Modern Automation

TEKsystems Global Services

WORKFORCE DEVELOPMENT

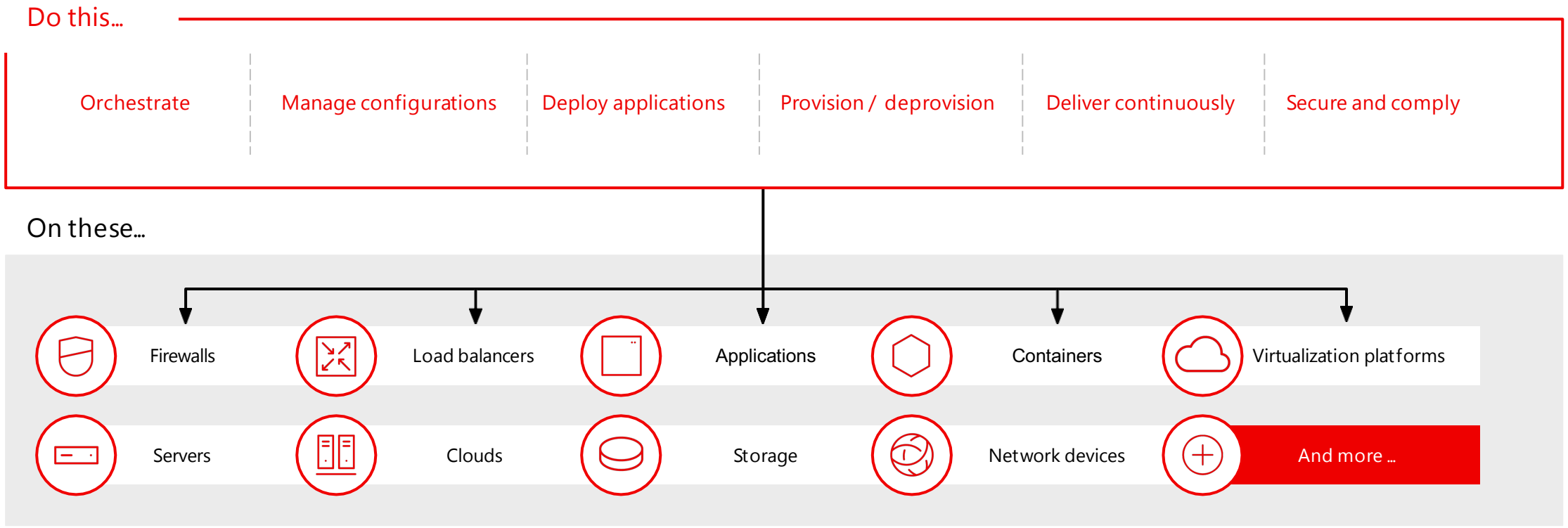
Lab page

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

Why The Ansible Automation Platform?

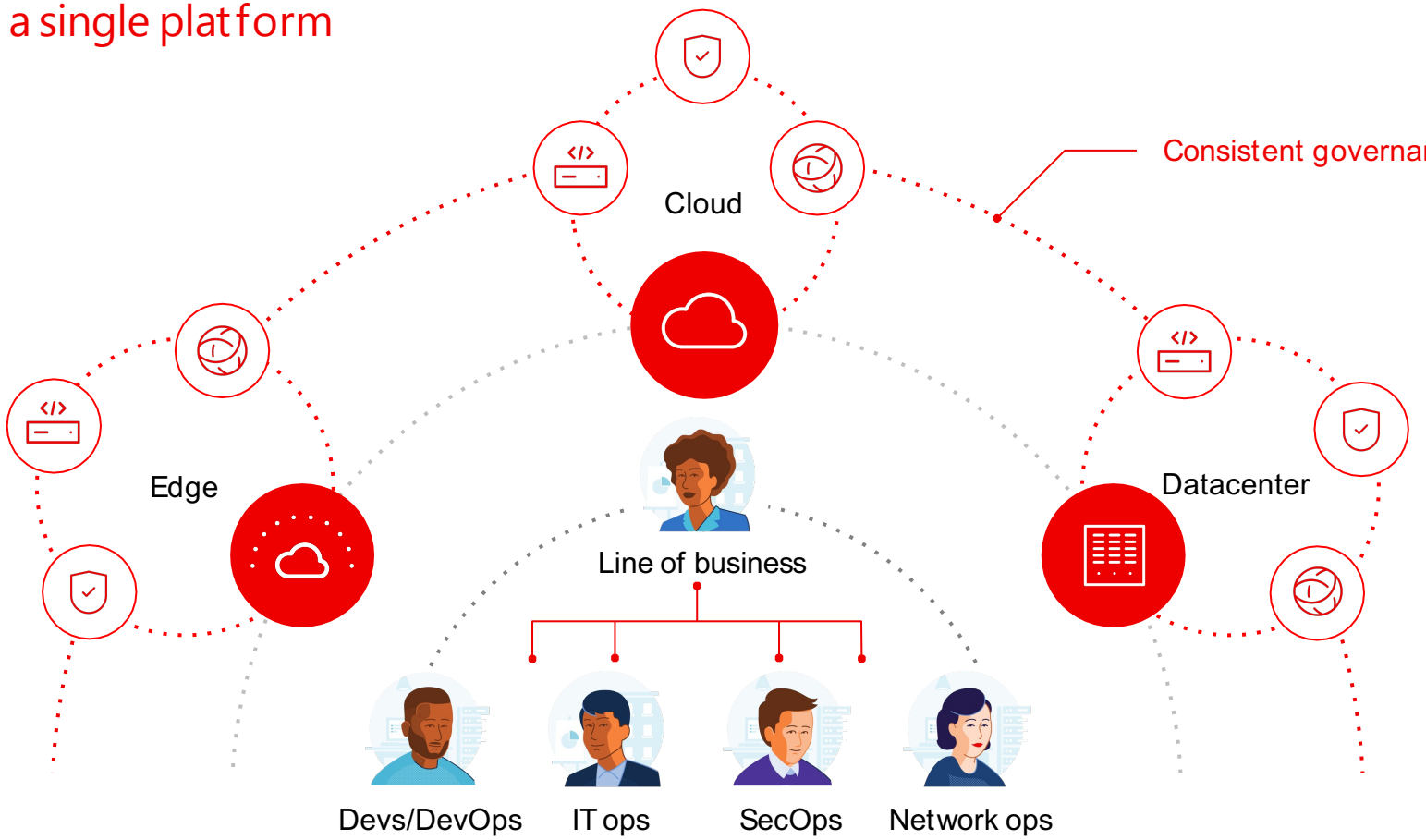
Automate the deployment and management of automation

Your entire IT footprint



Break Down Silos

Different teams a single platform



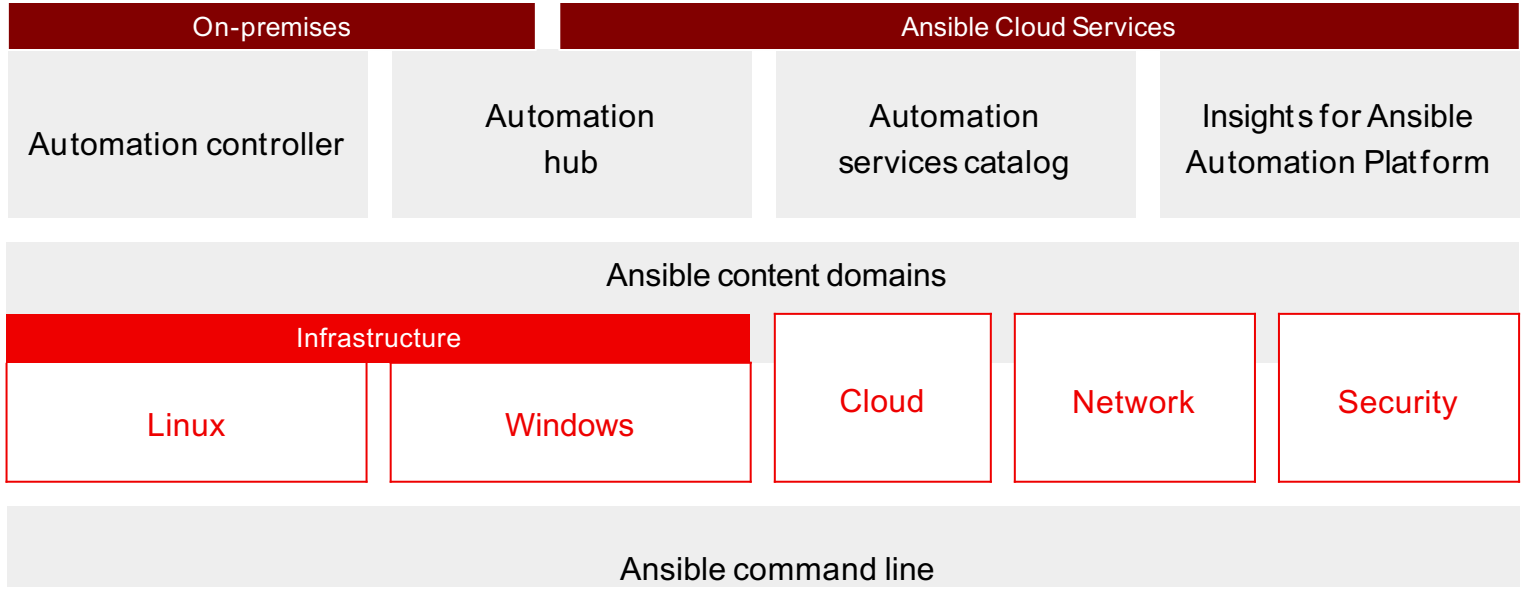
Red Hat
Ansible Automation Platform

Content creators

Operators

Domain experts

Users



Fueled by an open source community

Automation Platform Concepts

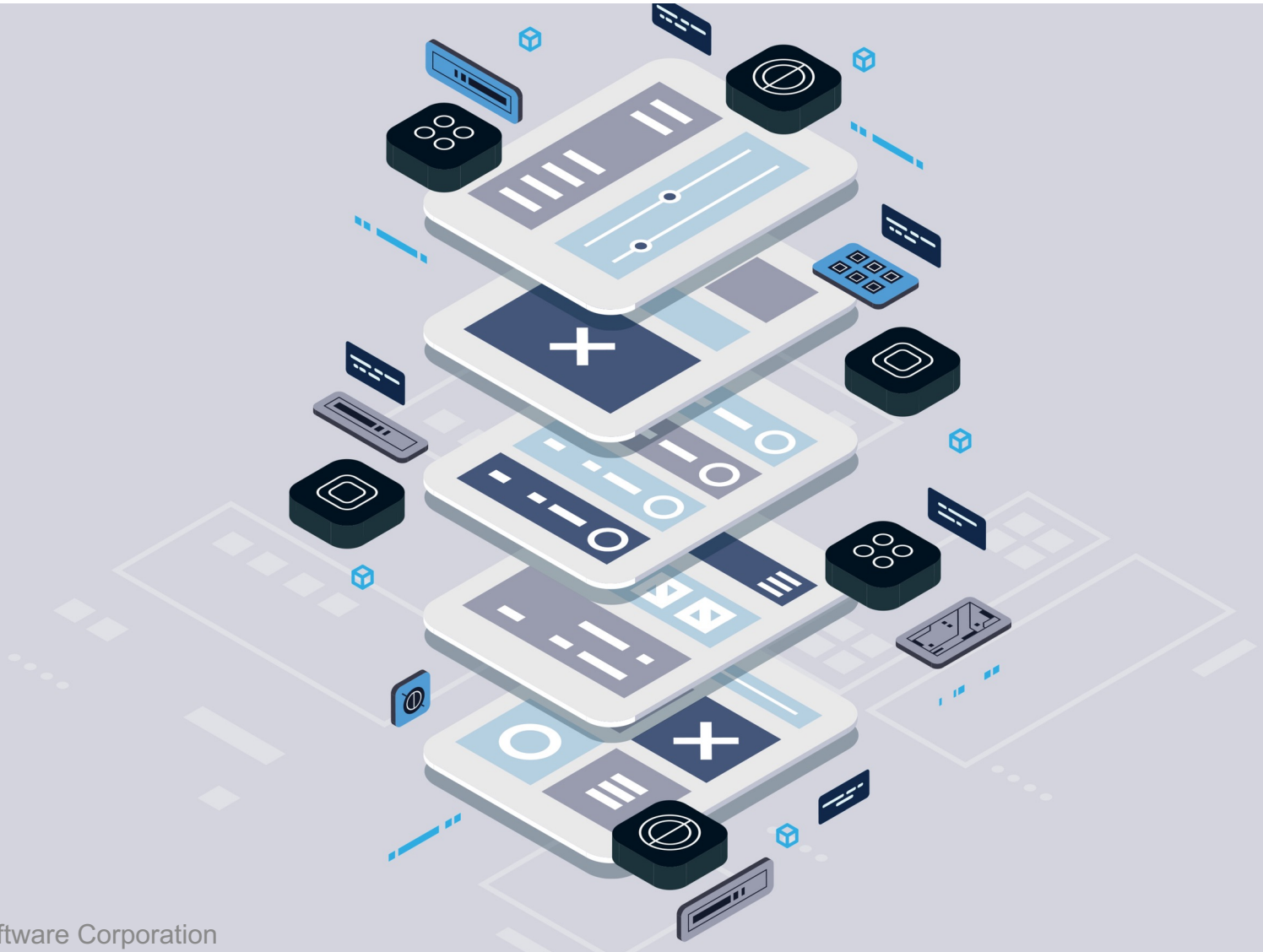
A control node is installed with Ansible and is used to run playbooks.

- 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)

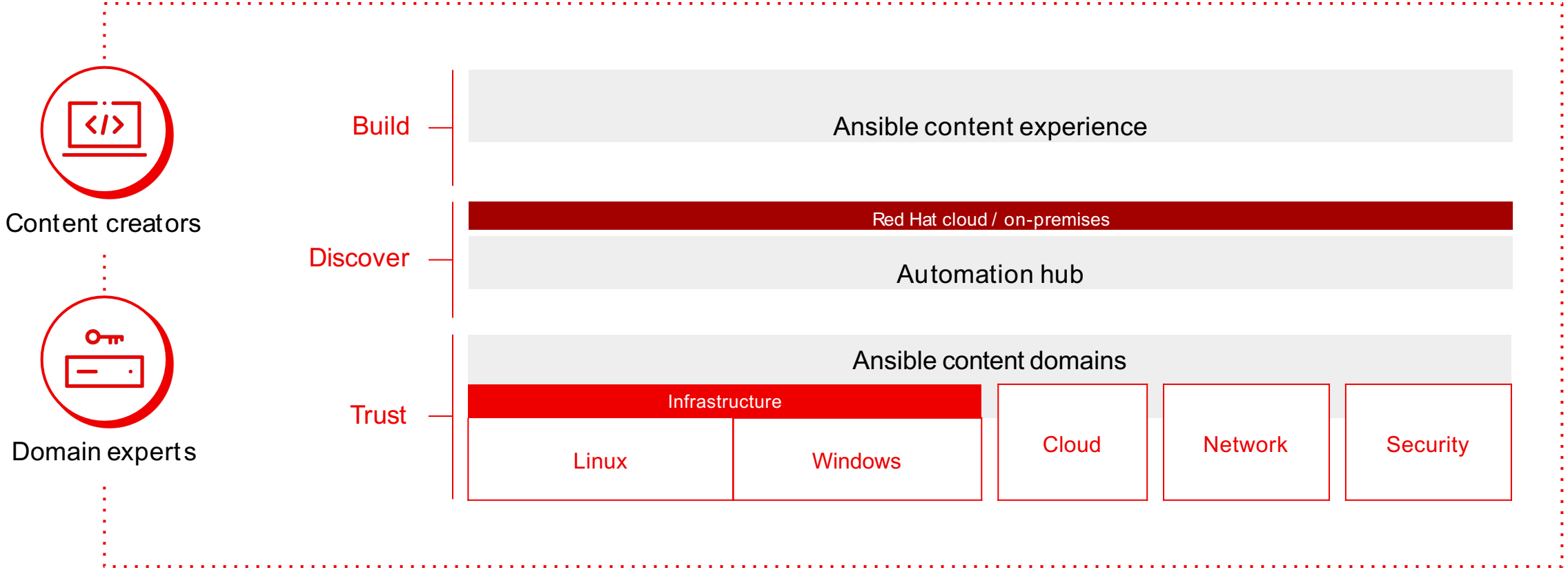
Ansible Automation Platform Infrastructure



Create Code

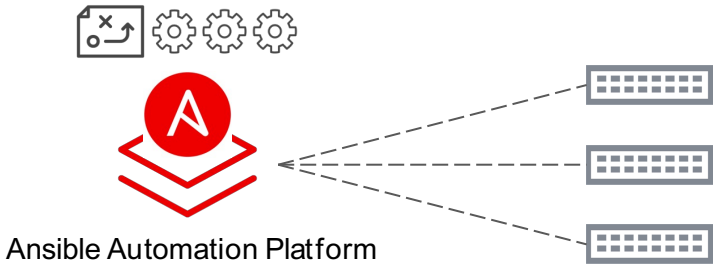
Create

The automation lifecycle



How Ansible Automation Works

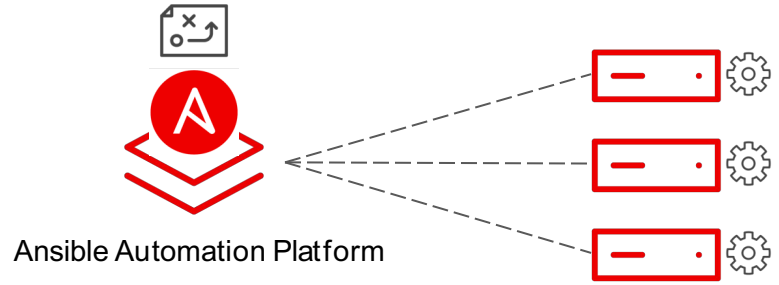
Module code is executed locally on the control node



