Red Hat
Training and
Certification

# Ansible Automation Platform 2.x Webinar

Travis Michette

Version 1.0

## Table of Contents

# Introduction to Ansible Automation

# 1. Ansible & Ansible Automation Engine (Past)

## 1.1. Ansible Infrastructure

The Ansible infrastructure and Ansible Automation consists of multiple components. The initial main foundation of Ansible is the Ansible Automation Engine.

For the most part, Ansible is a declarative automation platform that is considered idempotent meaning that Ansible will only execute tasks and plays if the item needs to be changed/modified. Otherwise, Ansible will skip to the next task or play in a playbook.

*Ansible Components*

- **Control Node**: System with Ansible installed, contains Ansible inventory files, **ansible.cfg**, and playbooks. This system manages and controls other managed hosts/nodes.
- **Managed Host/Managed Node**: System or node being managed in the Ansible environment. The **Control Node** executes various Ansible modules against these devices.

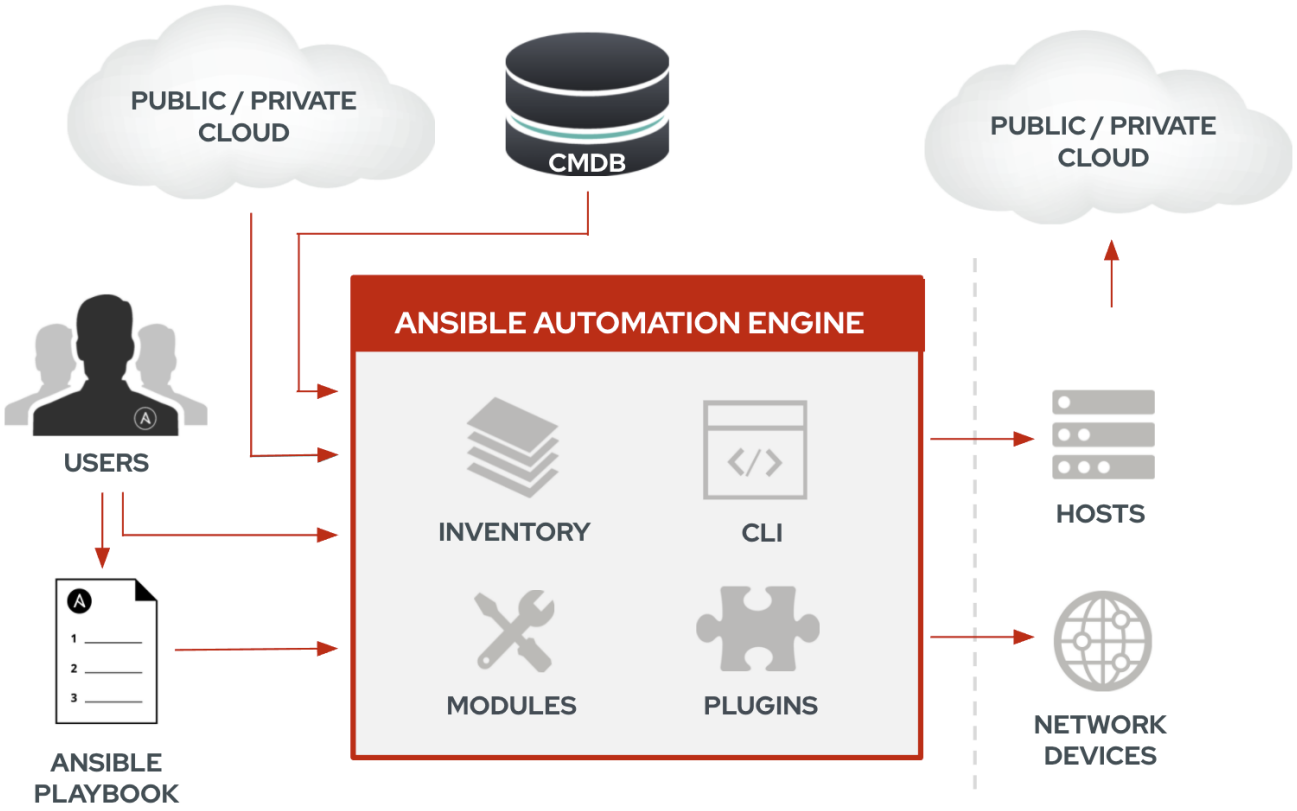


*Figure 1. Ansible Automation*

*Ansible Automation Engine*

- Inventory
- Command-Line Interface (CLI)
- Modules (Generally Python/Powershell)
- Plugins

Ansible Automation Engine utilizes the **ansible** command for Ad-Hoc Ansible Automation or the **ansible-playbook** command for running multiple tasks by leveraging and Ansible playbook containing one or more plays consisting of one or more tasks.

### 1.1.1. Inventory

List of systems in the infrastructure to be managed. Inventories can be static, dynamic, or a combination of both static and dynamic. Ansible also allows inventories to contain variables for the devices being managed. Devices must exist in inventory in order for Ansible to be capable of managing the devices.

### 1.1.2. Modules

Code utilized by the Ansible core engine which is used to perform a given tasks. Most modules are written in Python for Linux and Powershell for Windows. Modules can extend Ansible automation to multiple platforms simplifying and extending the automation to the entire stack.

*Non-Idempotent Modules*

There are some Ansible modules that aren't idempotent. Modules such as **commmand**, **shell**, and **raw** to name a few will execute regardless of the state. It is possible to use these modules with logic to make a playbook idempotent, but it is recommended to find an actual Ansible module to perform the task. These modules should be used as a last resort when no other module exists to perform a task.

### 1.1.3. Plugins

Code utilized by the Ansible core engine which is used to manipulate, transform, or otherwise modify either data in the playbook or items captured by the playbook and modules so that it is adaptable and usable on different platforms.

### 1.1.4. Playbooks

List of sequential tasks to allowing individual Ansible modules to be executed to perform a sequence of steps in an automation task. Playbooks are written in YAML and are simple easy-to-read steps on the end state of the system.

### 1.1.5. Ansible Tower

Ansible Tower delivers enterprise management and features to the Ansible family. Through Tower, Ansible can provide the following:

- Role-Based Access Control (RBAC)
- Restful API
- Push button deployment

- Workflows
- Credential and Secret Management
- Integration into SCM systems
- Integration into other management systems for dynamic inventory
- WebUI
- … and more

Ansible Tower allows enterprises to manage their IT environment by providing a centralized web solution to end-users and administrators to perform automation and self-service tasks. :pygments-style: tango :source-highlighter: pygments :icons: font :icons: font

## 1.2. Ansible Inventory, Ansible Config File, and Ansible Ad-Hoc Commands

### 1.2.1. Ansible Inventory

Ansible inventories can be either static, dynamic, or a combination of both static and dynamic. The traditional form of the Ansible inventory file is either YAML or INI. Inventory items consist of either individual managed nodes or groups of managed nodes.

*Listing 1. Ansible Inventory File* **INI** *Format*

```
servera ①
serverb
serverc
serverd

[load_balancers] ②
servere
serverf
```

① Individual managed host/node

② Inventory Group

*Converting INI to YAML Inventory*

Ansible provides the **ansible-inventory** command that will easily allow the inventory to be transformed from one form to another.

```
ansible-inventory --list -y
```

*Listing 2. Ansible Inventory File* **YAML** *Format*

```
all:
  children:
    load_balancers: ①
      hosts:
        servere: {}
        serverf: {}
    ungrouped:
      hosts: ②
        servera: {}
        serverb: {}
        serverc: {}
        serverd: {}
```

① Inventory Hosts in a Group

② Individual managed host/node (ungrouped)

#### 1.2.1.1. Inventory Variables

It is possible for Ansible playbooks and Ansible ad-hoc commands to utilize inventory variables. These variables can be defined directly within the static inventory files themselves or those variables can be defined within the directory structure of the project utilizing either project directories or inventory directories.

