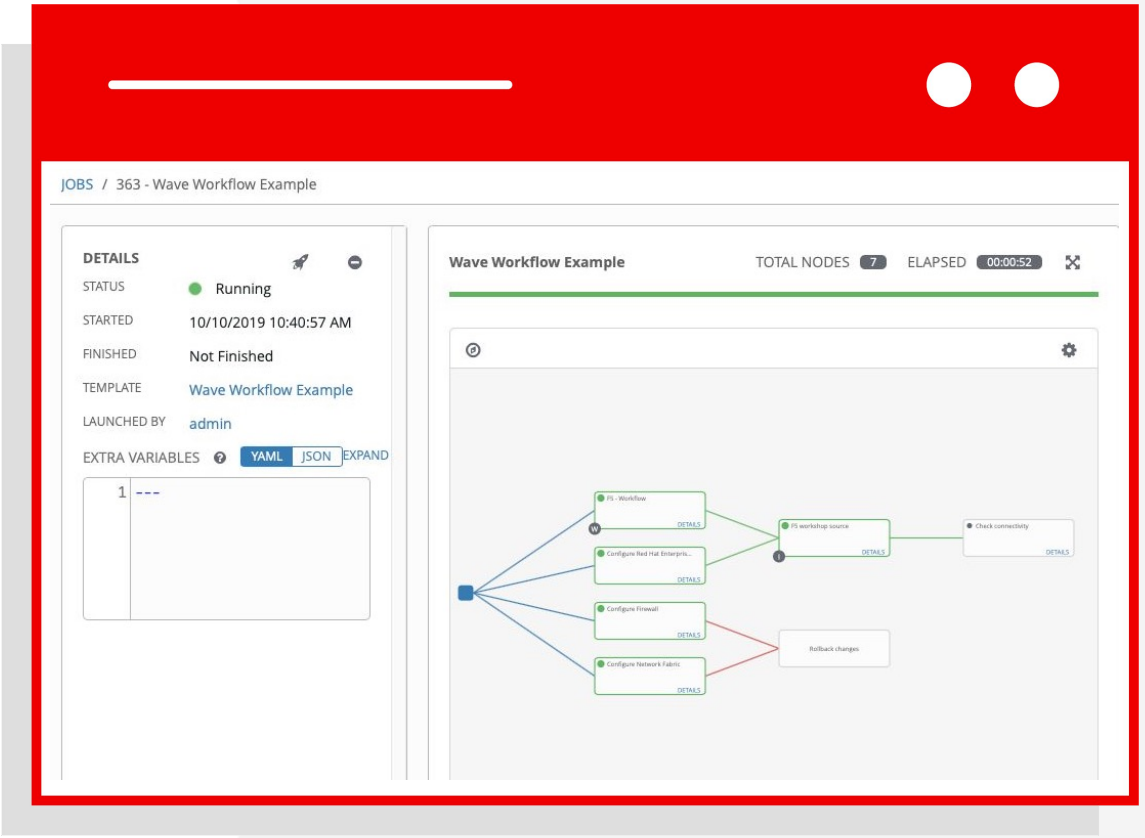
When launching a job, the user will now be prompted with the Survey. The user can be required to fill out the Survey before the Job Template will execute.



WORKFLOWS

Combine automation to create something bigger

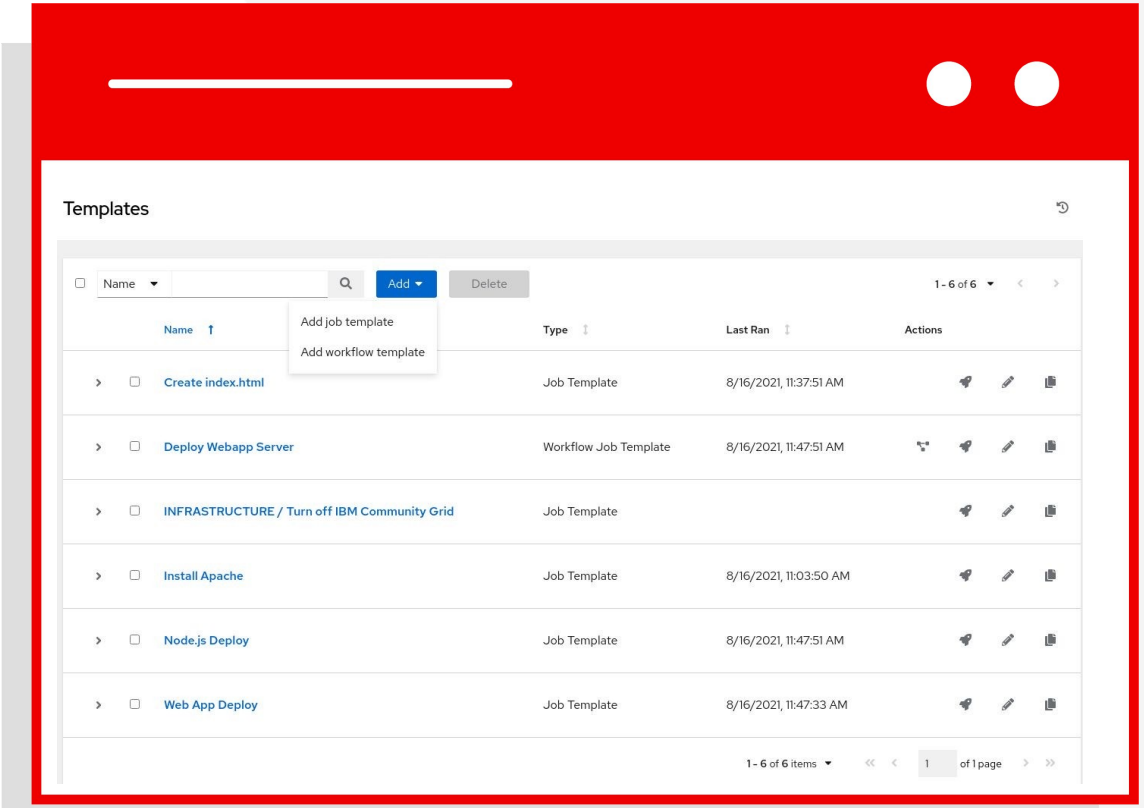
- Workflows enable the creation of powerful holistic automation, chaining together multiple pieces of automation and events.
- Simple logic inside these workflows can trigger automation depending on the success or failure of previous steps.