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

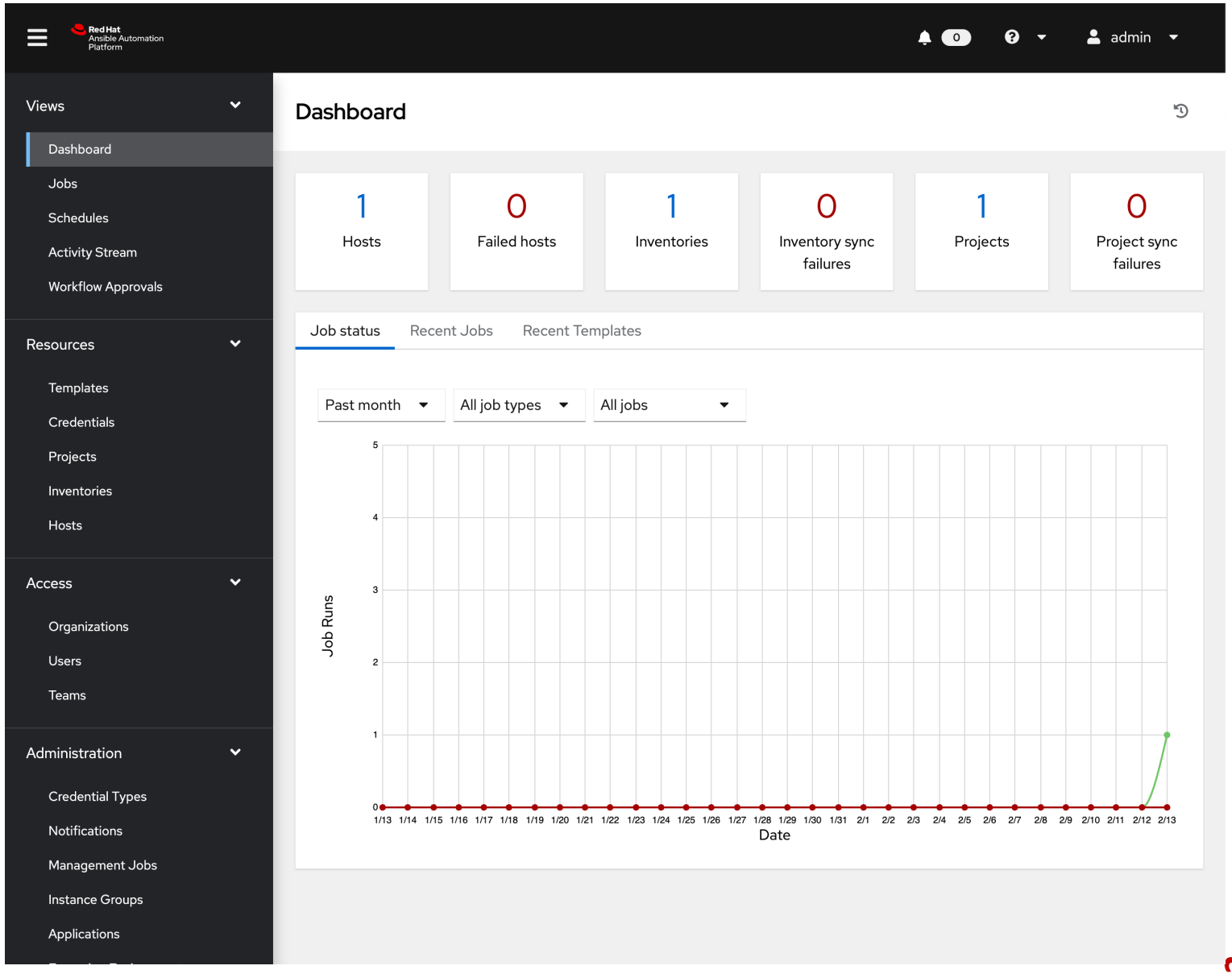
Two different installation packages are available for Automation Platform:

- 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



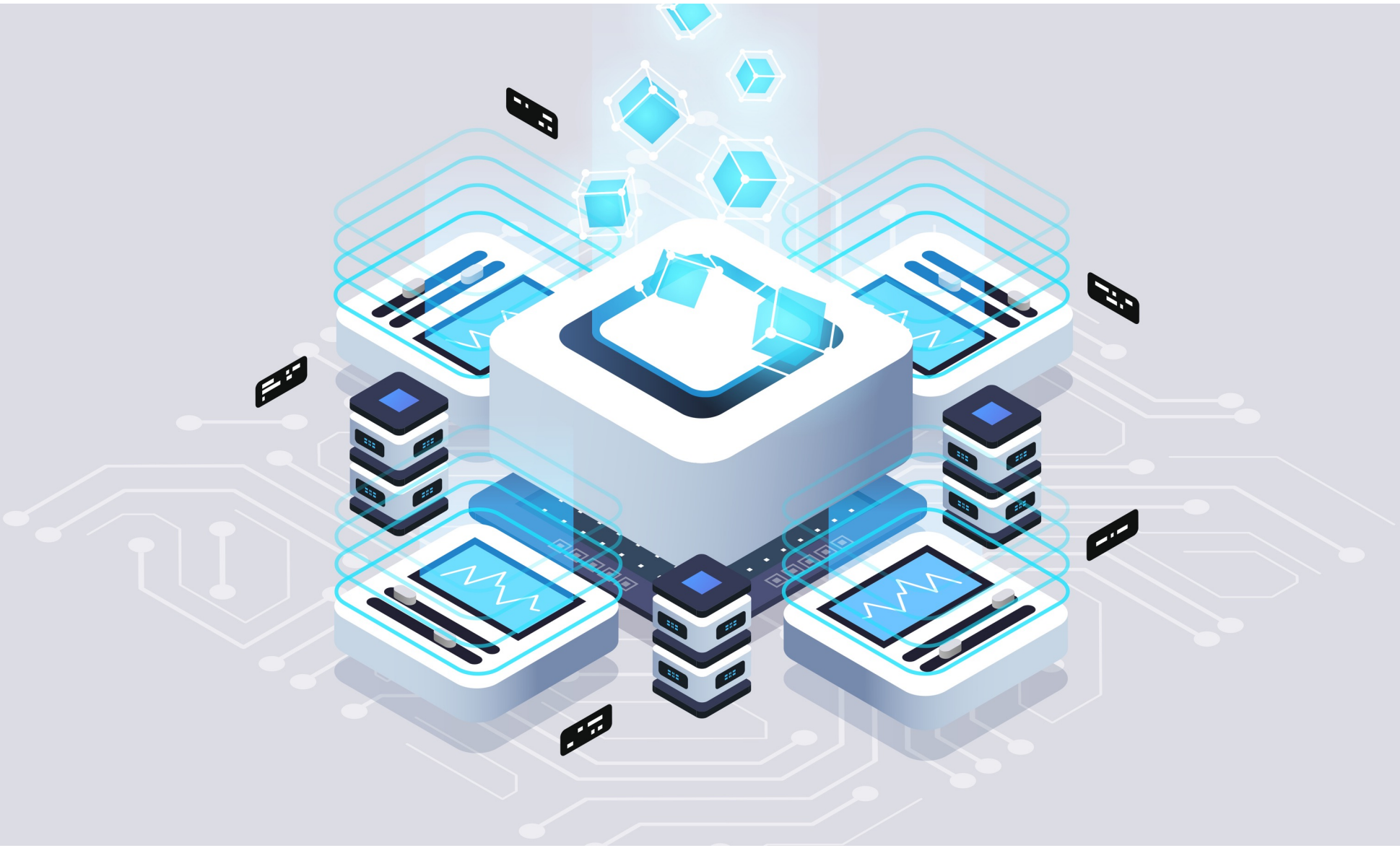
Dashboard Navigation

The dashboard contains links to common resources.

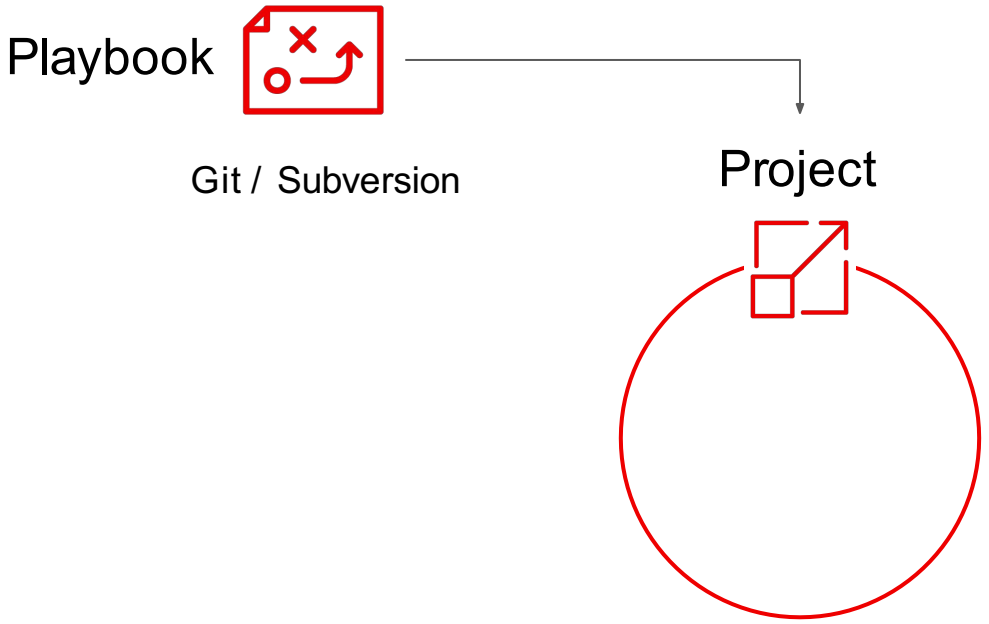
Organizations	Control what resources are visible to which users.
Teams	Groups of users that need to access the same resources
Users	User accounts
Jobs	A history of previous Ansible runs.
Templates	Prepared playbooks settings that can be “launched” to run a job.
Credentials	Authentication secrets for 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