*Keep Inventory Simple and Organized*

It is extremely important not to define variables for inventory in multipe locations as variable precedence and variable merging comes into play. It is equally important to devise an inventory strategy on where/how variables will be defined so that the playbooks and automation goals are kept simple and easy to understand and follow.

*Listing 3. Sample Inventory File with Variables*

```
[app1srv]
appserver01 ansible_host=10.42.0.2 ①
appserver02 ansible_host=10.42.0.3

[web]
node-[1:30] ansible_host=10.42.0.[31:60]

[web:vars] ②
apache_listen_port=8080
apache_root_path=/var/www/mywebdocs/

[all:vars] ③
ansible_user=kev
ansible_ssh_private_key_file=/home/kev/.ssh/id_rsa
```

① Defined variable at a **host** level

② Defined variables at a **group** level

③ Defined variables for all inventory items

### 1.2.2. Ansible Config

The **ansible.cfg** file controls how the **ansible** and **ansible-playbook** commands are run and interpreted. The configuration file has two (2) main sections that are commonly used, but include other sections as well. For the purpose of understanding how Ansible works, we will examine both the **[defaults]** section and the **[privilege_escalation]** section.

*Listing 4.* **ansible.cfg** *Defaults Section*

```
[defaults]
inventory = inventory ①
remote_user = devops ②
```

① Specifies which inventory file Ansible will use

② Specifies the remote user to be used by **ansible** or **ansible-playbook** commands.

A perfectly acceptable **ansible.cfg** might only have a [defaults] section specifying the inventory to be used.

*Listing 5.* **ansible.cfg** *Privilege Escalation Section*

```
[privilege_escalation]
become = False ①
become_method = sudo ②
become_user = root ③
become_ask_pass = False ④
```

① Sets default behavior whether to elevate privileges

② Sets method for privilege escalation

③ Sets username of privileged user

④ Sets option on whether or not user is prompted for password when perfoming privilege escalation.

*Ansible Config File Precedence*

- **ANSIBLE_CONFIG** - Environment Variable (highest)
- **ansible.cfg** - Config file in current working directory (most common and recommended)
- **~/.ansible.cfg** - Ansible config file in the home directory
- **/etc/ansible/ansible.cfg** - Ansible's installed default location (lowest)

### 1.2.3. Ansible Ad-Hoc Commands

Ansible Ad-Hoc commands are most often used to quickly perform an automation task using a single Ansible module. These commands can be executed against one or more hosts in the Ansible inventory file.

*Table 1. Ansible* **Ad-Hoc** *Command Arguments*

| Command Argument | Description |
|---|---|
| **-m MODULE_NAME** | Module name to execute for the ad-hoc command |

| Command Argument | Description |
|---|---|
| **-a MODULE_ARGS** | Module arguments needed for the ad-hoc command |
| **-b** | Runs ad-hoc command as a privileged user |
| **-K** | Runs ad-hoc command as a privileged user and requests the **become** password |
| **-e EXTRA_VARS** | Provides extra variables as **KEY=VALUE** to be used for the execution of the ad-hoc command |

### 1.2.4. DEMO - Ansible Ad-Hoc Commands

Demonstration and hands-on workshop for Ad-Hoc commands. The demo will utilize the **ping** module to ensure that the **ansible.cfg** and the **inventory** file are correctly setup and working within the Ansible environment.

*Example 1.* **DEMONSTRATION - Ansible Ping**

1. Change to correct directory

```
[student@workstation ~]$ cd /home/student/Github/AAP_Webinar/Past/ad-hoc
```

2. Run the **ansible** ad-hoc command

```
[student@workstation ad-hoc]$ ansible -m ping all
servere | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
servera | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
serverc | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
serverb | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
serverd | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
serverf | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

*Checking Sudoers Ability and Setup*

*Listing 6. Checking* **ansible.cfg** *for Ability to* **BECOME** *without* **sudo** *Password*

```
[student@workstation ad-hoc]$ ansible -m ping all --become
```

*Listing 7. Checking* **ansible.cfg** *for Ability to* **BECOME** *with* **sudo** *and Prompting for Password*

```
[student@workstation ad-hoc]$ ansible -m ping all --become -K
BECOME password:
```

The next demonstration will use the **copy** module to create a user in the managed systems making an entry to the **sudoers** file.

*Example 2.* **DEMONSTRATION - Ansible Ad-Hoc Command to Create User and Sudoers File**

1. Change to correct directory

```
[student@workstation ~]$ cd /home/student/Github/AAP_Webinar/Past/ad-hoc
```

2. Run the **ansible** commands to create the user and update the **sudoers** file.

   a. Create the user on the remote system.

```
[student@workstation ad-hoc]$ ansible -m user -a 'name=travis uid=1040 comment="Travis Michette" group=wheel' servera -b
servera | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "append": false,
    "changed": false,
    "comment": "Travis Michette",
    "group": 10,
    "home": "/home/travis",
    "move_home": false,
    "name": "travis",
    "shell": "/bin/bash",
    "state": "present",
    "uid": 1040
}
```

   b. Create the user in a **sudoers** file.

```
[student@workstation ad-hoc]$ ansible -m copy -a 'content="travis ALL=(ALL) NOPASSWD:ALL" dest=/etc/sudoers.d/travis' servera
-b
```

3. Test new user and sudo rights

   a. SSH to **servera**

```
[student@workstation ad-hoc]$ ssh travis@servera
```

   b. **sudo** without a password

```
[travis@servera ~]$ sudo -i
[root@servera ~]#
```

## 1.3. Ansible Playbooks

Ansible playbooks contain one or more tasks to execute against specified inventory nodes. Playbooks consist of one or more play and each play in a playbook consists of one or more tasks. Ansible playbooks and tasks are all about **key:value** pairs and **lists**. Understanding this basic format allows someone developing Ansible to form playbooks that are easier to create,

troubleshoot/debug, and for someone else to understand.

### 1.3.1. Playbook Basics

An Ansible playbook is written/formatted in YAML so horizontal whitespace is critical and often the most troublesome part of debugging new Ansible playbooks. Playbooks have a general structure for the plays with directives such as: **name**, **hosts**, **vars**, **tasks**, and more. These play-level directives help form a readable structure much like **task-level** directives.

*Listing 8. Play Structure Components*

```
---
- name: install and start apache ①
  hosts: web ②
  become: yes ③

  tasks: ④
```

① Name of play in playbook

② List of hosts from inventory to execute play against **(required)**

③ Directive to override **ansible.cfg** and elevate privileges

④ Beginning of **tasks** section.

There can be other directives here, but at the most basic playbook, you will generally always see a **hosts** and a **tasks** section.

*Naming Plays*

It is recommended and considered a best practice to name all plays within a playbook.

The first indentation level in a playbook denoted by **-** is the list of plays and this level will contain the **key:value** pairs that correspond to Ansible playbook directives. Understanding this and developing good habits and standards for indentations allows Ansible users to create playbook skeletons which help tremendously during the development/debugging cycle.

#### 1.3.1.1. Task Structure

A task within a playbook is generally specified similar to a play having a **name** section so that it is easier to debug.

*Listing 9. Task Structure and Components*

```
tasks:
  - name: httpd package is present ①
     yum: ②
       name: httpd ③
       state: latest ④

  - name: latest index.html file is present ⑤
    template:
      src: files/index.html
      dest: /var/www/html/