ADDING A NEW TEMPLATE

- To add a new **Workflow** click on the **Add** button.

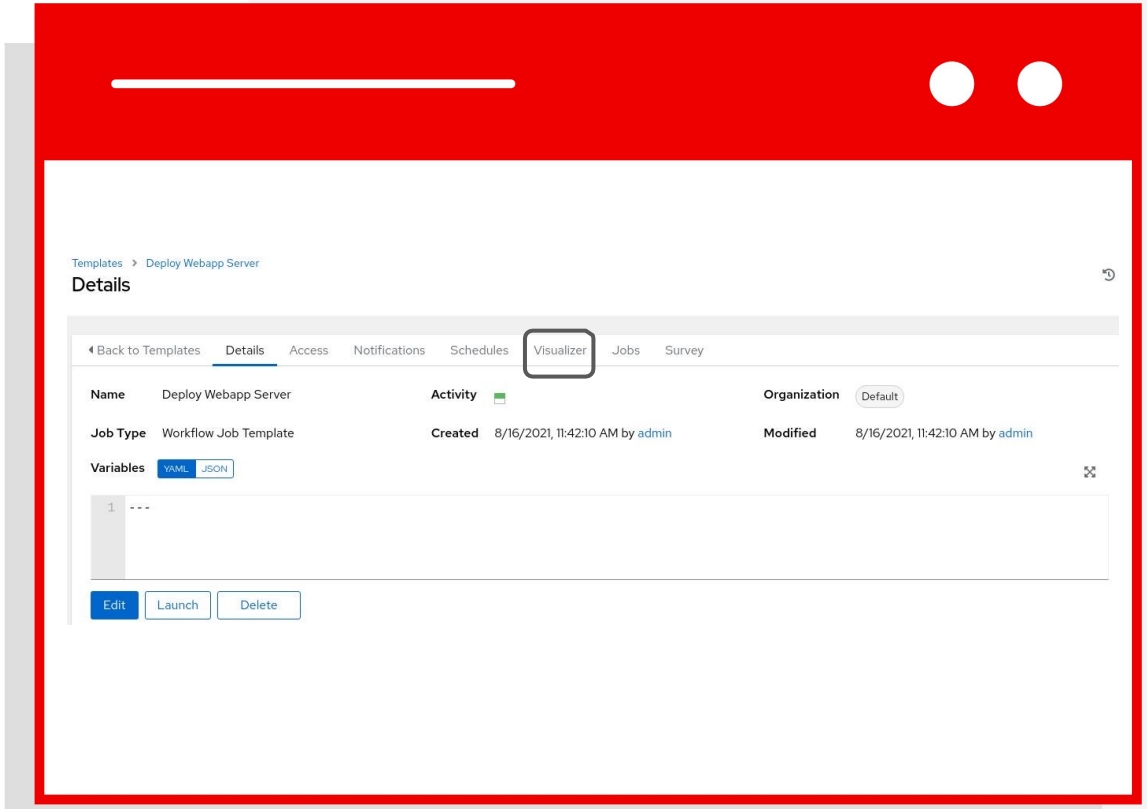
  This time select the **Add workflow template**