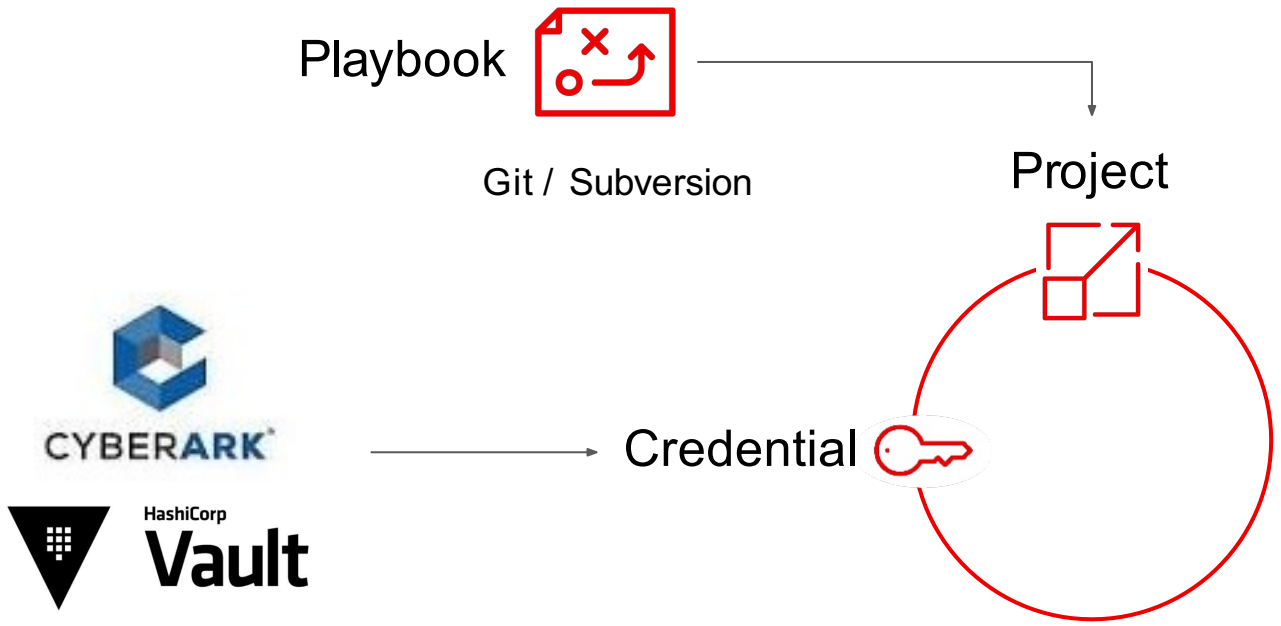
Automation Controller



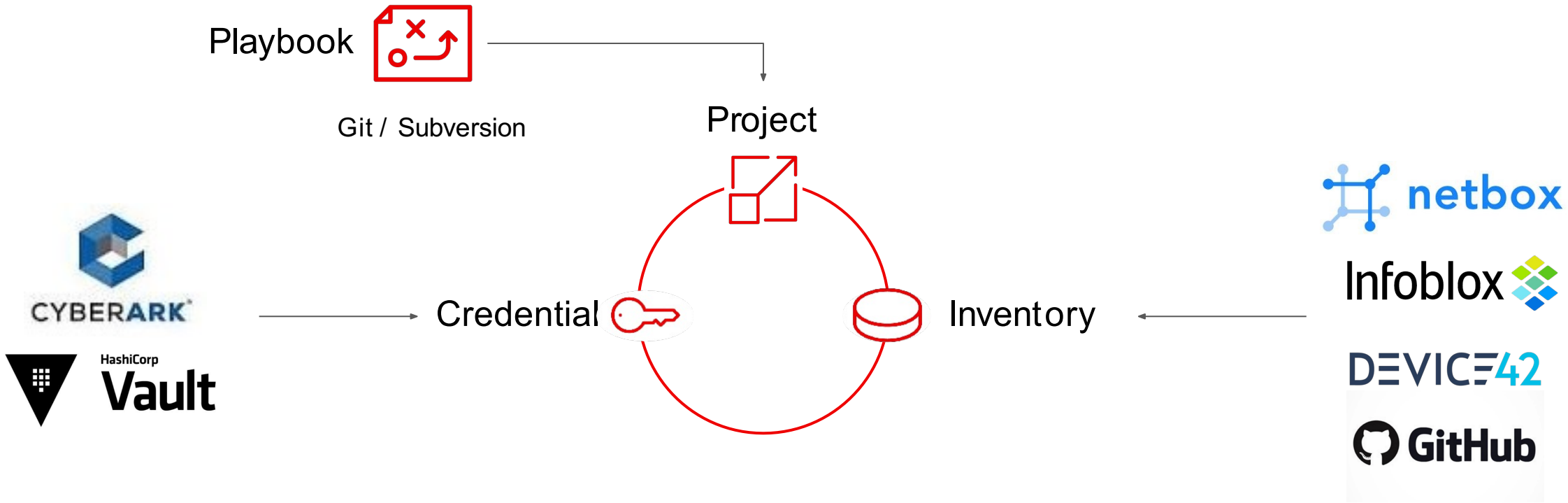
Anatomy of An Automation Job



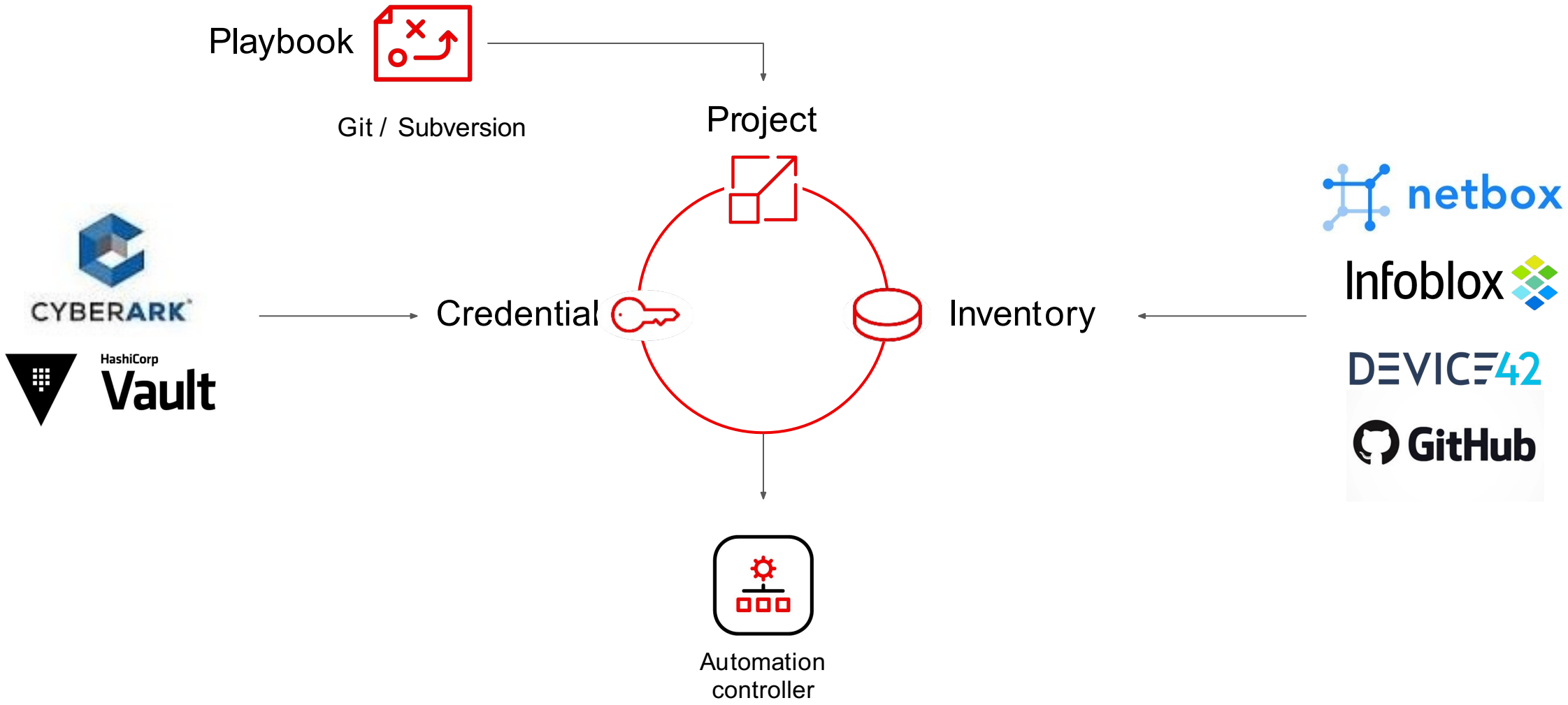
Anatomy of An Automation Job



Anatomy of An Automation Job

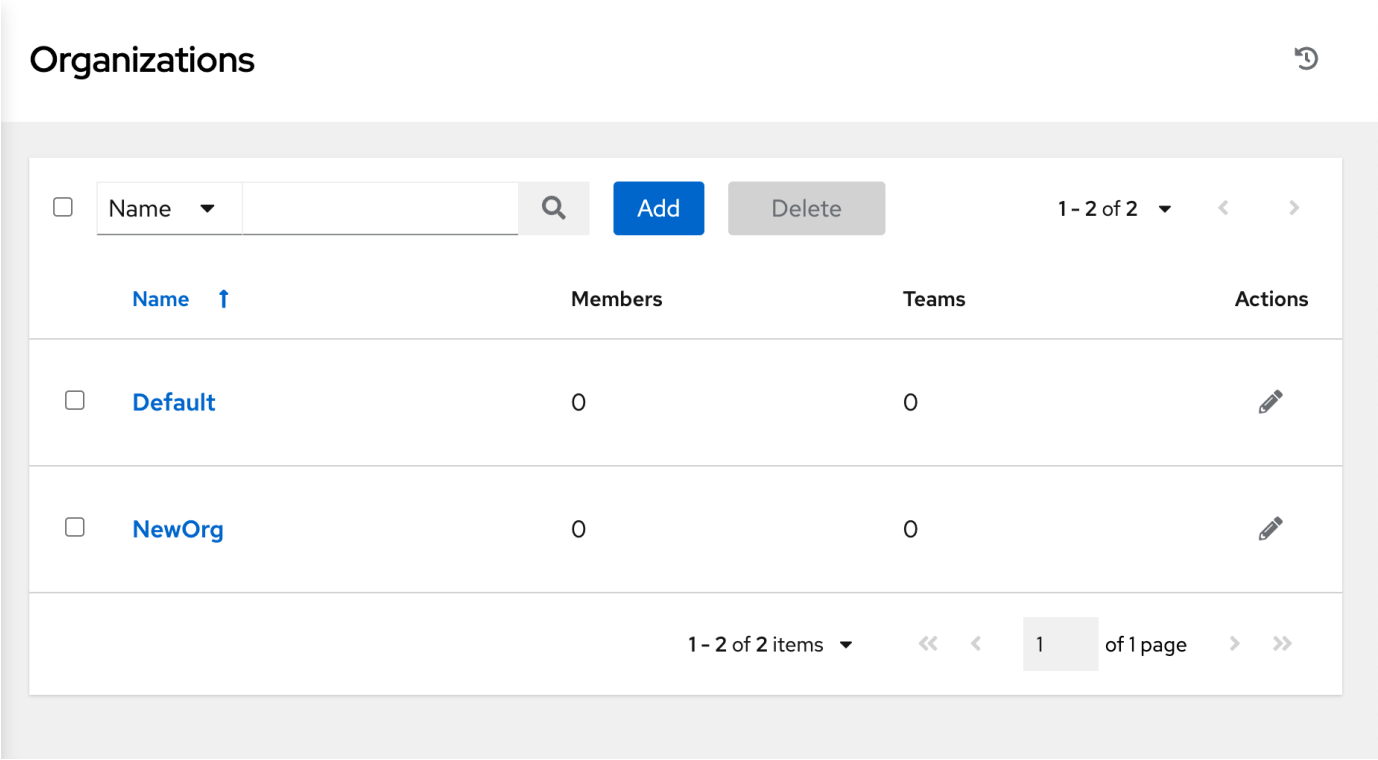


Anatomy of An Automation Job



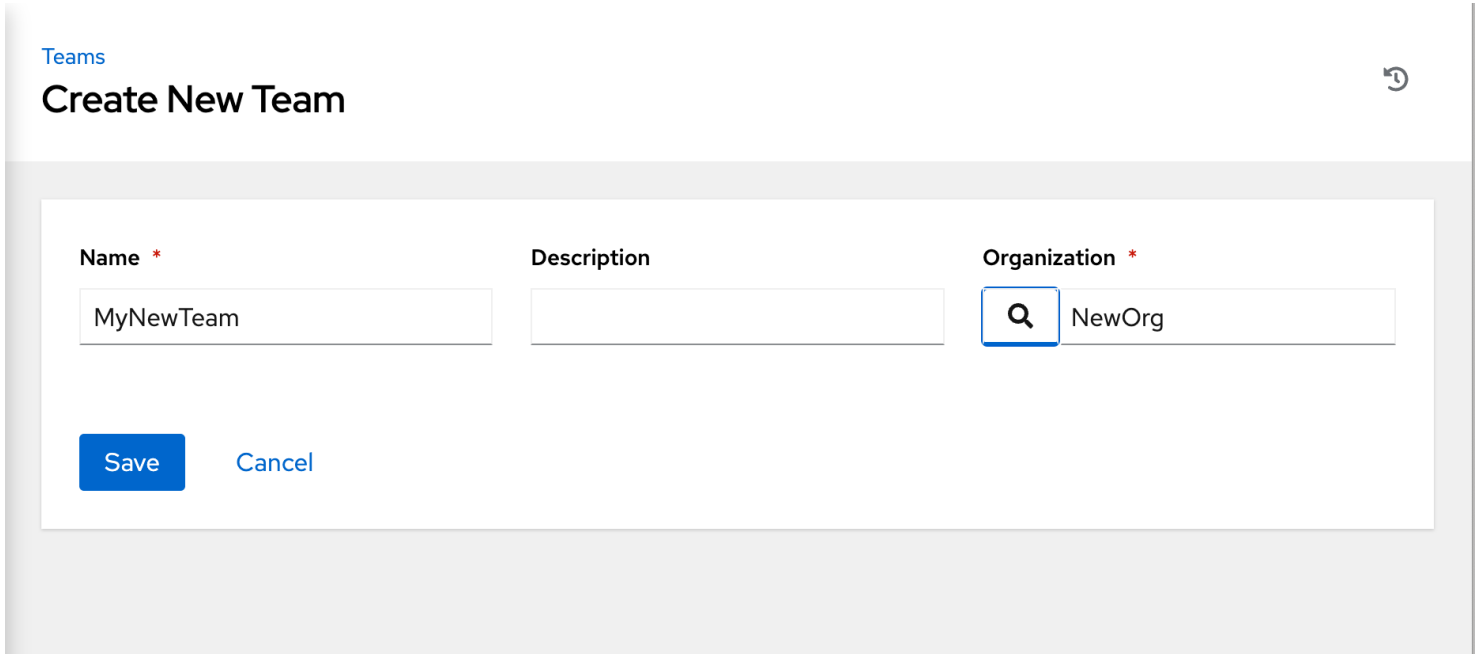
Organization

- 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.



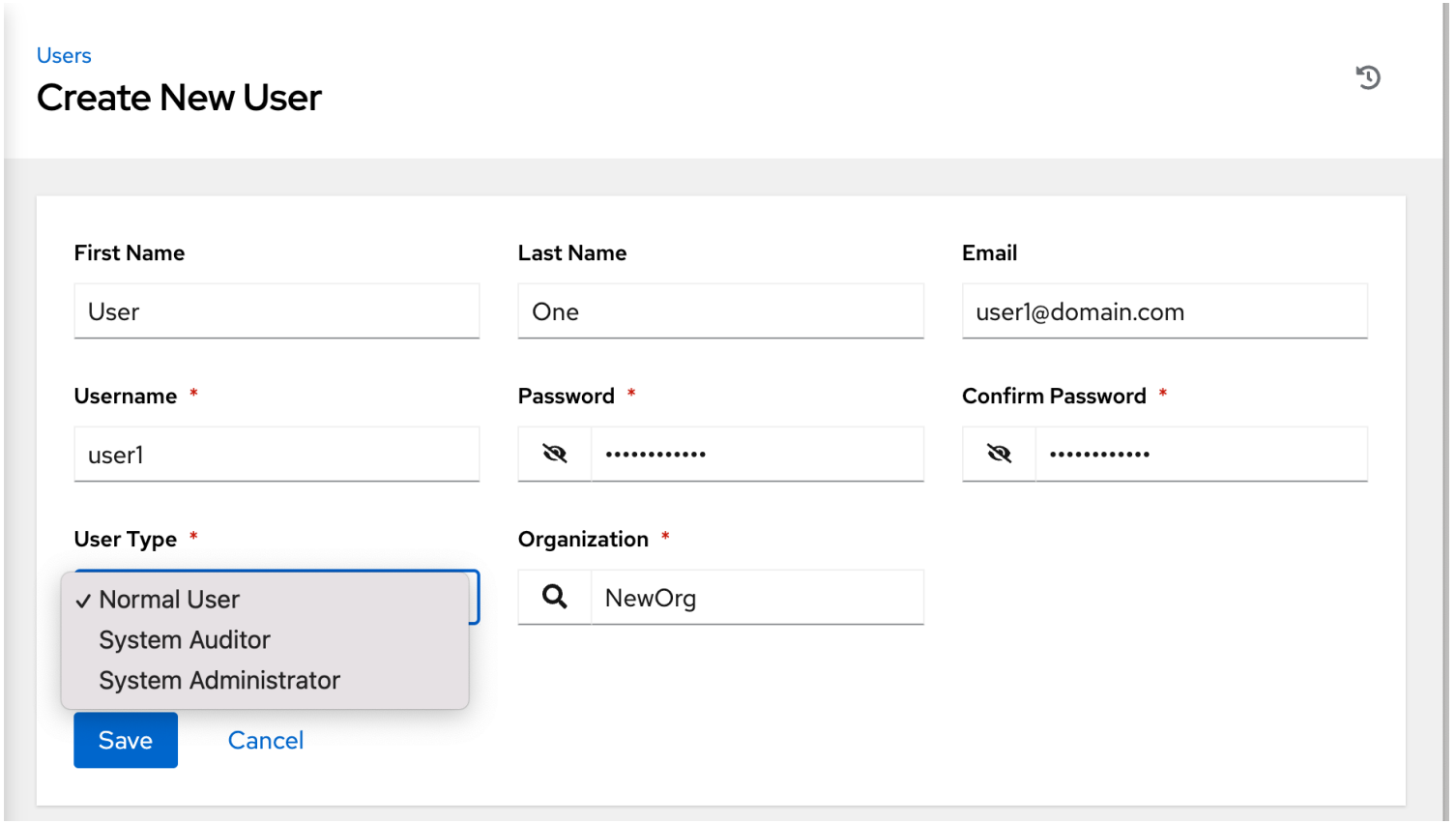
Team

- You can apply permissions at the team level.



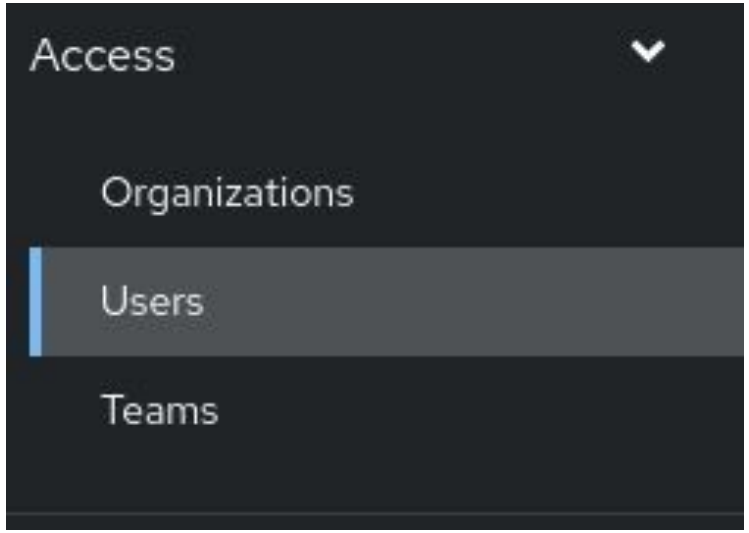
User Roles

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



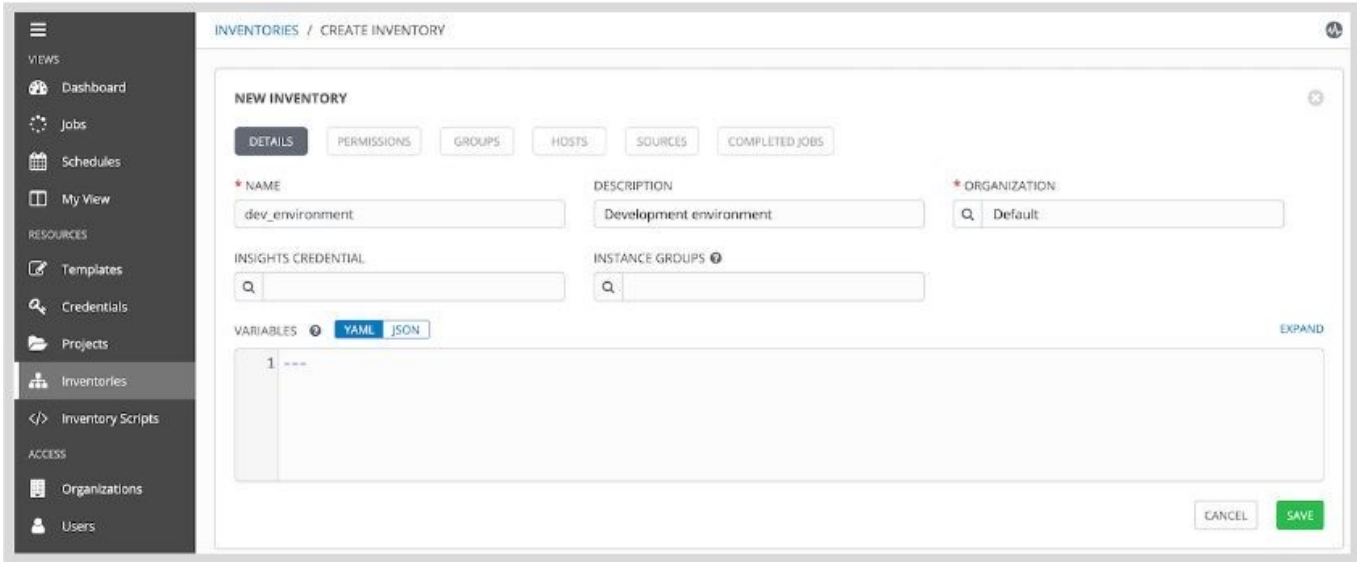
User Management

- 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.



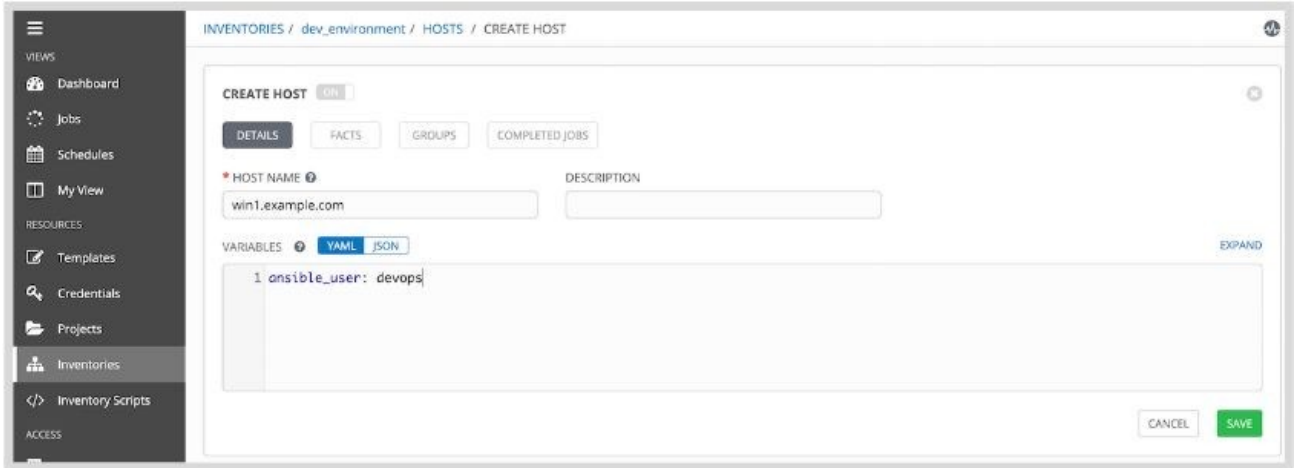
Inventory

- 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**.

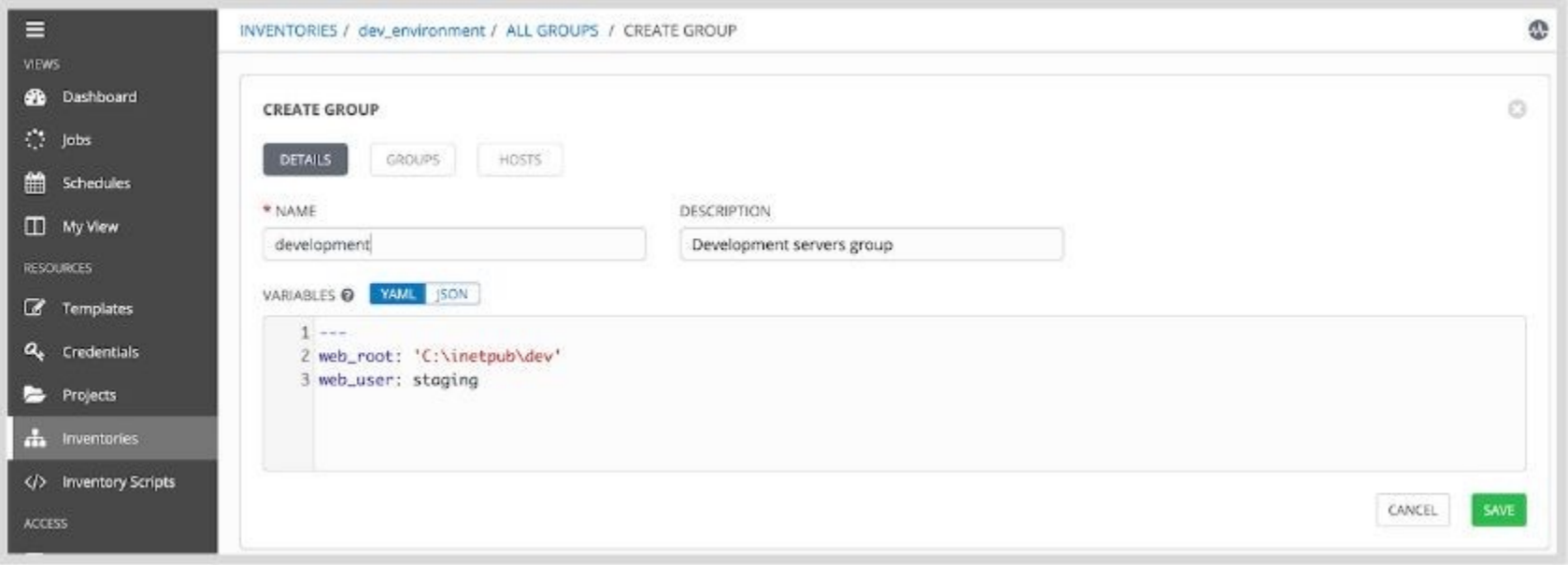


Inventory Groups

- 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**.