  - name: httpd is started ⑥
    service:
      name: httpd
      state: started
```

① Name of first task in playbook

② Name of module to be used in first task in playbook

③ Argument/Option provided to module, in this instance **name** and is ***required*** for the package name in the case of the **yum** module.

④ Argument/Option provided to module, in this instance the **state** describes whether the module will install, update, or remove the package for the **yum** module.

⑤ Name of second task in playbook

⑥ Name of third task in playbook

*Naming Tasks*

It is recommended and considered a best practice to name all tasks within a playbook. Naming tasks especially helps with debugging issues in playbooks as the Ansible STDOUT will display and record task names.

The second indentation level in a playbook denoted by **-** is generally under the **tasks:** section and contains the list of tasks. This level will contain the **key:value** pairs that correspond to Ansible task directives always starting with the module being used at the same level before going to the third indentation level which are the **key:value** pair options that belong to the module being used in that task.

### 1.3.2. Running Playbooks

Playbooks can be run just like Ansible **ad-hoc** commands. In order to execute or run a playbook, it is necessary to use the **ansible-playbook** command and specify the playbook. The additional options available for the **ad-hoc** commands such as: **-e**, **-K**, **-b**, and others all still apply and perform the same functions when leveraged with the **ansible-playbook** command.

*Example 3. DEMONSTRATION - Running Ansible Playbooks*

In this demonstration, we will be creating a user on **serverb** using playbooks as opposed to leveraging **ad-hoc** commands.

1. Switch to correct directory

```
[student@workstation ~]$ cd ~/Github/AAP_Webinar/Past/Playbooks
```

2. Examine playbook

```
[student@workstation Playbooks]$ vim playbook.yml

---
- name: Playbook to Create User and Sudoers without Password
  hosts: serverb
  tasks:
    - name: Create User Named Travis
      user:
        name: travis
        uid: 1040
        comment: "Travis Michette"
        group: wheel

    - name: Create User in Sudoers File
      copy:
        content: "travis ALL=(ALL) NOPASSWD:ALL\n"
        dest: /etc/sudoers.d/travis
        validate: /usr/sbin/visudo -csf %s
```

3. Execute and run the playbook

```
[student@workstation Playbooks]$ ansible-playbook playbook.yml -b

PLAY [Playbook to Create User and Sudoers without Password] ***************************

... OUTPUT OMITTED ...
```

4. Test and Verify User

   a. SSH to remote system

```
[student@workstation Playbooks]$ ssh travis@serverb
```

   b. Verify Sudo without Password

```
[travis@serverb ~]$ sudo -i
[root@serverb ~]#
```

*Example 4. DEMONSTRATION - Failure of Old Playbook*

It is important to constantly test playbooks with the most current and recent versions of Ansible to ensure all modules work as expected and items haven't been deprecated. The following playbook was developed for use with Ansible 2.8 and earlier. The playbook now fails as some of the modules being used have been migrated from Ansible **built-in** modules to Ansible collections. More on this migration and discussion of collections will come in future chapters and sections.

1. Examine Playbook for Website

```
---
- name: Playbook to Fully Setup and Configure a Webserver
  hosts: servera
  tasks:
    - name: Install Packages for Webserver
      yum:
        name:
          - httpd
          - firewalld
        state: latest

    - name: Create Content for Webserver
      copy:
        content: "I'm an awesome webserver"
        dest: /var/www/html/index.html

    - name: Create Content for Webserver
      copy:
        content: "I'm an awesome webserver"

    - name: Firewall is Enabled
      service:
        name: firewalld
        state: started
        enabled: true

    - name: HTTP Service is Open on Firewall
      firewalld:
        service: http
        state: enabled
        permanent: true
        immediate: yes

    - name: httpd is started
      systemd:
        name: httpd
        state: started
        enabled: true
```

2. Execute the playbook

```
[student@workstation Playbooks]$ ansible-playbook Website_Past.yml
ERROR! couldn't resolve module/action 'firewalld'. This often indicates a misspelling, missing collection, or incorrect module
path. ①

The error appears to be in '/home/student/Github/AAP_Webinar/Past/Playbooks/Website_Past.yml': line 27, column 7, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:


    - name: HTTP Service is Open on Firewall
      ^ here
```

① The **firewalld** module is not available. This was moved in AAP 2.x to an Ansible collection and is no longer able to be referenced without the collection and module being installed.

*Test Often*

As Ansible has changed going into Ansible Automation Platform 2+, many changes have been made. There was a duplication and mapping of several of the modules allowing for existing playbooks to still run correctly, however, at some point modules become completely deprecated, and mappings get removed. It is extremely important to execute old playbooks and test with new versions of Ansible and to look for **deprecation warnings** so that playbooks can be fixed proactively instead of reactively.

## 1.4. Ansible Roles

Ansible Roles allow Ansible developers to create re-usable code snippets and tasks that can be shared in the form of Ansible Roles. These roles are commonly shared via Ansible Galaxy (https://galaxy.ansible.com/) via Github projects by the role developers. With the new Ansible Automation Platform (AAP) 2.x and beyond roles are also included as part of Ansible Collections which will be covered as part of a later topic.

### 1.4.1. Ansible Role Overview

Ansible roles are resusable Ansible components that allow common tasks to be repeated/repurposed without needing to re-write or create custom playbooks. Ansible roles work the same way as Ansible Playbooks and tasks except that roles are generally published/shared generically to be used by others to perform a task or set of tasks in automation playbooks.

Roles provide Ansible with a way to load tasks, handlers, and variables from external files. Static files and templates can also be associated and referenced by a role.

**Role Benefits**

- Roles group content which allows easy sharing of code with others
- Roles can be written that define the essential elements of system type: web server, database server, git repository, or other purpose
- Roles make larger projects more manageable
- Roles can be developed in parallel by different administrators

*Table 2.* **Ansible role subdirectories**

| Subdirectory | Function |
|---|---|
| **defaults** | The **main.yml** file in this directory contains the default values of role variables that can be overwritten when the role is used. |
| **files** | This directory contains static files that are referenced by role tasks. |
| **handlers** | The **main.yml** file in this directory contains the role's handler definitions. |
| **meta** | The **main.yml** file in this directory defines information about the role, including author, license, platforms, and optional role dependencies. |
| **tasks** | The **main.yml** file in this directory contains the role's task definitions. |

| Subdirectory | Function |
| --- | --- |
| **templates** | This directory contains Jinja2 templates that are referenced by role tasks. |
| **tests** | This directory can contain an inventory and **test.yml** playbook that can be used to test the role. |
| **vars** | The **main.yml** file in this directory defines the role's variable values. |

*Defining Variables and Defaults*

- **Role** variables are defined by creating ***var/main.yml*** with name/value pairs in the role directory heirarchy.
- **Default variables** are defined by creating a ***defaults/main.yml*** file with name/value pairs in the role directory heirarchy.

It is best to define a given variable in either ***var/main.yml*** or ***defaults/main.yml*** but not both.

**Playbook Roles and Include Statements**: http://docs.ansible.com/ansible/playbooks_roles.html

*Reference for Roles*

There is another workshop with some information on creating and publishing Ansible Roles.

https://github.com/tmichett/LUG/blob/main/Ansible_Roles/Workbook/Ansible.adoc

### 1.4.2. Using Roles

In order to use Ansible roles, they must first be installed and made available on the Ansible control node utilizing the role.

*Listing 10. Installing an Ansible Role*

```
$ ansible-galaxy install tmichett.deploy_packages
```

*Example 5. DEMONSTRATION - Using a Role from Ansible Galaxy*

1. Change to correct working directory

```
[student@workstation ~]$ cd Github/AAP_Webinar/Past/Roles/
```

2. Install Role from Ansible Galaxy

```
[student@workstation Roles]$ ansible-galaxy install -r roles/requirements.yml -p roles
Starting galaxy role install process
- downloading role 'ansiblize', owned by tmichett
- downloading role from https://github.com/tmichett/Ansiblize/archive/master.tar.gz
- extracting tmichett.ansiblize to /home/student/Github/AAP_Webinar/Past/Roles/roles/tmichett.ansiblize
- tmichett.ansiblize (master) was installed successfully
```

3. Install Collections