CREATING THE WORKFLOW

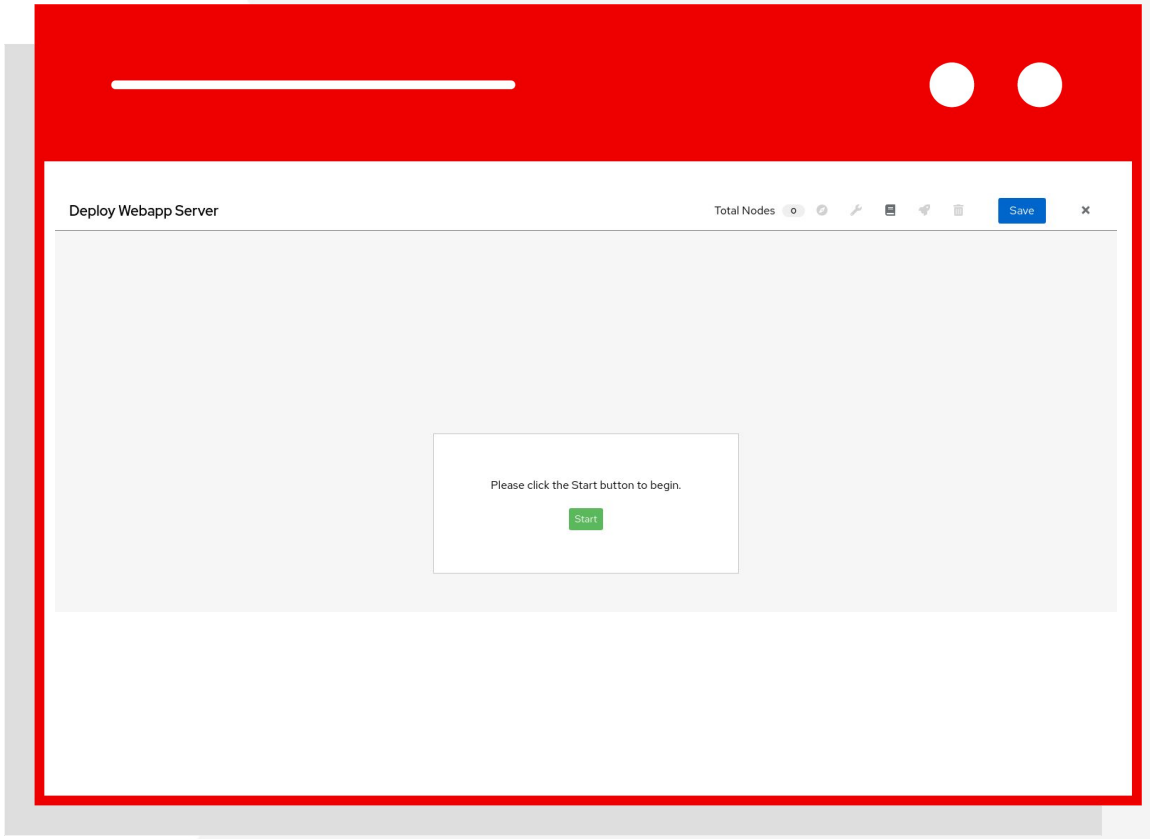
- Fill out the required parameters and click **Save.**

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



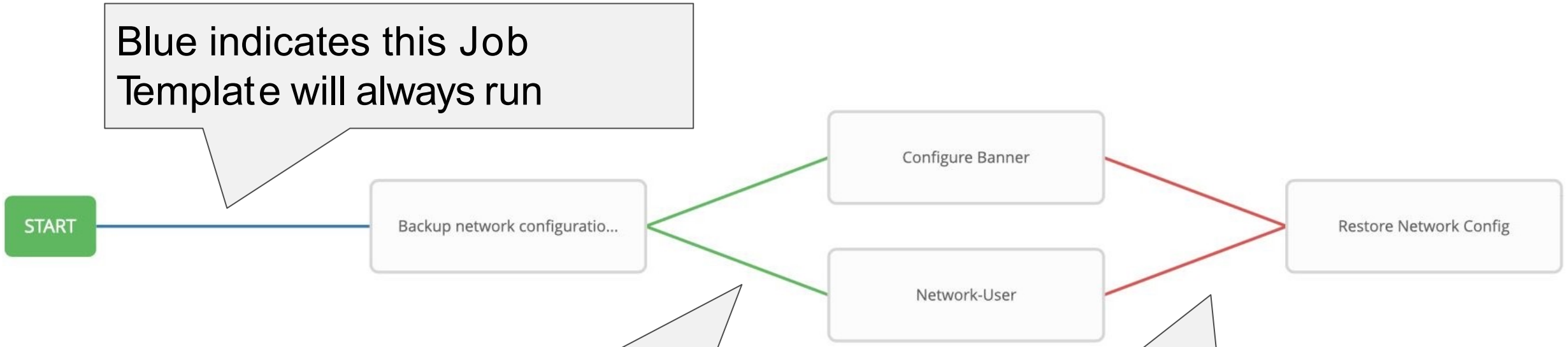
WORKFLOW VISUALIZER

- 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