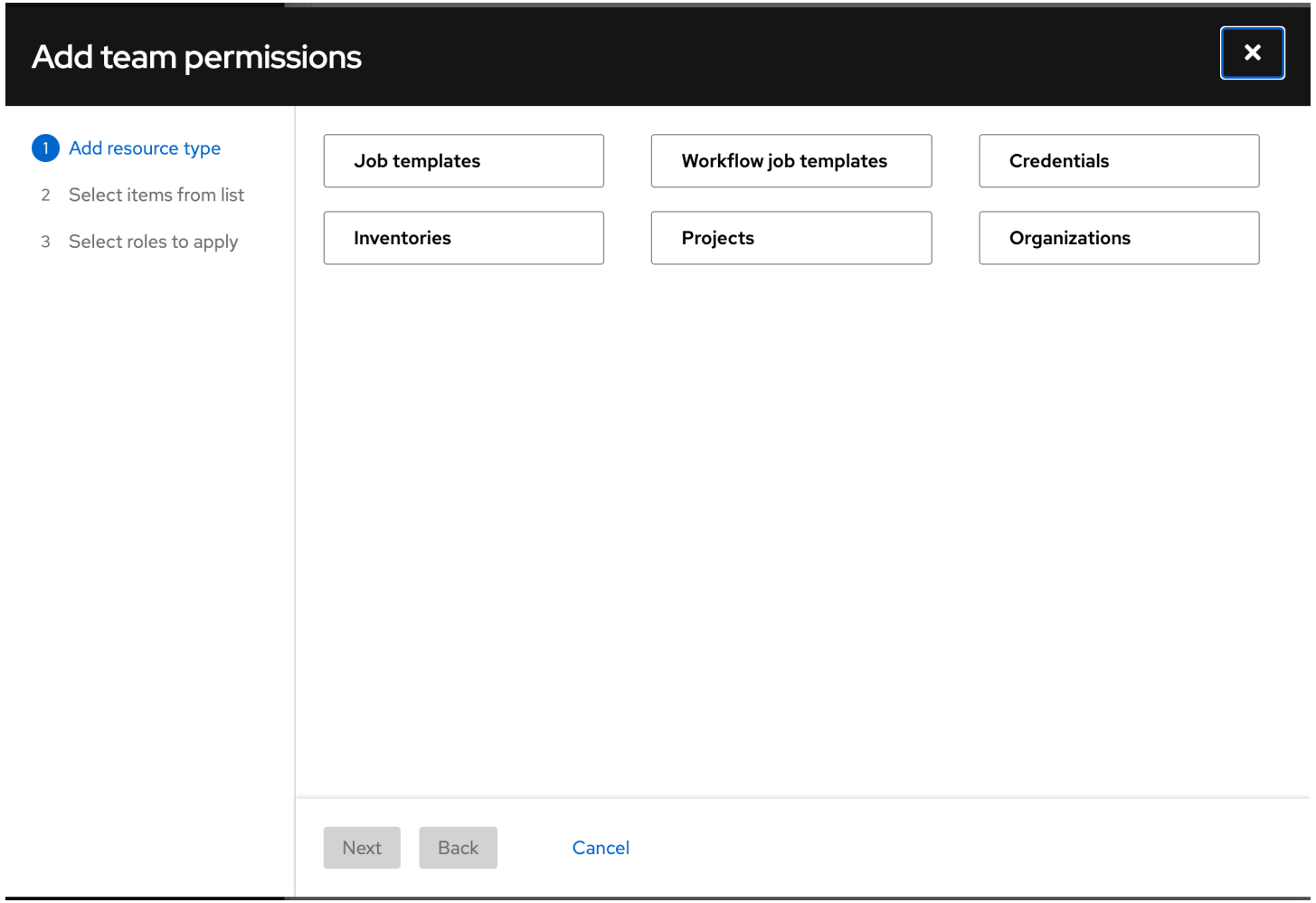


Inventory Groups

- 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

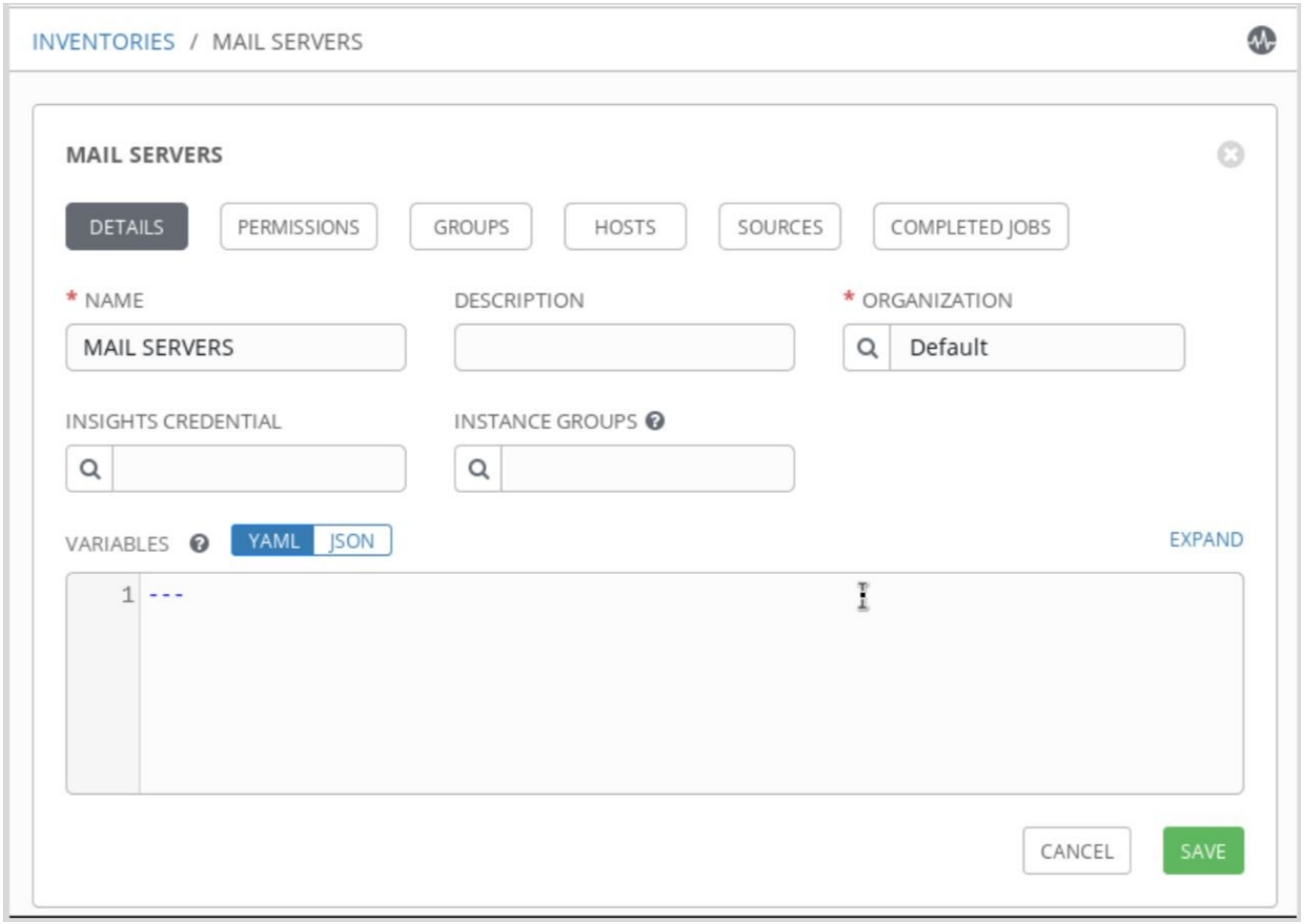
Role	Description
Admin	The inventory **Admin** role grants users full permissions over an inventory. These permissions include deletion and modification of the inventory. In addition, this role also grants permissions associated with the inventory roles **Use**, **Ad Hoc**, and **Update**.
Use	The inventory **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

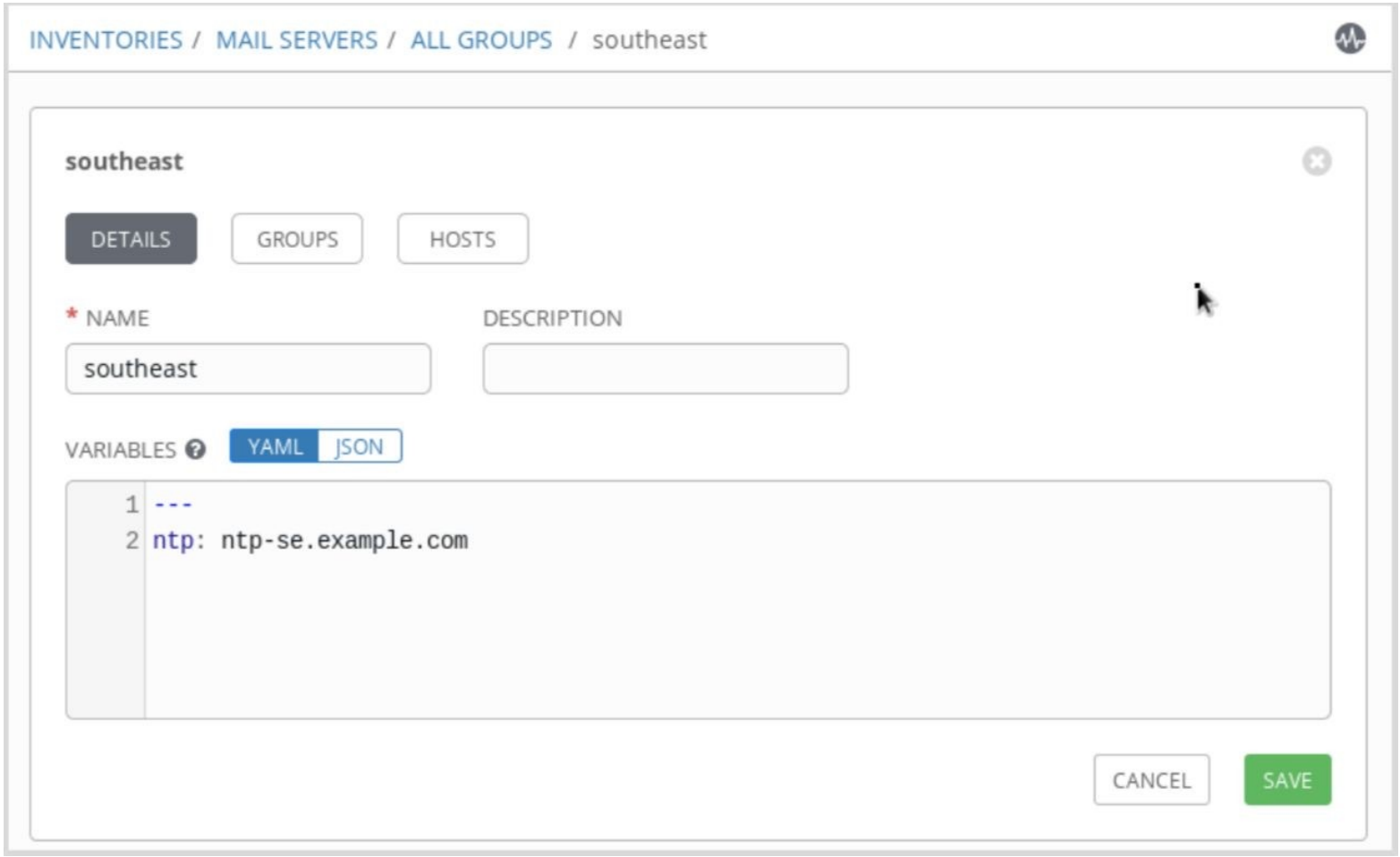
When you manage a static inventory in the Ansible Platform web UI, you may define inventory variables directly in the inventory objects.

- 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`**.