```
[student@workstation Roles]$ ansible-galaxy collection install -r collections/requirements.yml -p collections/
Starting galaxy collection install process
[WARNING]: The specified collections path
'/home/student/Github/AAP_Webinar/Past/Roles/collections' is not part of the
configured Ansible collections paths
'/home/student/.ansible/collections:/usr/share/ansible/collections'. The installed
collection won't be picked up in an Ansible run.
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/ansible-posix-1.3.0.tar.gz to /home/student/.ansible/tmp/ansible-local-
37837_73lx2j8/tmpnadbl_rx/ansible-posix-1.3.0-xr73p6ye
Installing 'ansible.posix:1.3.0' to '/home/student/Github/AAP_Webinar/Past/Roles/collections/ansible_collections/ansible/posix'
ansible.posix:1.3.0 was installed successfully
```

Because we are using a newer Ansible version, the regular modules are no longer available so we must install the Ansible Posix collection to use some of the modules in the role.

4. Create/Modify Playbook with Correct Values and Providing Variables for the Role

```
 create mode 100644 Workshop_Guide/redhat-theme.yml
[student@workstation Past]$ cd
[student@workstation ~]$ ls
---
- name: Playbook to Fully Setup and Configure a new User with a Role
  hosts: serverc
  vars: ①
    ansible_user_name: travis
    ansible_user_password: redhat
    ssh_key_answer: no
  roles: ②
    - tmichett.ansiblize
```

① Providing required variables for the role

② Providing the role being used

5. Run the playbook with the **ansible-playbook** Command

```
[student@workstation Roles]$ ansible-playbook Roles_Playbook_Demo.yml
```

6. Test the results on **serverc**

   a. SSH to ServerC

   ```
   [student@workstation Roles]$ ssh travis@serverc
   ```

   b. Attempt to become root without password

   ```
   [travis@serverc ~]$ sudo -i
   [root@serverc ~]#
   ```

# 2. Ansible Automation Platform 1.x (Present)

## 2.1. Ansible Automation Hub

Ansible Automation hub was released so that enterprises could host their own Ansible collections, Ansible execution environments in a disconnected fashion. Ansible Automation Hub provides self-hosted services from Ansible Galaxy as well as services provided by Red Hat's Ansible Automation Hub found at https://console.redhat.com/ansible/ansible-dashboard.

### 2.1.1. Ansible Automation Platform

Provides curated collections, modules, roles that are supported by Red Hat or Red Hat Partners. The collections, modules, roles, and playbooks downloaded from Red Hat Automation Hub are supported and required an Ansible Automation Platform subscription entitlement. This is different from the unsupported community modules/collections/roles found at Ansible Galaxy (https://galaxy.ansible.com/). :pygments-style: tango :source-highlighter: pygments :icons: font :icons: font

## 2.2. Ansible Collections

Ansible 2.9 introduced the concept of collections and provided mapping for Ansible modules that were moved into a collection namespace. Ansible 2.9 provided a mapping of the new module locations in collections and this mapping automatically works for existing Ansible playbooks in the initial versions of Ansible Automation Platform.

*Ansible Module and Collection Mapping*

https://github.com/ansible/ansible/blob/devel/lib/ansible/config/ansible_builtin_runtime.yml

### 2.2.1. Using Ansible Automation Platform Collections

Collections allowed development of Ansible core components to be separated from module and plug-in development. Upstream Ansible unbundled modules from Ansible core code beginning with Ansible Base (core) 2.10/2.11. Never versions of Ansible require collections to be installed in order for modules to be available for Ansible. Playbooks should be developed using the FQCNs when referring to modules in tasks. Existing playbooks can be fixed easily to work with collections, but it is better to re-write the playbooks to use the fully-qualified collection name (FCQN).

*Example 6. DEMONSTRATION - Using Ansible Collections*

1. Change to the correct working directory

```
[student@workstation ~]$ cd /home/student/Github/AAP_Webinar/Present/Playbooks
```

2. View the Playbook

```
[student@workstation Playbooks]$ vim Website_Present.yml
---
- name: Playbook to Fully Setup and Configure a Webserver
  hosts: servera
  tasks:
    - name: Install Packages for Webserver
      yum:
        name:
          - httpd
          - firewalld
        state: latest

    - name: Create Content for Webserver
      copy:
        content: "I'm an awesome webserver\n"
        dest: /var/www/html/index.html

    - name: Firewall is Enabled
      systemd:
        name: firewalld
        state: started
        enabled: true

    - name: HTTP Service is Open on Firewall
      ansible.posix.firewalld:
        service: http
        state: enabled
        permanent: true
        immediate: yes

    - name: httpd is started
      systemd:
        name: httpd
        state: started
        enabled: true
```

3. Install the Ansible Collections

```
[student@workstation Playbooks]$ ansible-galaxy collection install -r collections/requirements.yml -p collections/
```

4. Execute the playbook with the **ansible-playbook** command

```
[student@workstation Playbooks]$ ansible-playbook Website_Present.yml -b
```

5. Test the Website