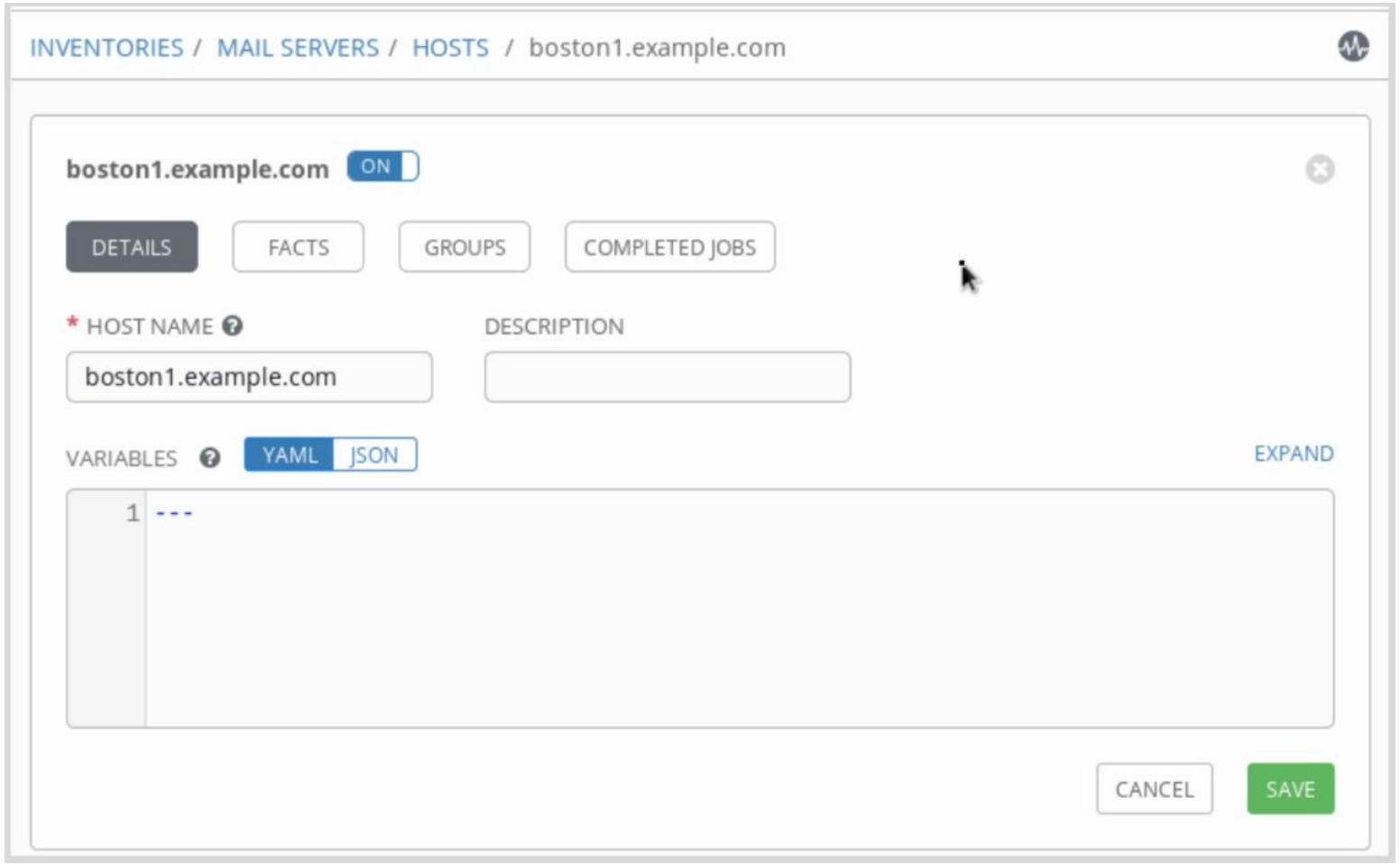
Inventory Variables



Inventory Group Variables



Inventory Host Variables

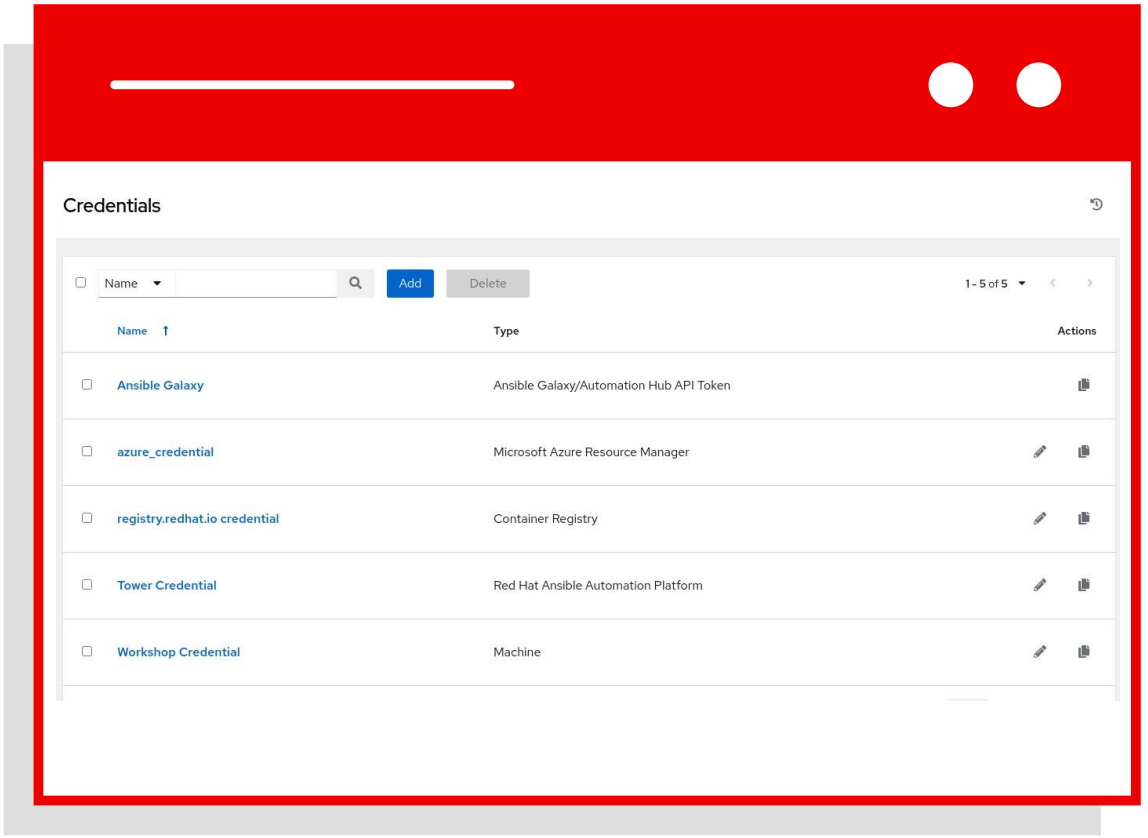


Credentials

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

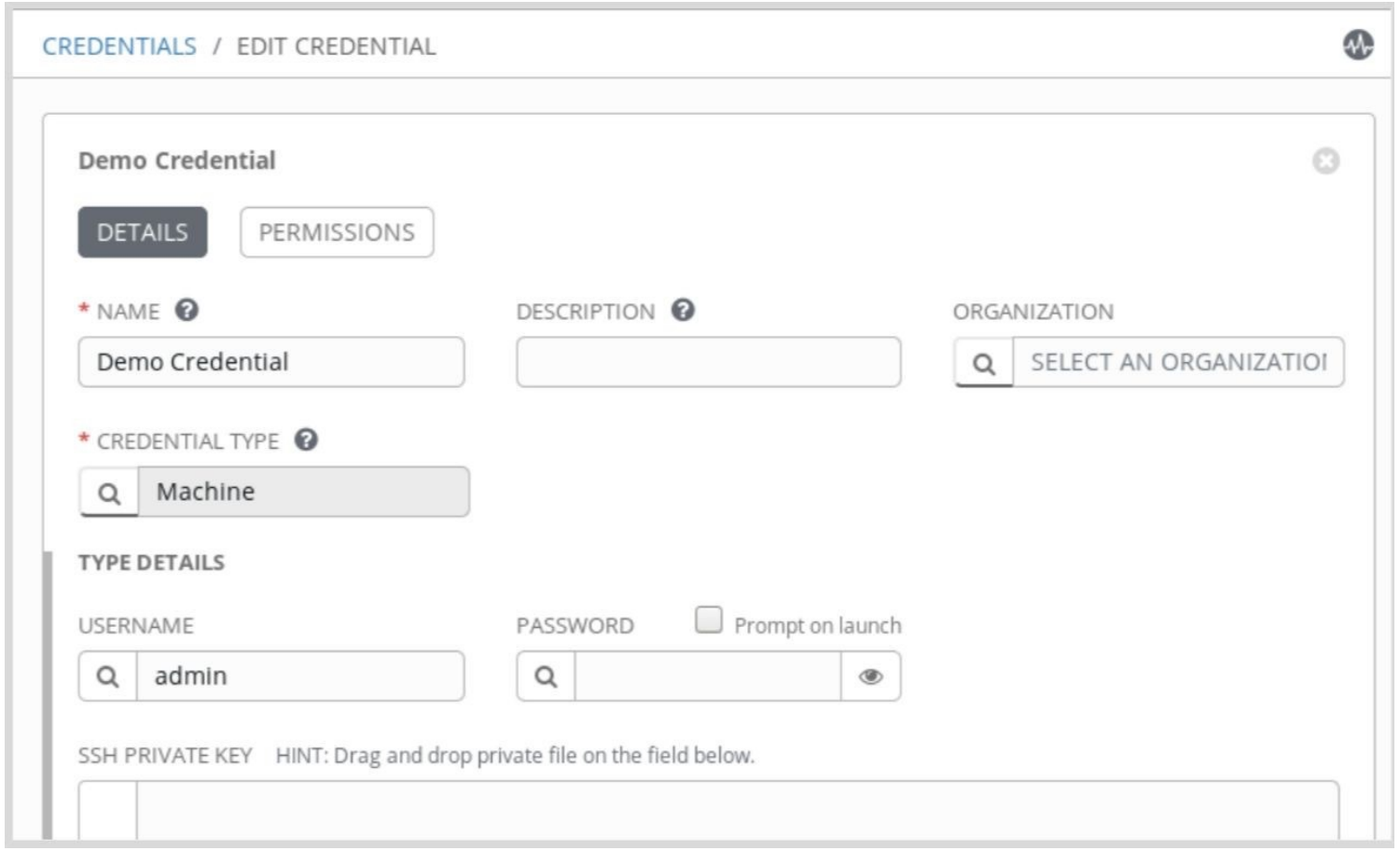
- Connecting to remote machines to run jobs
- Syncing with inventory sources
- Importing project content from version control systems
- Connecting to and managing network devices

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**.

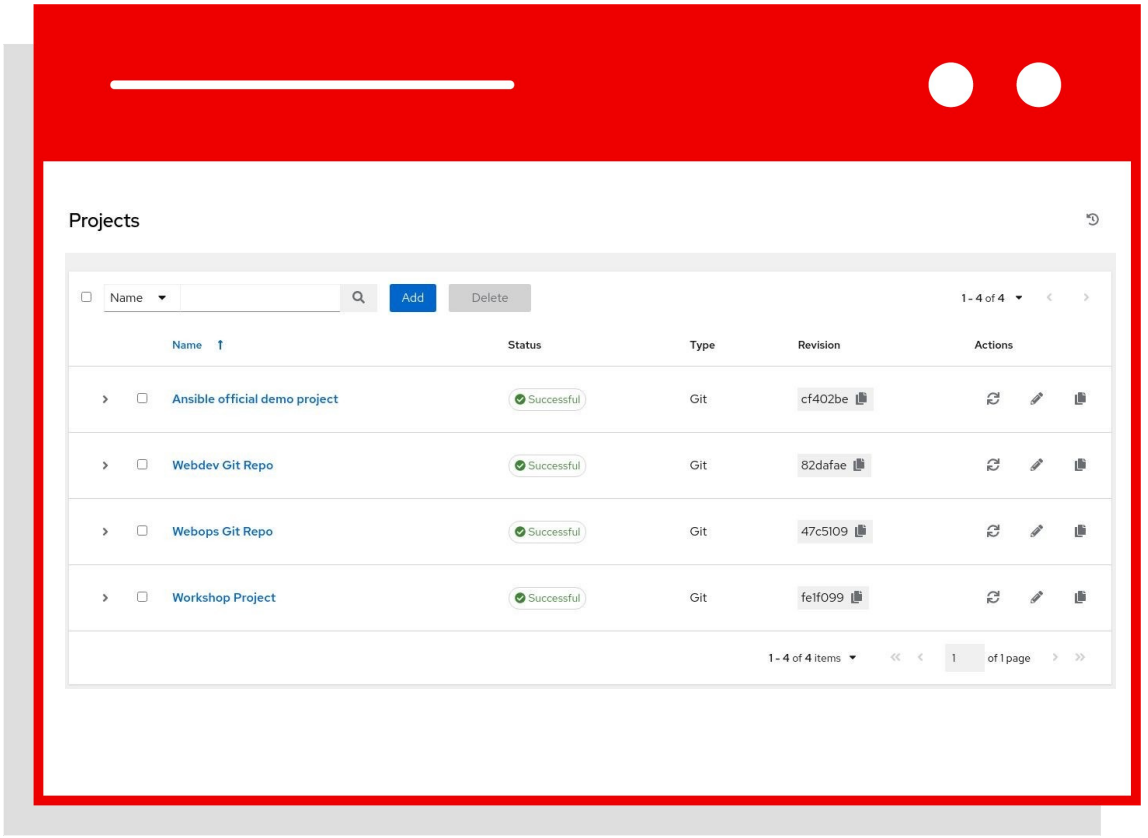


Lab: AAP Inventories, credentials, and ad-hoc commands

PROJECT

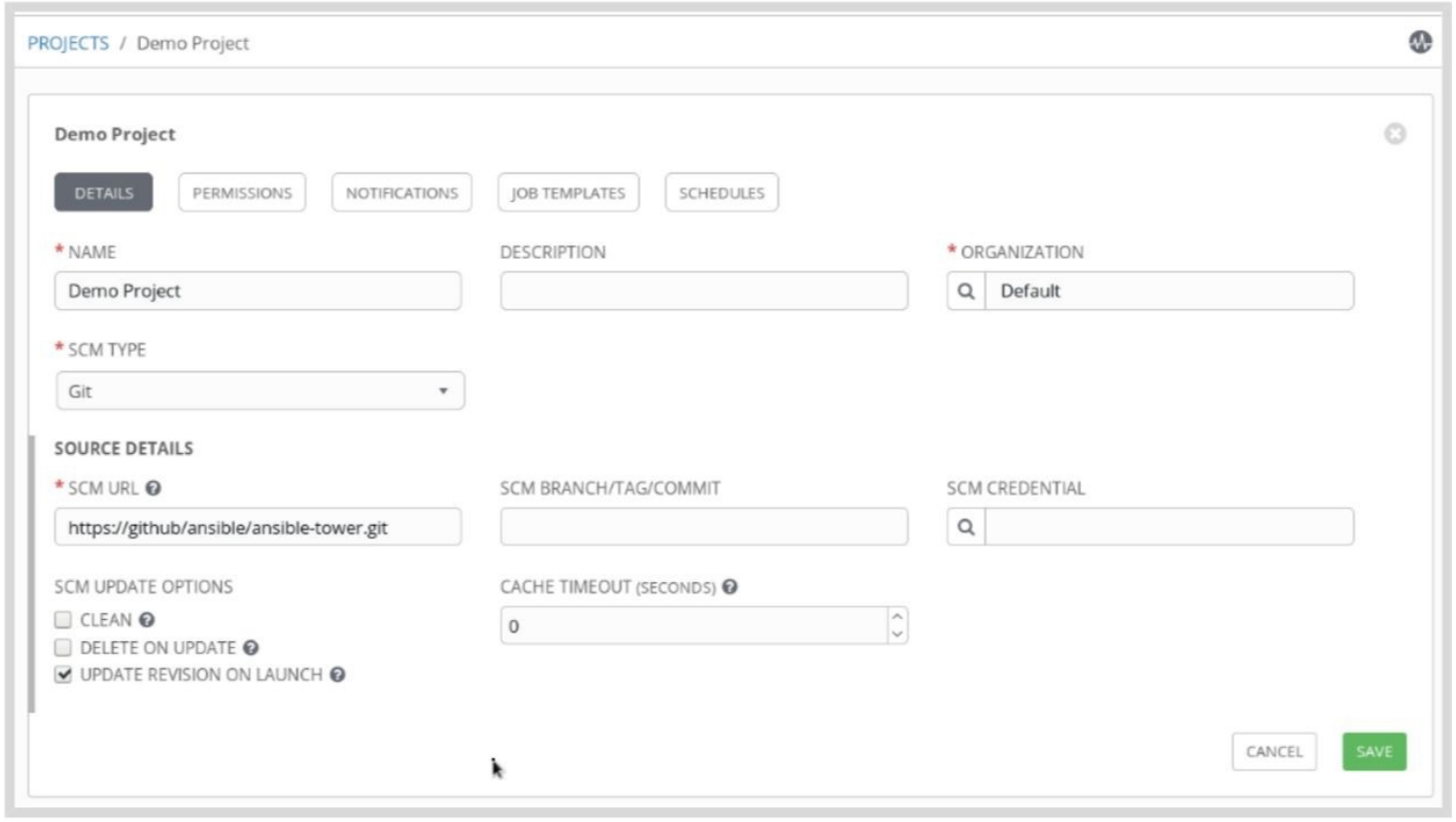
A project is a logical collection of Ansible Playbooks, represented in Ansible Automation Controller.

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.



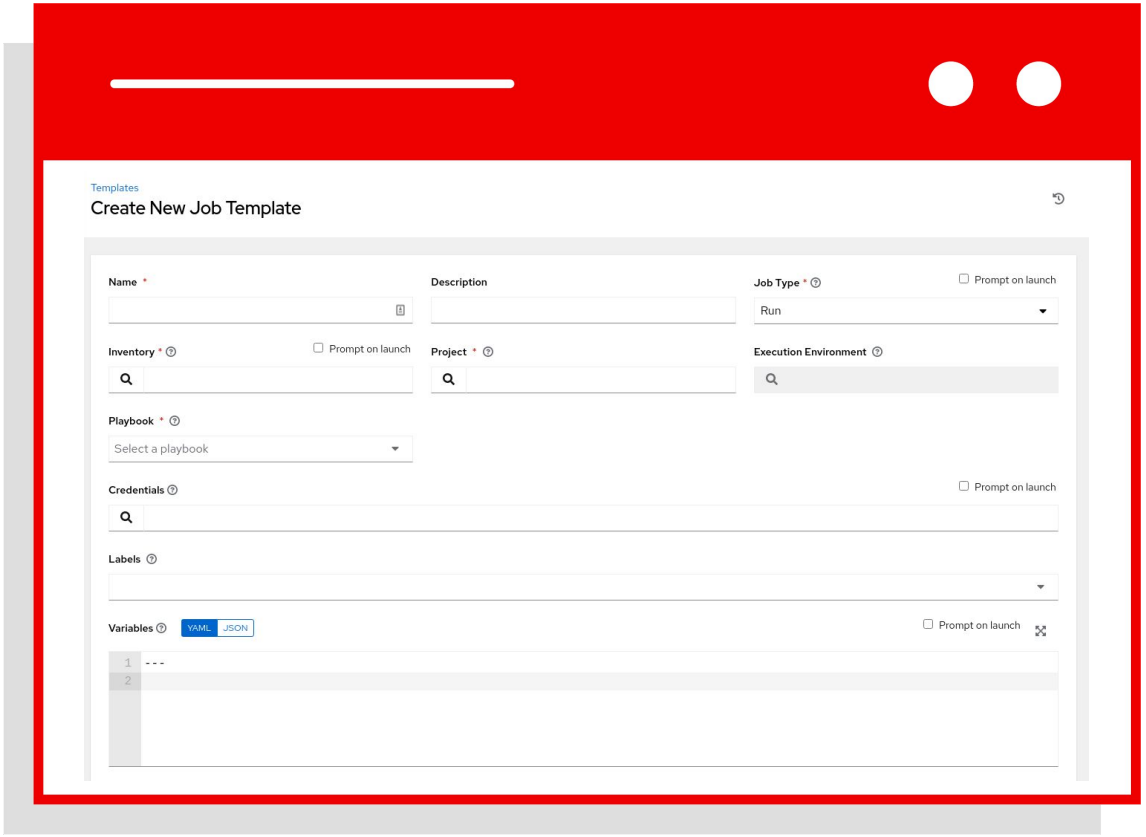
JOB TEMPLATES

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.

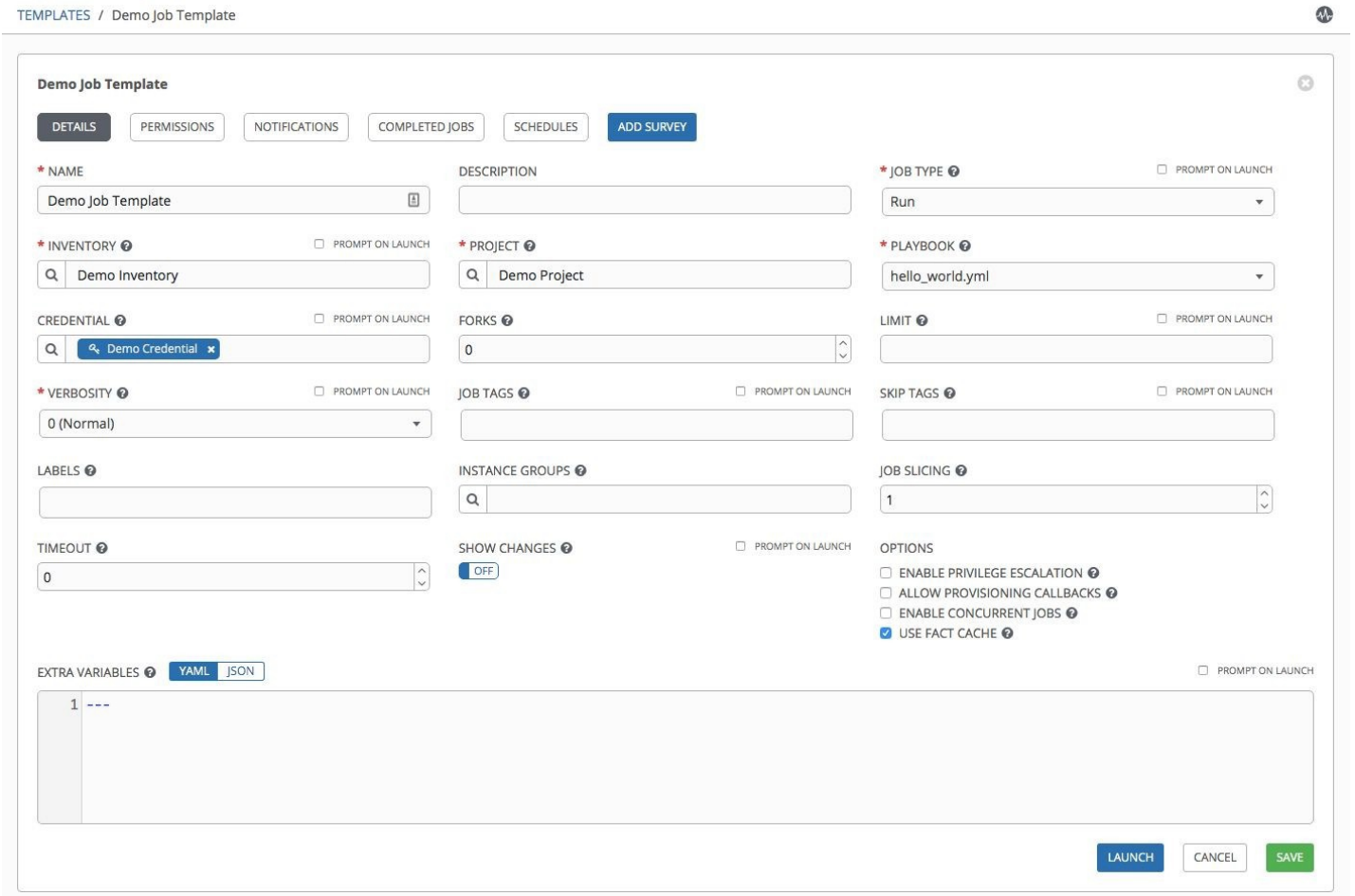
A **Job Template** requires:

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



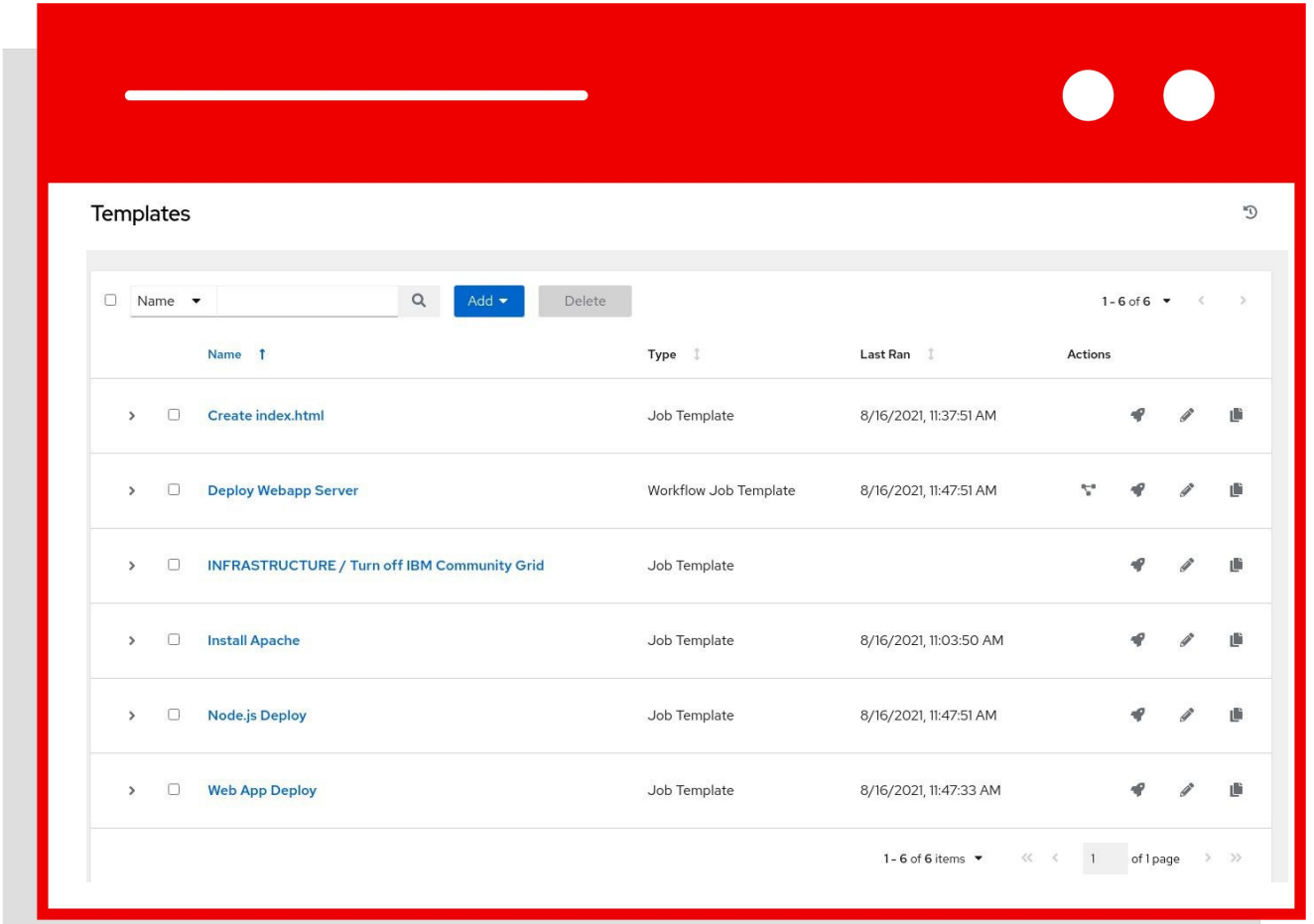
JOB TEMPLATES

- 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.



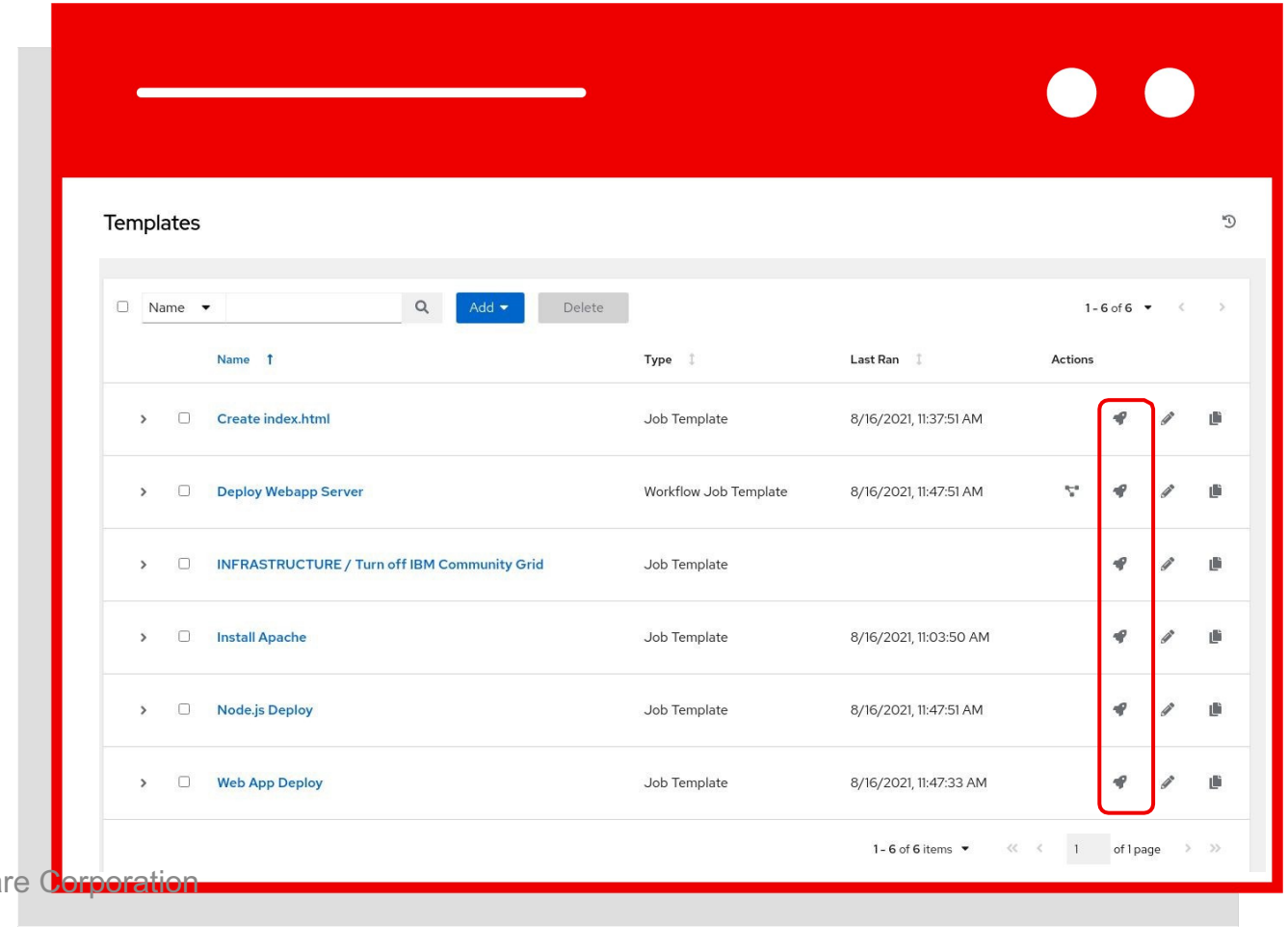
EXPANDING JOB TEMPLATES

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



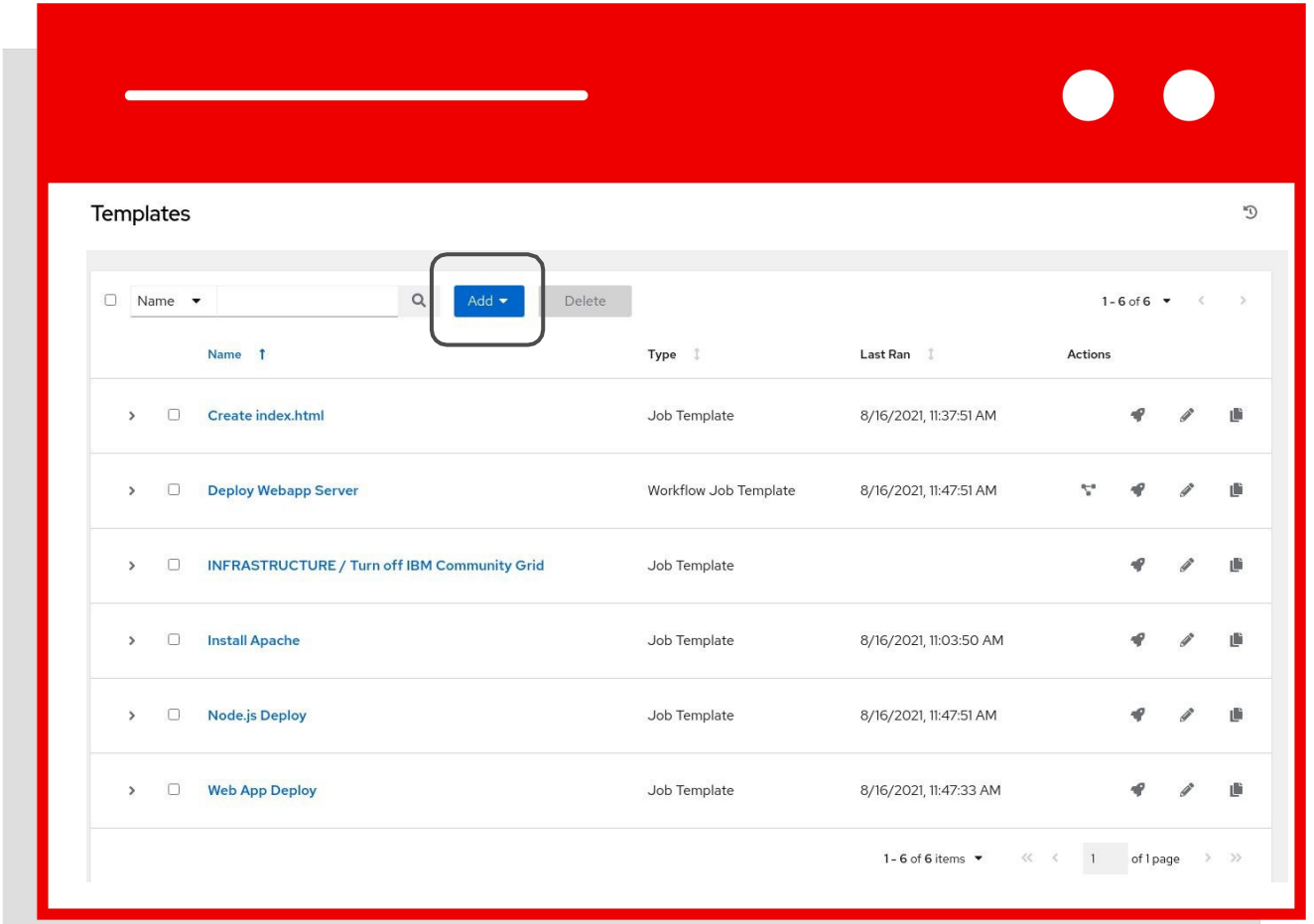
EXECUTING AN EXISTING JOB

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



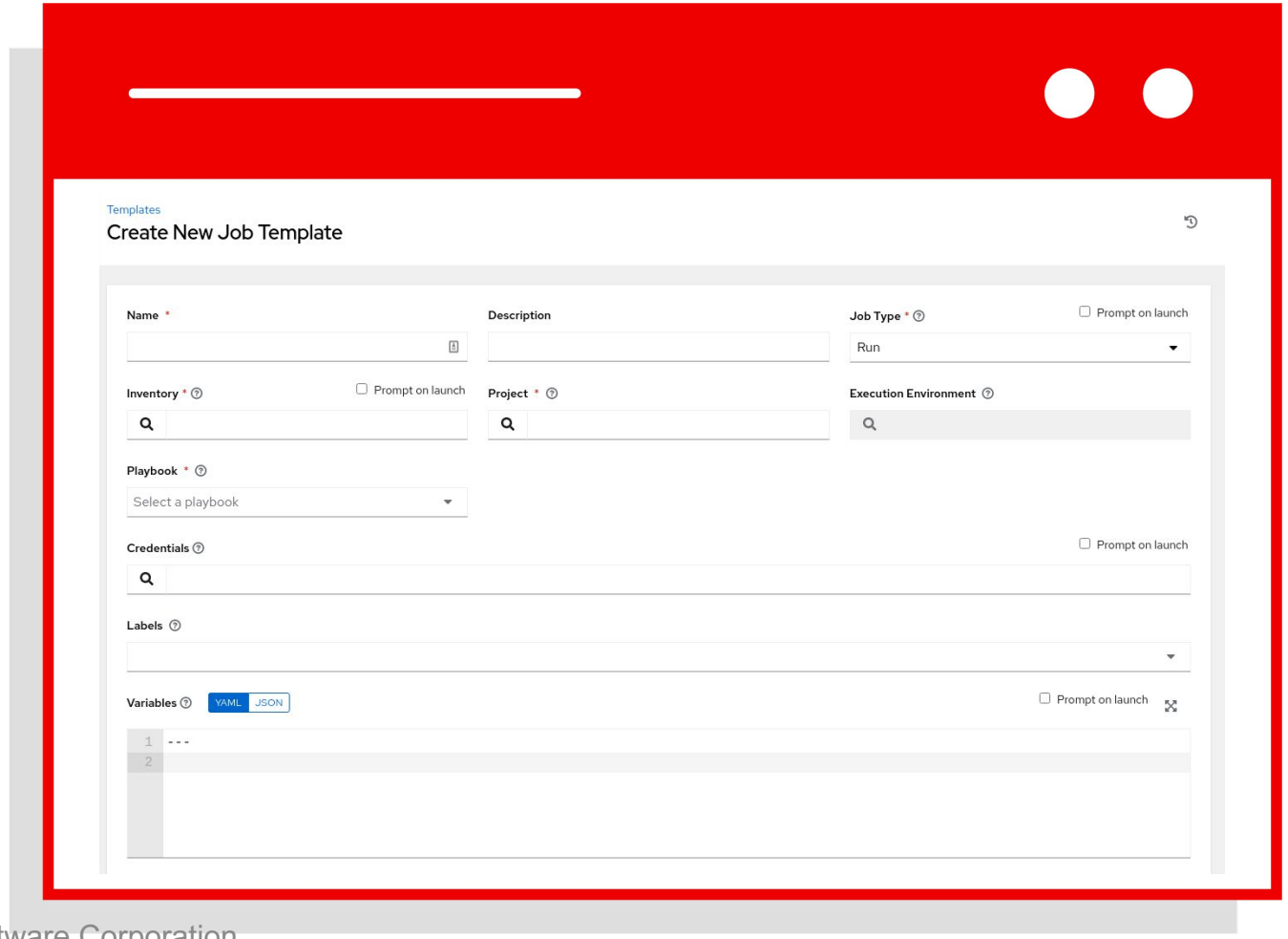
CREATING A NEW JOB TEMPLATE (1/2)

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 .

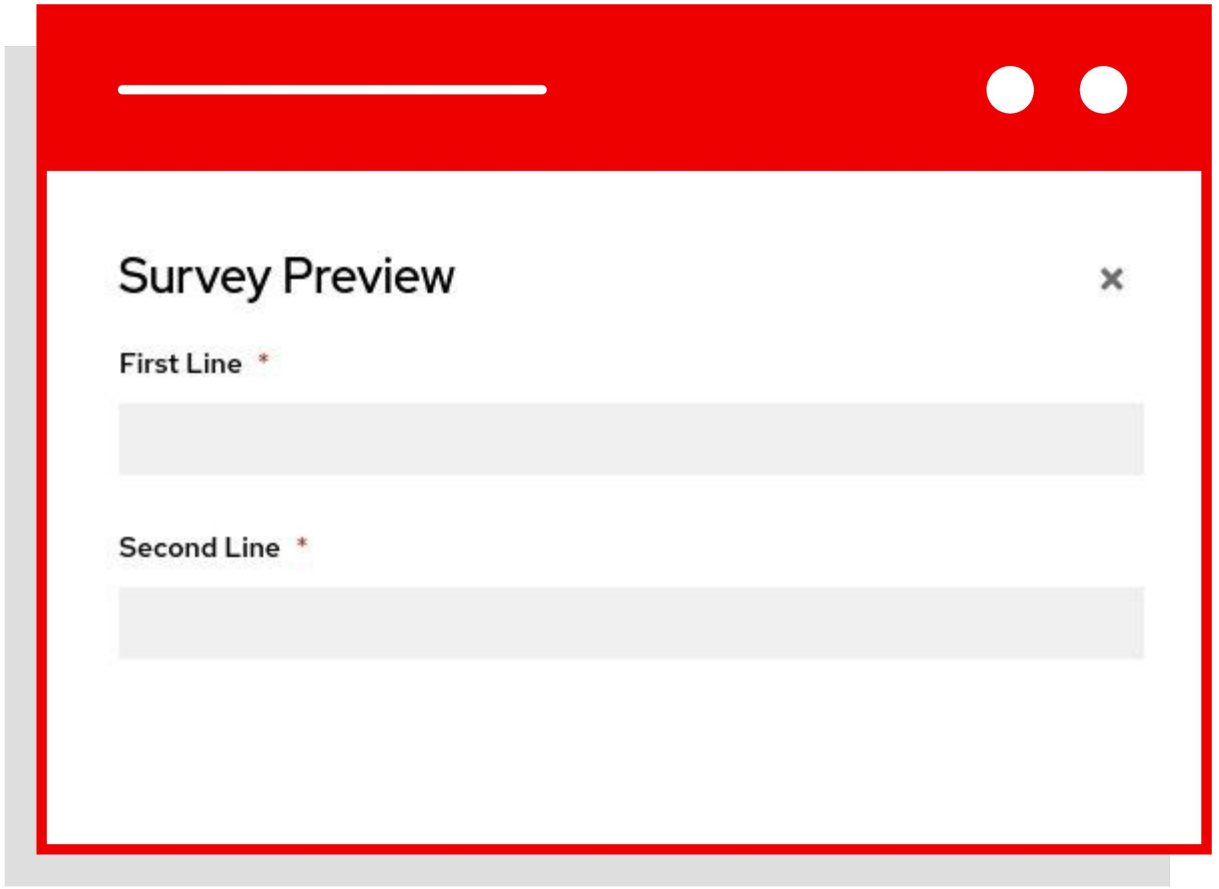


SURVEYS

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.