```
[student@workstation Playbooks]$ curl servera
I'm an awesome webserver
```

# 3. Ansible Automation Platform 2.x (Future)

## 3.1. Introduction to AAP 2.x Components

Section Info Here

### 3.1.1. Ansible Content Navigator

### 3.1.2. Ansible Execution Environments

## 3.2. Introduction to AAP 2.x - Ansible Automation Hub

Section Info Here

### 3.2.1. Private Automation Hub

### 3.2.2. Custom Execution Environments

## 3.3. Introduction to AAP 2.x - Ansible Controller (formerly Ansible Tower)

Section Info Here

### 3.3.1. Organizations, Teams, and RBAC

*Example 7. DEMONSTRATION - Creating an Organization with Users and Teams*

*Creating an Organization*

1. Login to **Ansible Controller**

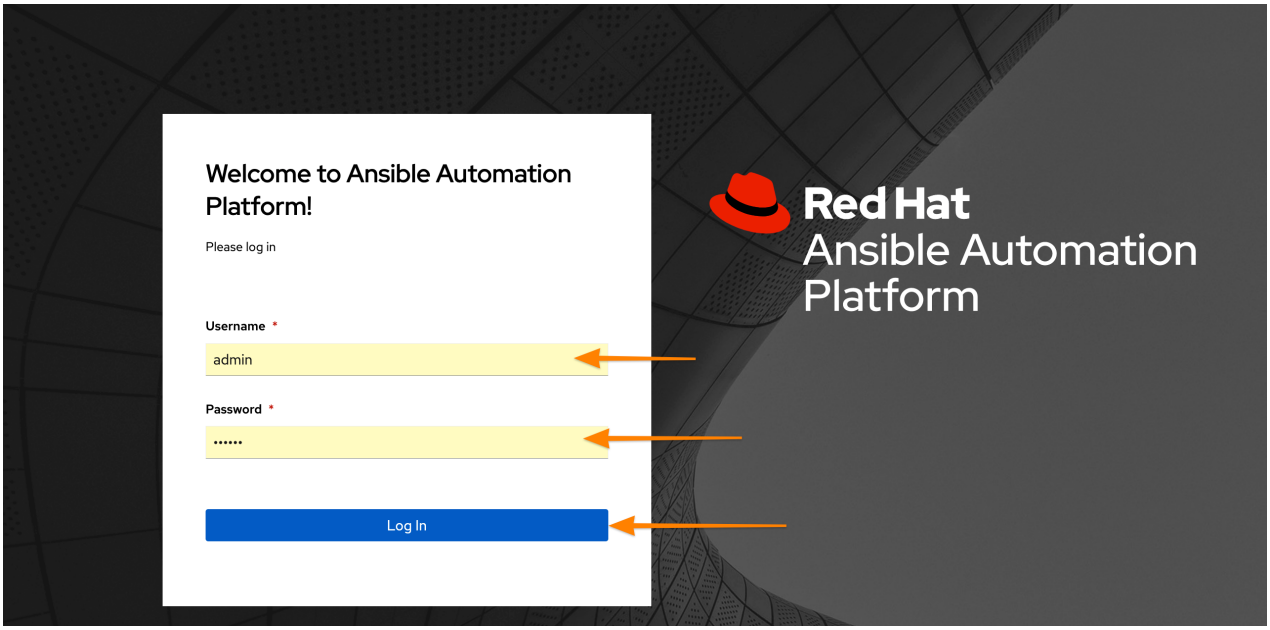


*Figure 2. Ansible Controller Login*

2. Click **Organizations**

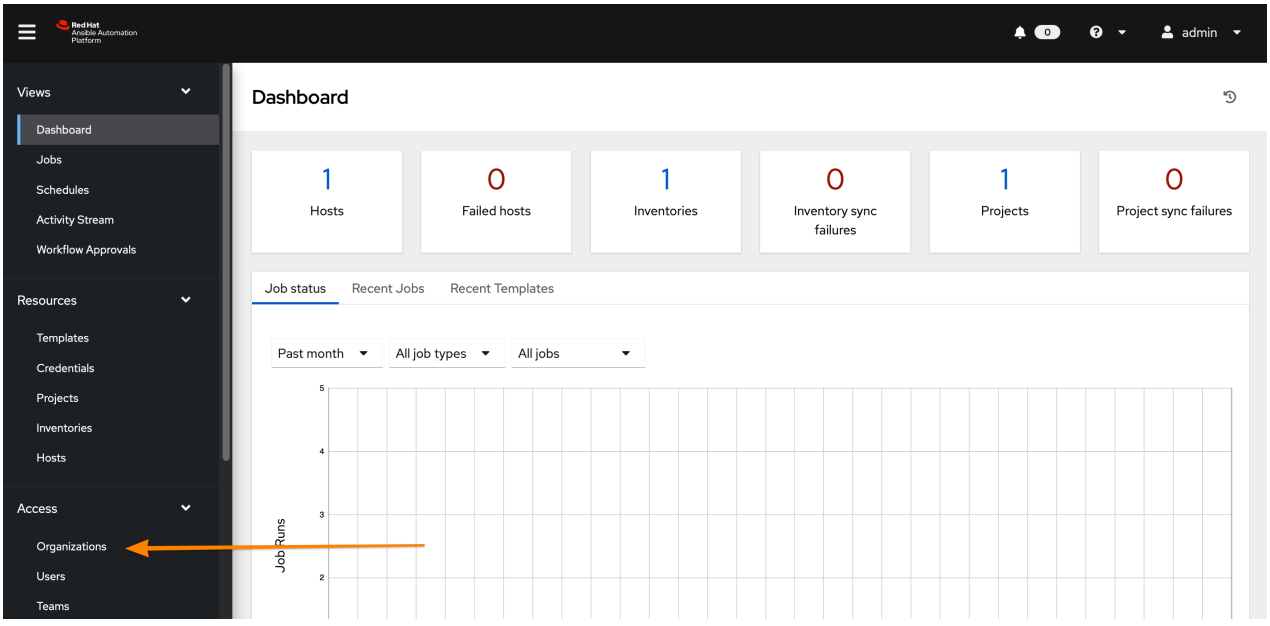


*Figure 3. Ansible Controller* - **Organizations**

3. Click **Add**

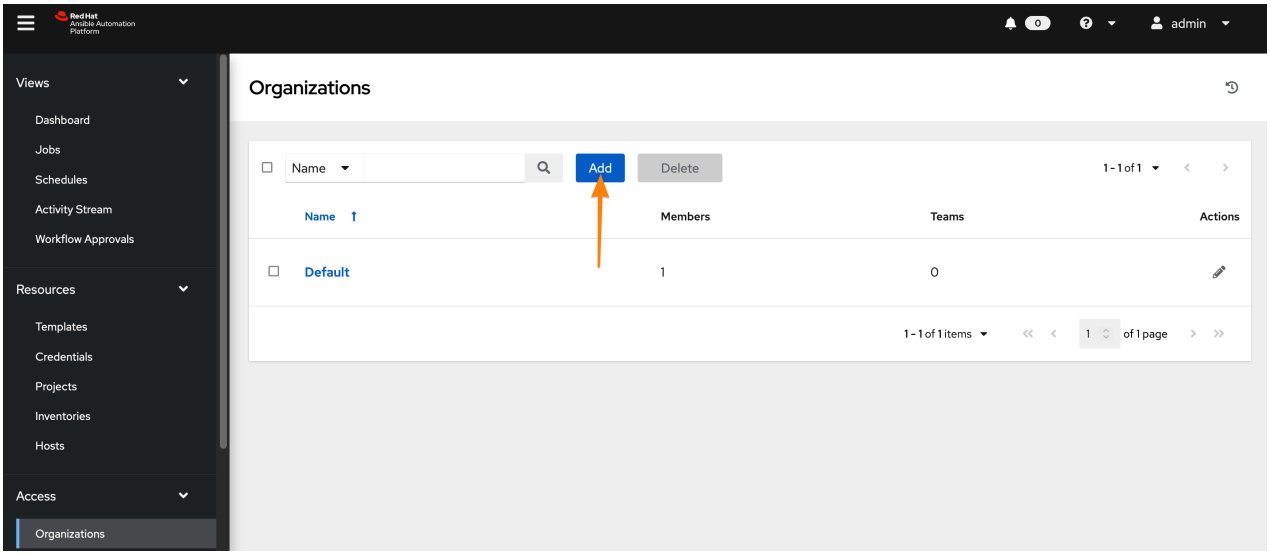


*Figure 4. Ansible Controller* - **Adding an Organization**

4. Create the new organization with the appropriate details and then click **Save**
    a. **Name**: Required for name of Organization
    b. **Description**: *Optional but helpful*
    c. **Execution Environment**: Default execution environment for playbooks, projects, and workflows in the environment.

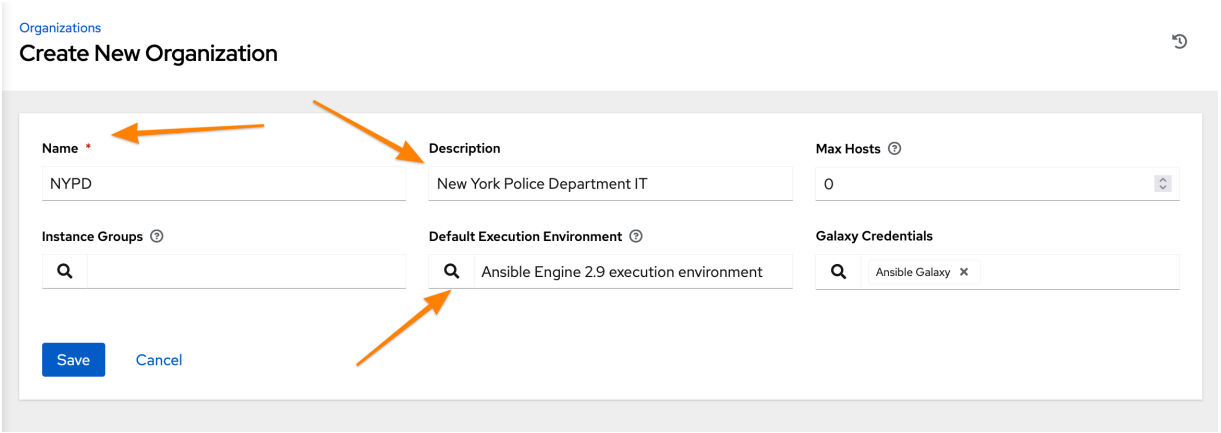


*Figure 5. Ansible Controller* - **Configuring the Organization**

*Default Execution Environment*

It is helpful to setup the default execution environment (EE) which will control the running of Ansible playbooks within your Organization. It is possible for individual projects and playbooks to be selectively run with another execution environment, but the default EE will be used if another EE isn't specified.

In the above example, the **Ansible Engine 2.9 execution environment** was selected as it has the best compatibility with older playbooks before the realigned modules and collections. Ansible Engine 2.9 can utilize collections, but also has the mapping for the Ansible modules allowing older playbooks to run without updating to using FQCN.

*Creating a Team*

1. Click on **Teams**

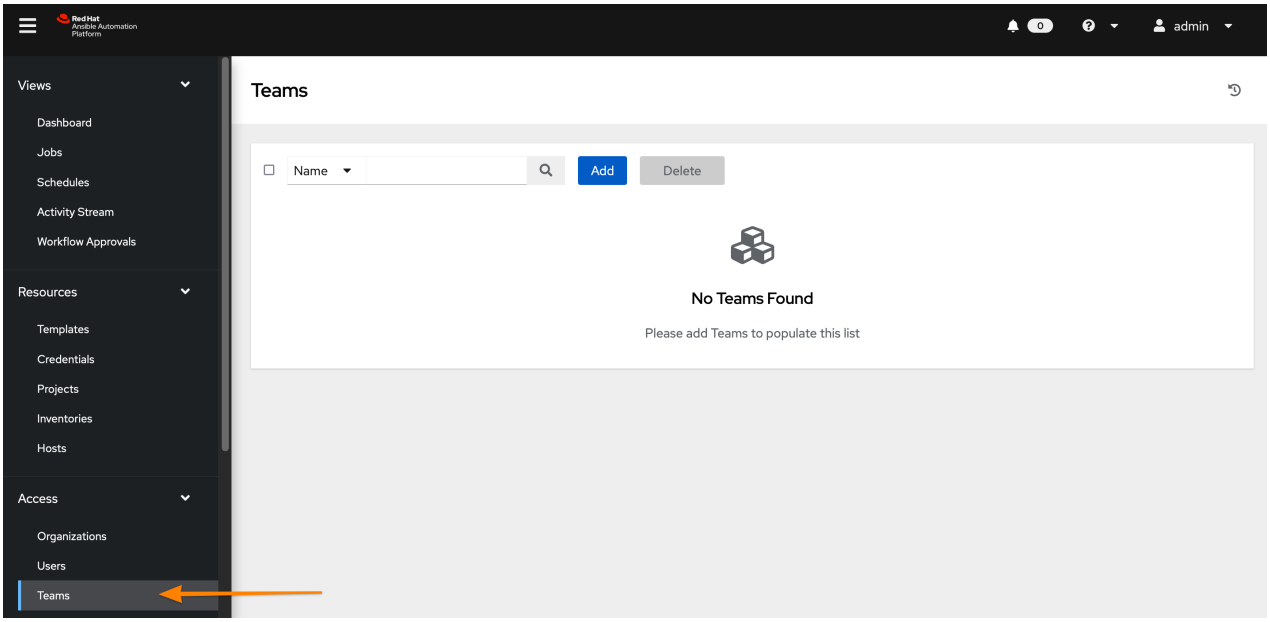


*Figure 6. Ansible Controller* - **Creating a Team**