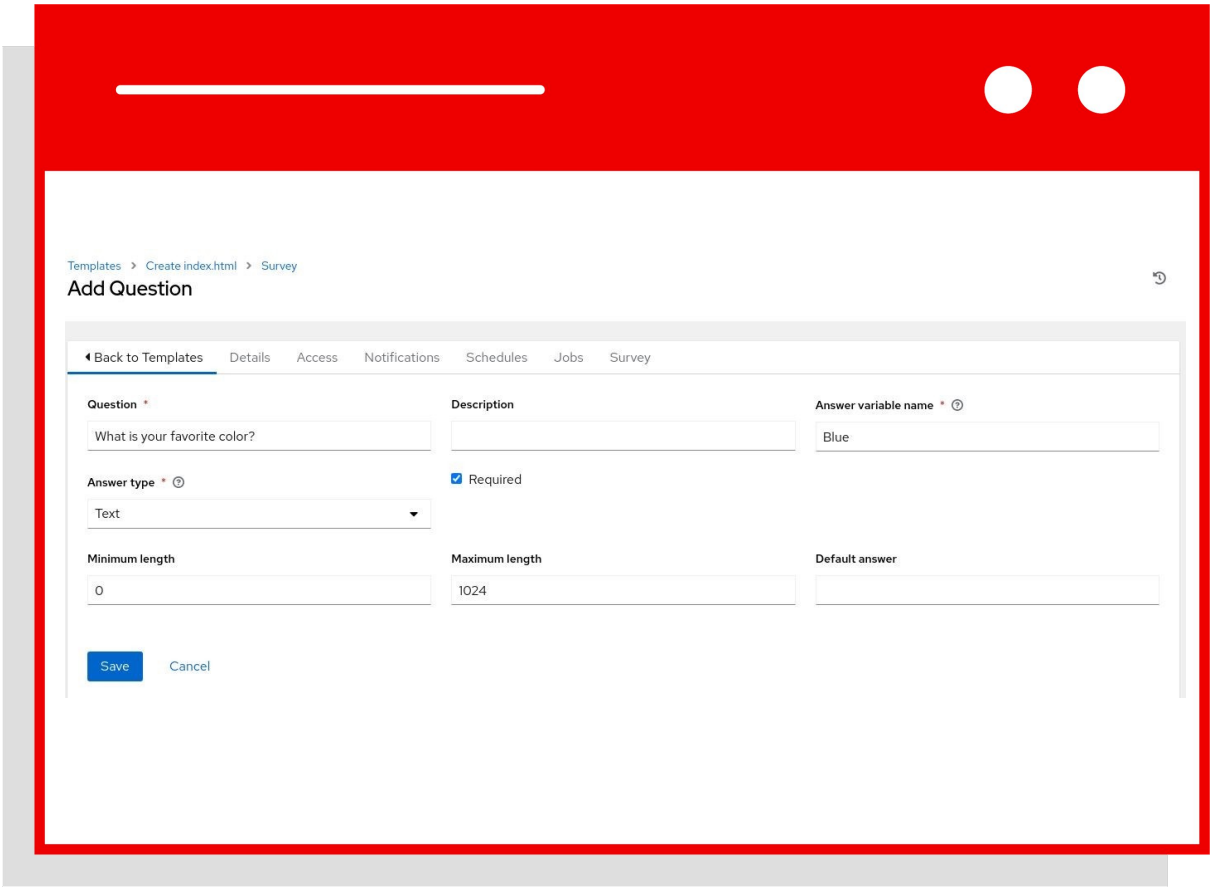
Combine that with Controller's role-based access control, and you can build simple, easy self-service for your users.



CREATING A SURVEY (1/2)

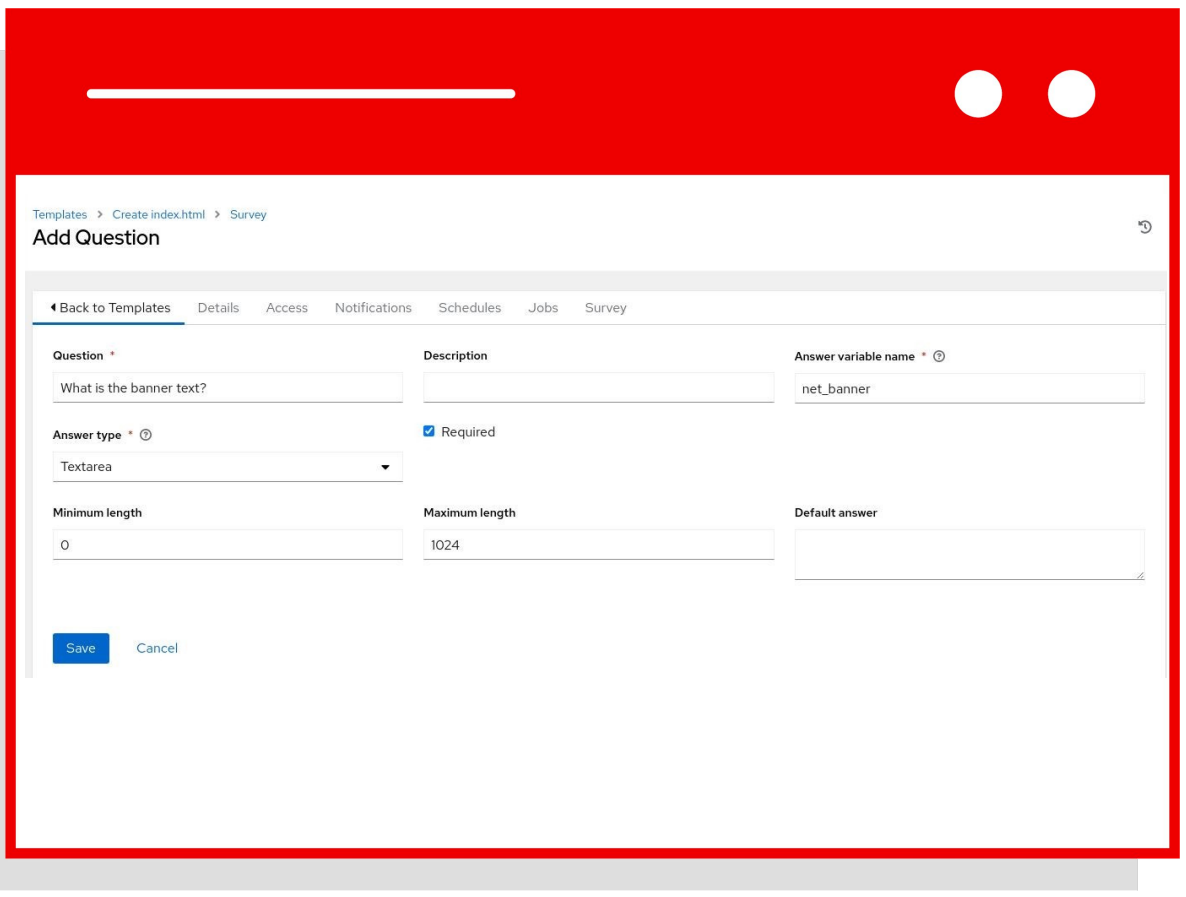
Once a Job Template is saved, the Survey menu will have an **Add Button**

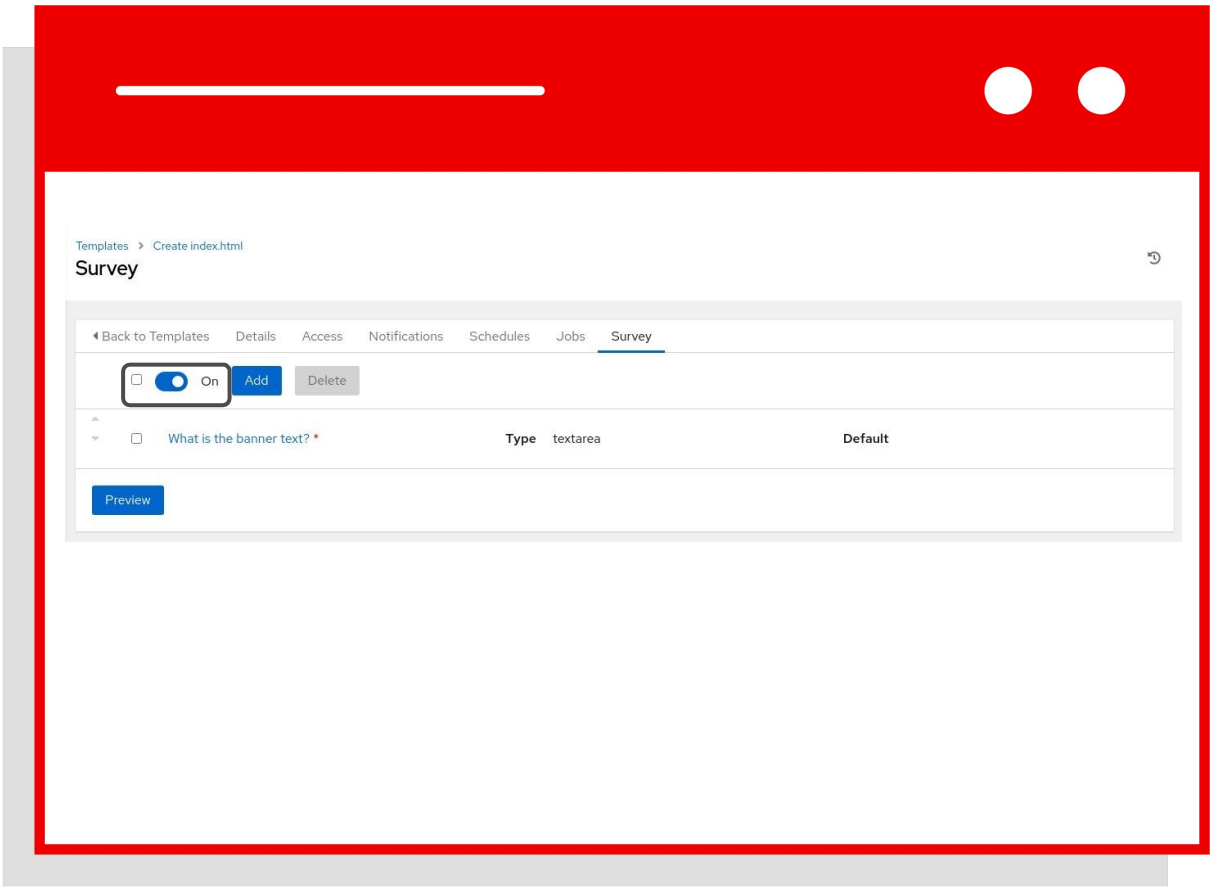
Click the button to open the Add Survey window.



CREATING A SURVEY (2/2)

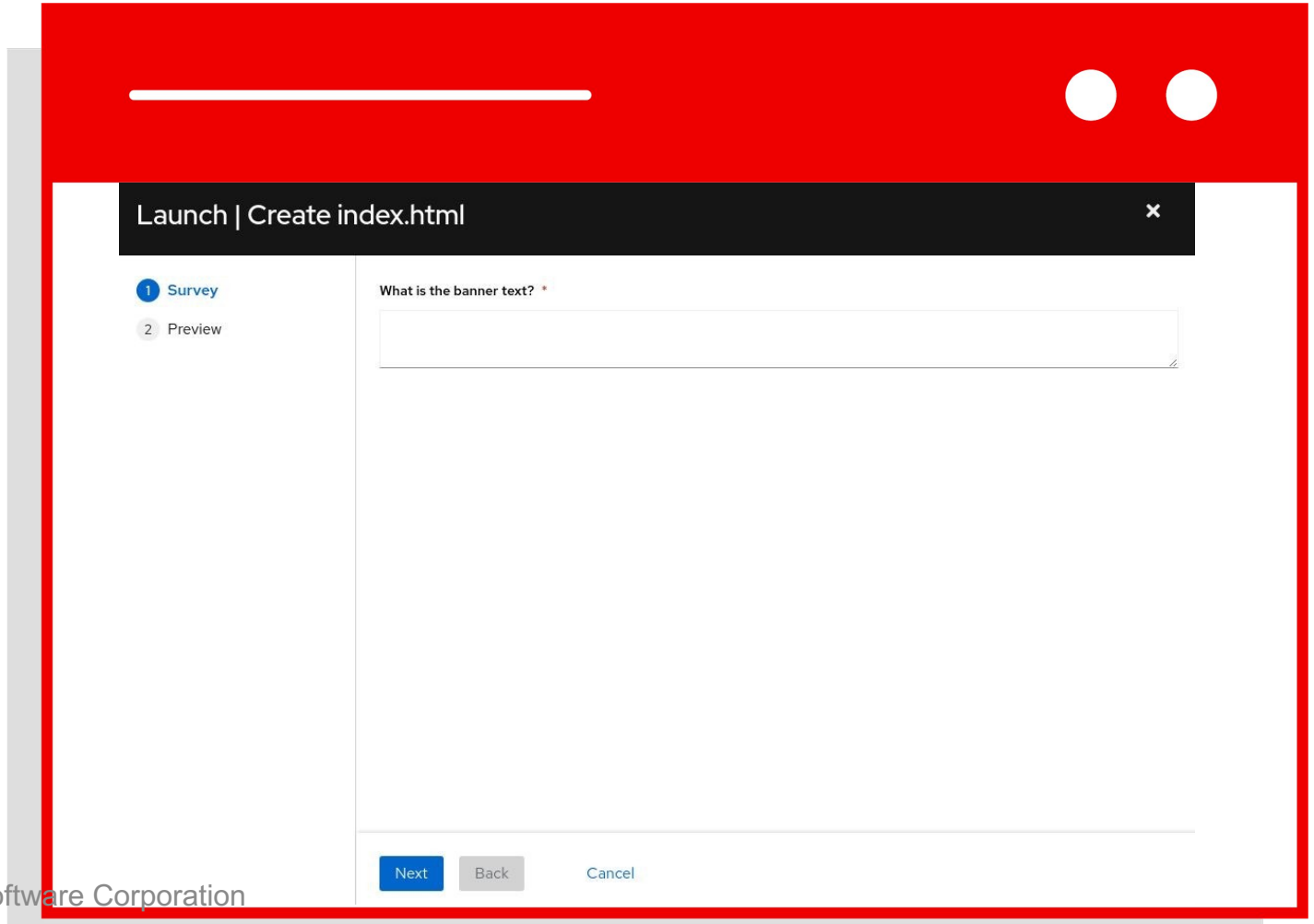
The Add Survey window allows the Job Template to prompt users for one or more questions. The answers provided become variables for use in the Ansible Playbook.





USING A SURVEY

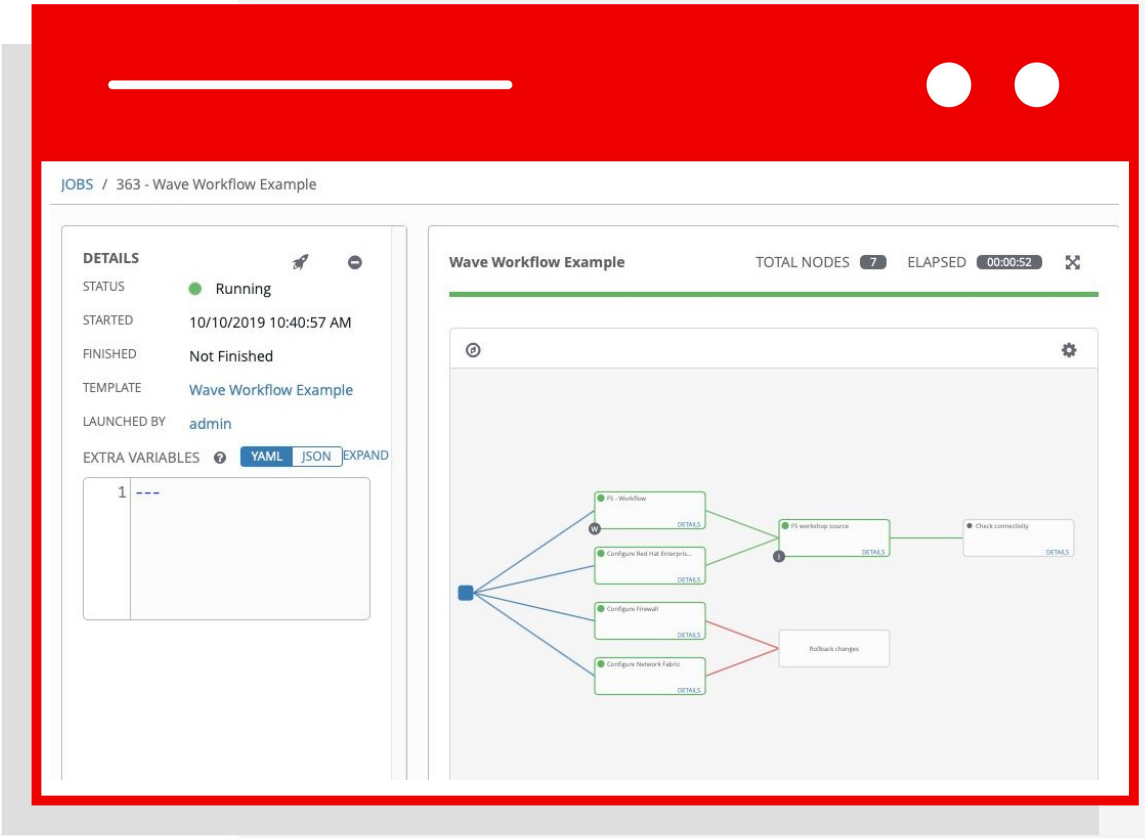
When launching a job, the user will now be prompted with the Survey. The user can be required to fill out the Survey before the Job Template will execute.