2. Click **Add** and fill in appropriate values and then click **Save**
    a. **Name**: Required for name of team
    b. **Organization**: Required to select an existing organization from drop-down
        i. Click magnifying glass and select Organization

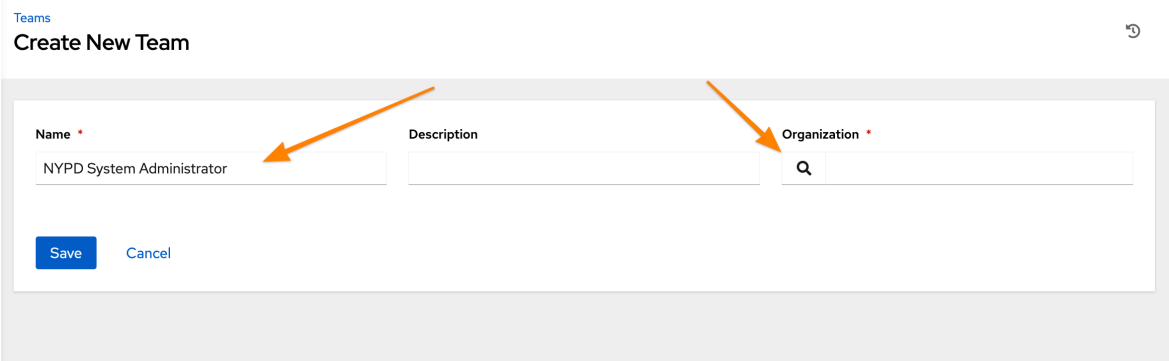


*Figure 7. Ansible Controller* - **Configuring a Team**


*Figure 8. Ansible Controller* - **Selecting an Organization**

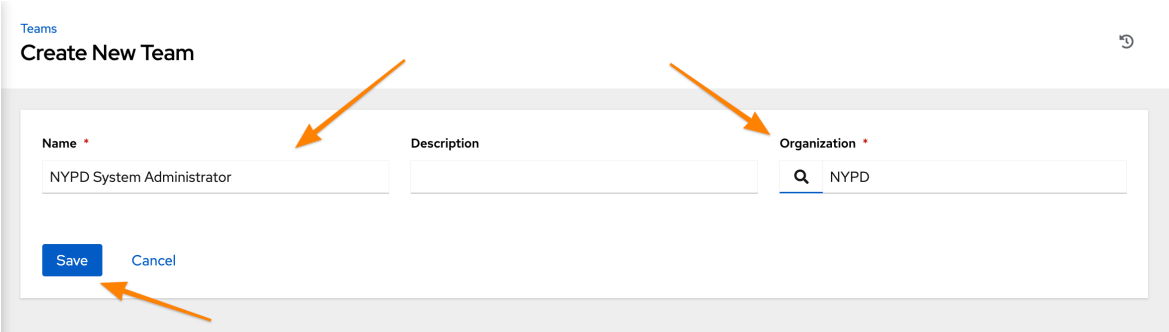


*Figure 9. Ansible Controller* - **Completing Team Creation**

*Creating Users*

1. Click **Users**

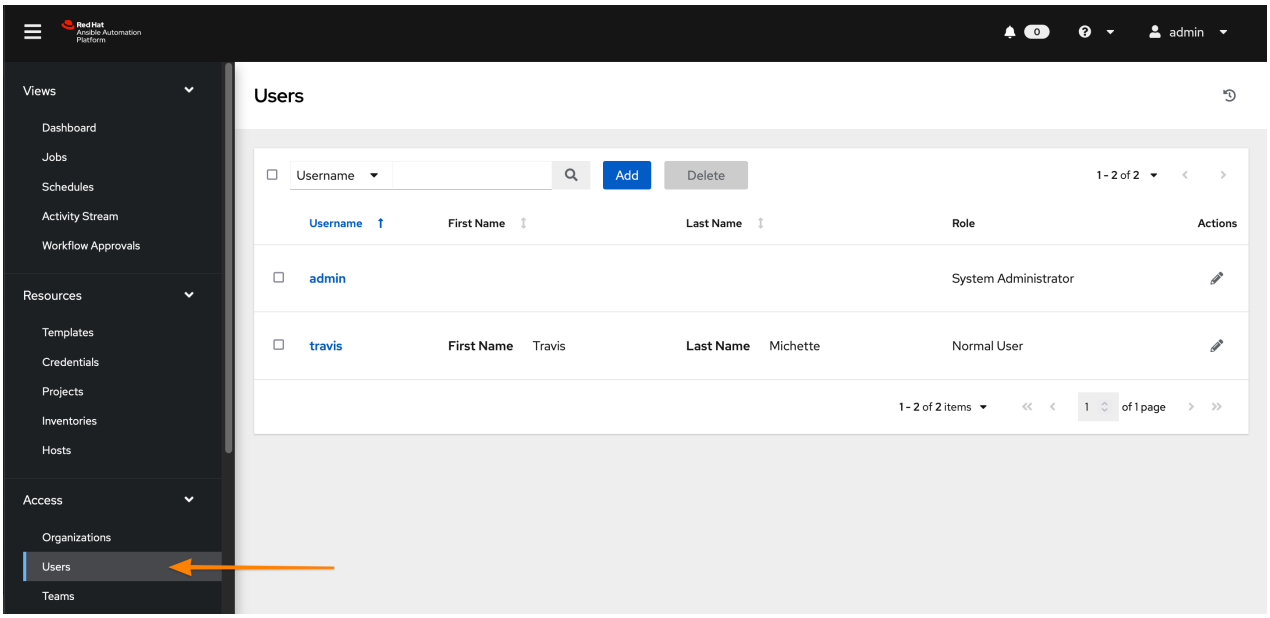


*Figure 10. Ansible Controller* - **Creating a User**

2. Click **Add** and fill in appropriate information and click **Save**


*Figure 11. Ansible Controller* - **Selecting an Organization**

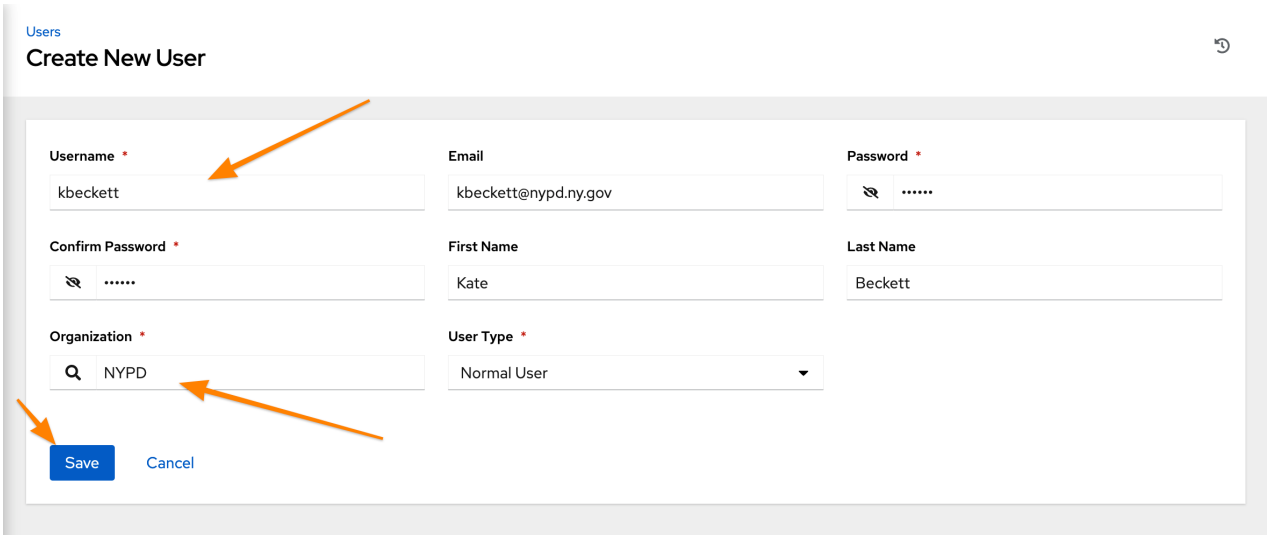


*Figure 12. Ansible Controller* - **Configuring a User**

*Adding a User to a Team*

Adding users to a team can be done multiple ways, but in this example, we will be modifying a recently created user and use the **Teams** option on the User **Details** tab.

1. Click on **Teams**