WORKFLOWS

Combine automation to create something bigger

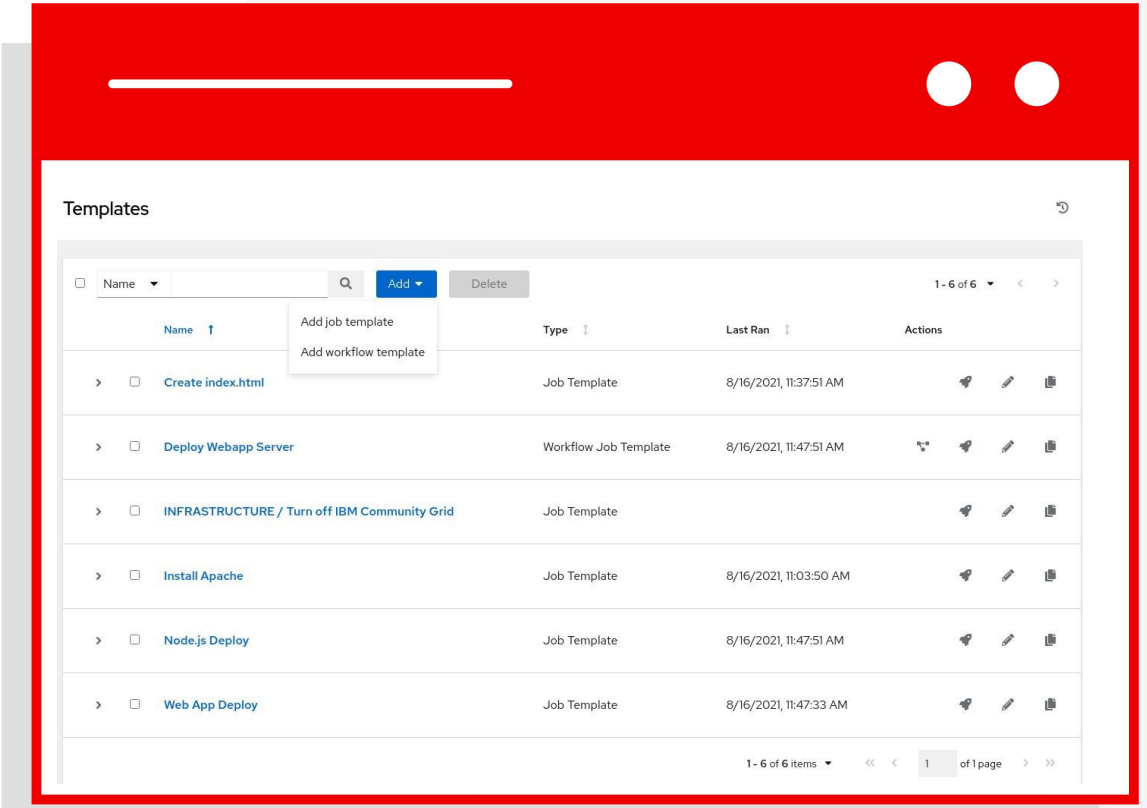
- Workflows enable the creation of powerful holistic automation, chaining together multiple pieces of automation and events.
- Simple logic inside these workflows can trigger automation depending on the success or failure of previous steps.



ADDING A NEW TEMPLATE

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

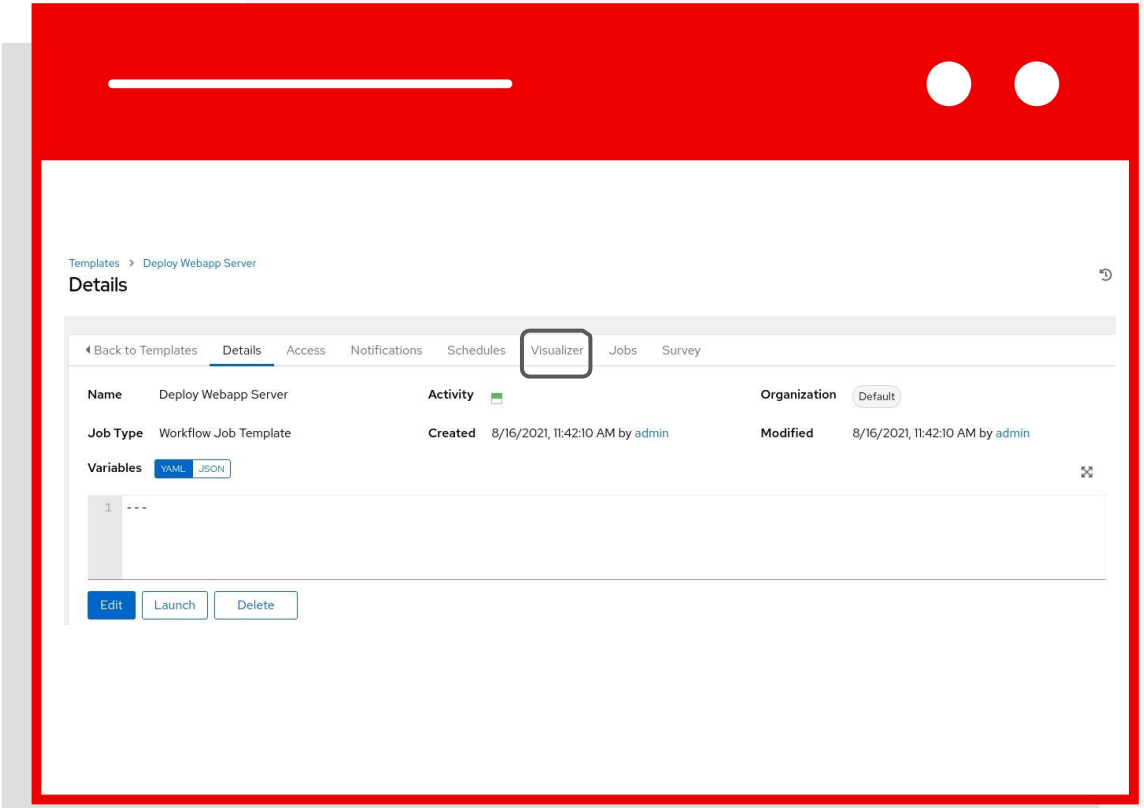
  This time select the **Add workflow template**



CREATING THE WORKFLOW

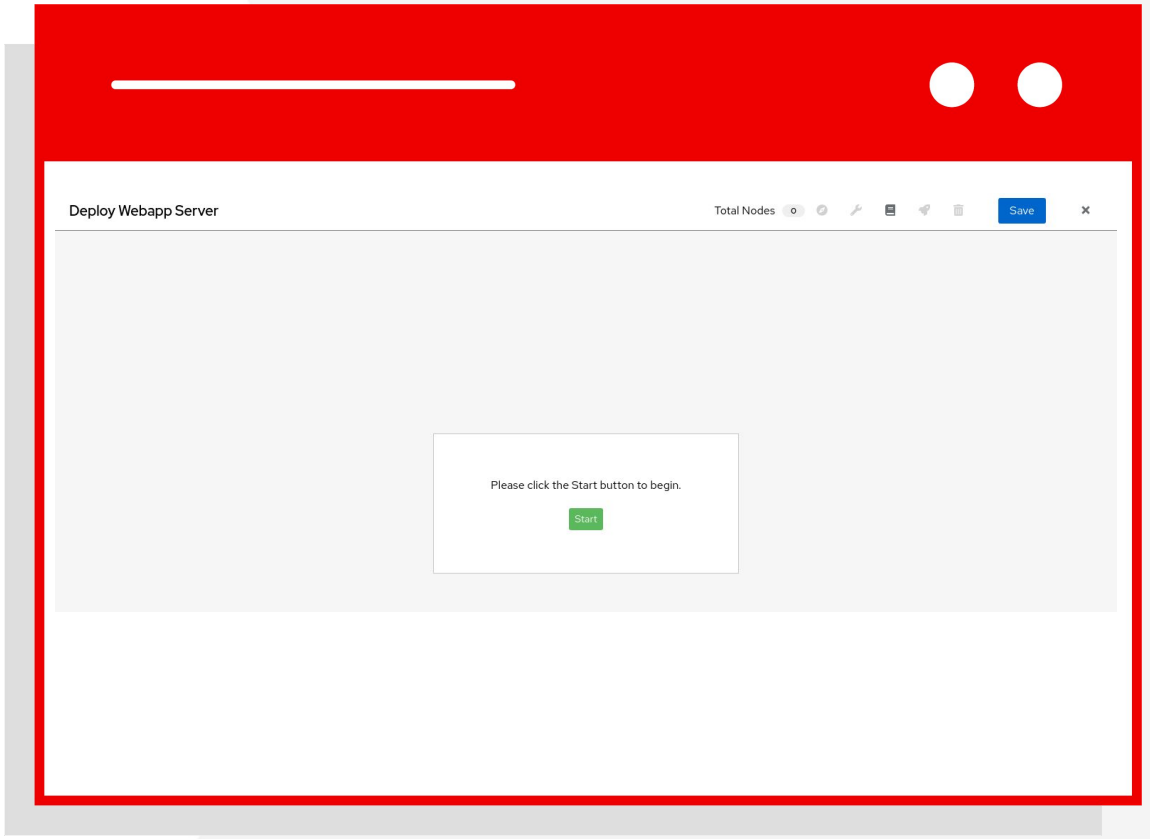
- Fill out the required parameters and click **Save.**

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



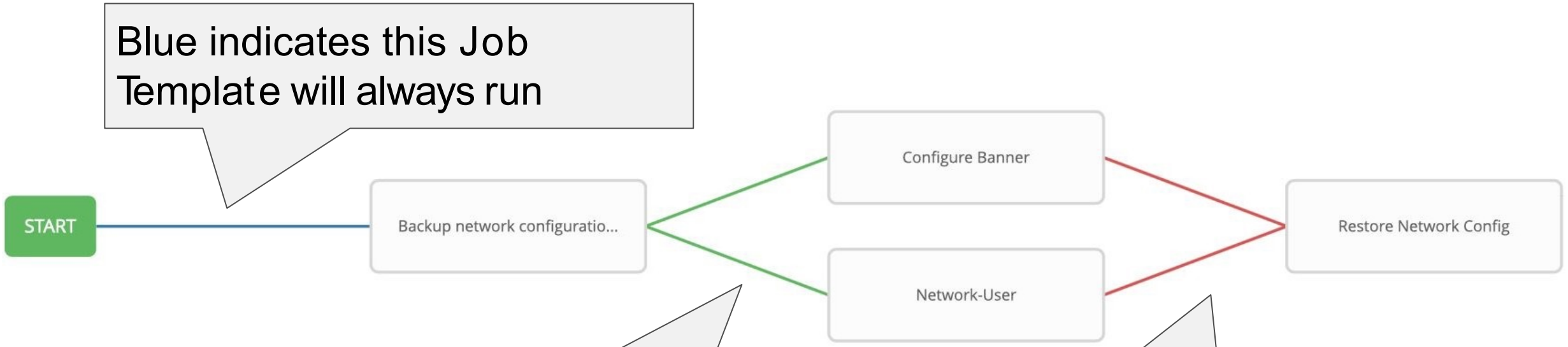
WORKFLOW VISUALIZER

- 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

Blocks

Example with Ansible Blocks

```
---
- name: Ansible Blocks
  hosts: server1
  gather_facts: false

  tasks:
    - block:
        - name: List usr directory
          command: "ls -l /usr/"

        - name: List root directory
          command: "ls -l /root"
      become: yes

    - name: List home directory
      command: "ls -l ~/"
```

Recovery

An additional benefit of using ansible blocks is to perform recovery operations.
If any of the tasks within a block fail, the playbook will exit.

With blocks, we can assign a `rescue` block that can contain a bunch of tasks. If any of the tasks within the block fail, the tasks from the recovery block will automatically be executed to perform clean-up activity.

Rescue Variables

Ansible provide a couple of variables for tasks in the `rescue` portion of a block:

- `ansible_failed_task`

  The task that returned 'failed' and triggered the rescue. For example, to get the name use `ansible_failed_task.name`

- `ansible_failed_result`

  The captured return result of the failed task that triggered the rescue. The same as registering the variable.

Rescue Block

Example with Rescue block

```
---
- name: Ansible Blocks
  hosts: server1
  gather_facts: false

  tasks:
    - block:
        - name: List home directory
          command: "ls -l ~/"

        - name: Failing intentionally
          command: "ls -l /tmp/not-home"
      rescue:
       - name: Rescue block (perform recovery)
         debug:
           msg: "Something broke! Cleaning up.."
```

Always Block

An `always` block will be called independent of the task execution status. It can be used to give a summary or perform additional tasks whether the block tasks fail or not.

Always Block

Example with Always block

```
---
- name: Ansible Blocks
  hosts: server1
  gather_facts: false

  tasks:
    - block:
        - name: List home directory
          command: "ls -l ~/"

        - name: Failing intentionally
          command: "ls -l /tmp/not-home"
      always:
       - name: This always executes
         debug:
           msg: "Can't stop me..."
```

Block Practical Example

Now that we've discussed how a block can be used, let's look at practical examples.

- Install, configure, and start a service
- Apply logic to all tasks in the block
- Enable error handling

Block Practical Example

Practical example (Install, configure, and start Apache

```
---
tasks:
  - name: Install, configure, start Apache
    block:
     - name: Install httpd and memcached
       ansible.builtin.yum:
         name:
         - httpd
         - memcached
         state: present
     - name: Apply config template
       ansible.builtin.template:
         src: templates/src.j2
         dest: /etc/template.conf

     - name: Start/enable service
       ansible.builtin.service:
         name: httpd
         state: started
         enabled: true
    when: ansible_facts['distribution'] == 'CentOS'
    become: true
    become_user: root
```

Block Practical Example

The `when` condition evaluated for all tasks in block.

Practical example (Install, configure, and start Apache

```
---
tasks:
  - name: Install, configure, start Apache
    block:
     - name: Install httpd and memcached
       ansible.builtin.yum:
         name:
         - httpd
         - memcached
         state: present
     - name: Apply config template
       ansible.builtin.template:
         src: templates/src.j2
         dest: /etc/template.conf

     - name: Start/enable service
       ansible.builtin.service:
         name: httpd
         state: started
         enabled: true
    when: ansible_facts['distribution'] == 'CentOS'
    become: true
    become_user: root
```

Block Rescue Practical Example

Practical example (Install nginx and capture failed task)

block_rescue.yml

```yaml
- block:
      - name: Install Nginx
        ansible.builtin.yum:
          name: nginx
          state: present

      - name: Start and enable Nginx service
        ansible.builtin.service:
          name: nginx
          state: started
          enabled: true

    rescue: # These tasks run only if there is a failure on the block
      - name: Show which task failed
        ansible.builtin.debug:
          msg: "{{ ansible_failed_task }}"
```

Block Rescue Task Status

If an error occurs in the block and the rescue task succeeds, Ansible reverts the failed status of the original task for the run and continues to run the play as if the original task had succeeded.

The rescued task is considered successful. However, Ansible still reports a failure in the playbook statistics.

Force Block Failure

You can force a block to fail even if the rescue task is successful:

fail_block.yaml

```yaml
- block:
    - name: Task 1
      ansible.builtin.debug:
        msg: "A task"

    - name: Task 2
      ansible.builtin.debug:
        cmd: "A second task"

  rescue:
  - name: A rescue task
    ansible.builtin.debug:
      msg: "Rescued"

  - name: Fail the block when rescued
    ansible.builtin.fail:
      msg: "A task failed, so failing the whole block"
```

Block Handlers

You can use blocks with `flush_handlers` in a rescue task to ensure that all handlers run even if an error occurs:

```
---
tasks:
  - name: Attempt graceful rollback
    block:
     - name: Print a message
       ansible.builtin.debug:
         msg: 'I execute normally'
       changed_when: yes
       notify: run me even after an error

     - name: Force a failure
       ansible.builtin.command: /bin/false
    rescue:
      - name: Make sure all handlers run
        meta: flush_handlers

handlers:
  - name: Run me even after an error
    ansible.builtin.debug:
      msg: 'This handler runs even on error'
```

Lab: Ansible error handling

Jinja2 Templates

Jinja2 templates are simple template files that store variables that can change from time to time. When Playbooks are executed, these variables get replaced by actual values defined in Ansible Playbooks. This way, templating offers an efficient and flexible solution to create or alter configuration file with ease.

Jinja2 Templates

A Jinja2 template file is a text file that contains variables that get evaluated and replaced by actual values upon runtime or code execution. In a Jinja2 template file, you will find the following tags:

`{{ }}` : These double curly braces are the widely used tags in a template file and they are used for embedding variables and ultimately printing their value during code execution.

`{% %}` : These are mostly used for control statements such as loops and if-else statements.

`{# #}` : These denote comments that describe a task.

Jinja2 Templates

In most cases, Jinja2 template files are used for creating files or replacing configuration files on servers.

Apart from that, you can perform conditional statements such as `loops` and `if-else` statements and transform the data using filters and so much more.

Template files have the `.j2` extension, implying that Jinja2 templating is in use.

Jinja2 Templates

A simple Jinja2 template example.

```
Hey guys! Apache webserver {{ version_number }} is running on {{ server }} Enjoy!
```

```
The variables are "{{ version_number }}"
and "{{ server }}"
```

Jinja2 Templates

When the playbook is executed, the variables in the template file are replaced with declared vars.

```
---
- hosts:
  vars:
    version_number: "2.3.52"
    server: "Ubuntu"
tasks:
  - name: Jinja 2 template example
    template:
      src: my_template.j2
      dest: /home/ansible/myfile.txt
```

Lab: Ansible 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.

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'
        },
    }
```

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