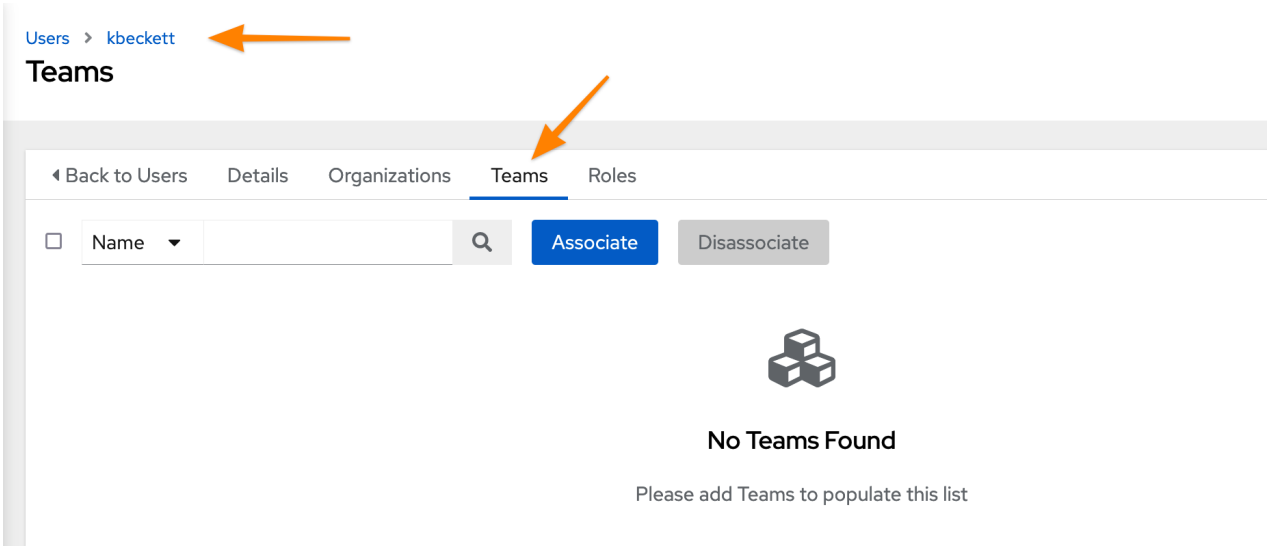


*Figure 13. Ansible Controller* - **User Teams Menu**

2. Click the **Associate** button to search for and select a team, then click **Save**

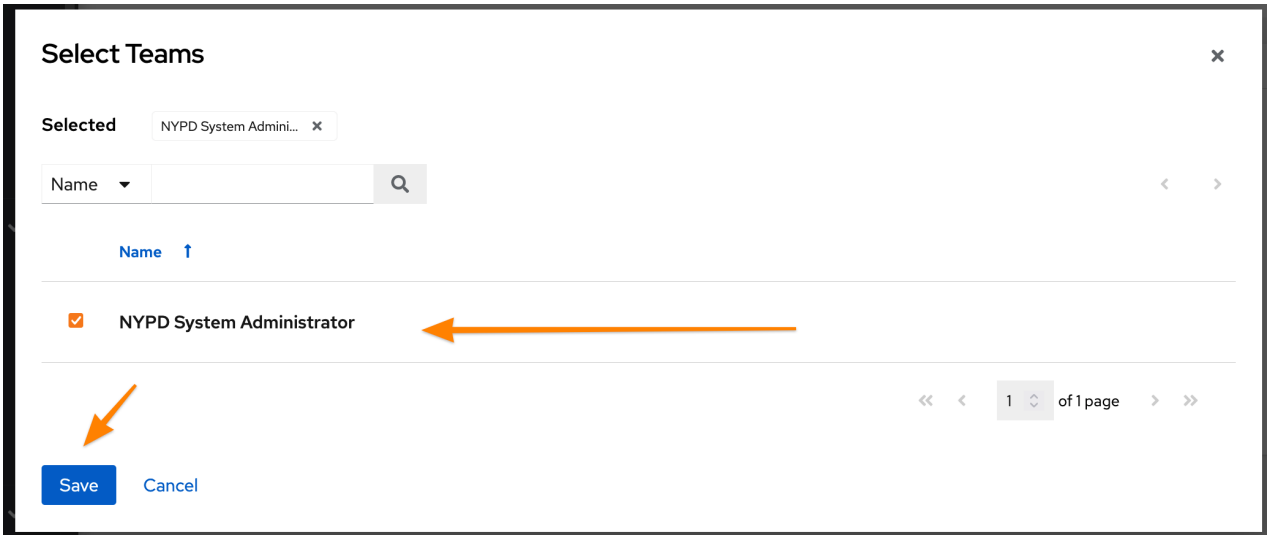


*Figure 14. Ansible Controller* - **User Team(s) Selection**

3. Verify user was associated with the correct team(s).

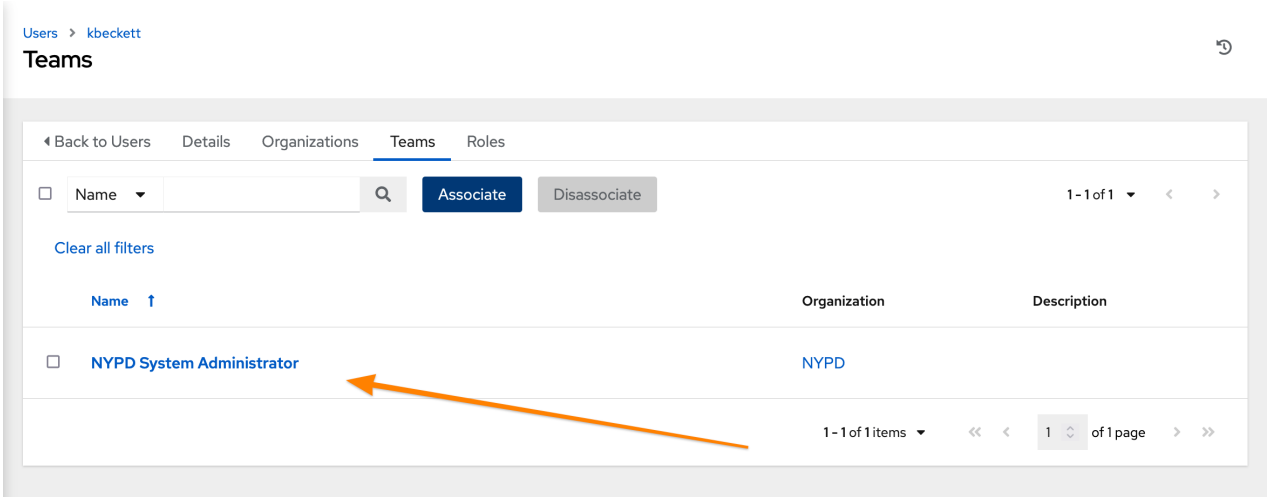


*Figure 15. Ansible Controller* - **User Team(s) Verification**

*Teams*

Teams are used to group users together so that RBAC controls can be more easily managed at a group level versus an individual user level. It is still possible to give individual users additional privileges, but teams is the preferred way of permission management.

### 3.3.2. Inventories and Credentials

*Example 8. DEMONSTRATION - Creating an Inventories and Credentials*

*Creating an Iventory*

1. Login to **Ansible Controller**

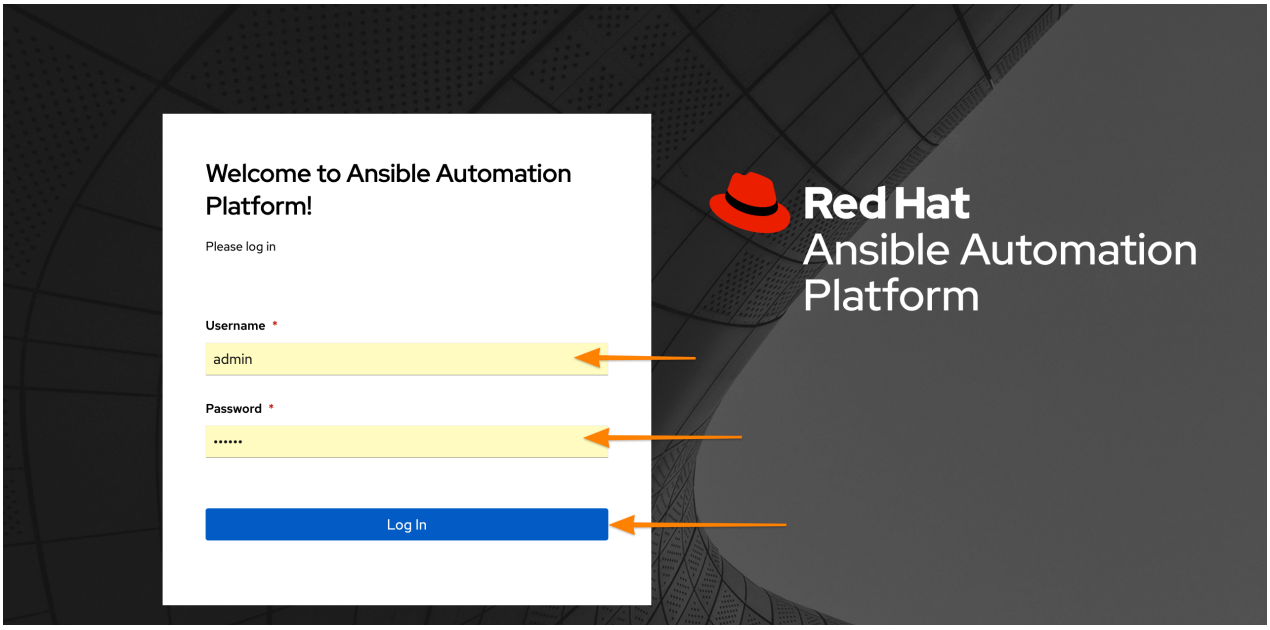


*Figure 16. Ansible Controller Login*

2. Click **Inventories** and then click **Add** to Add an Inventory

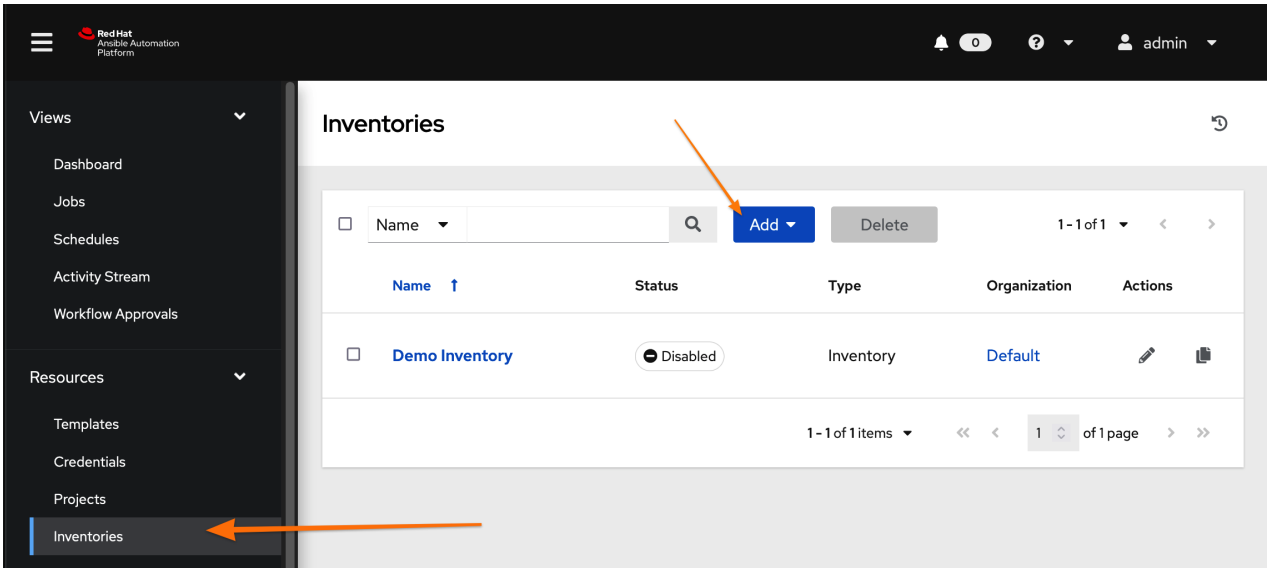


*Figure 17. Ansible Controller - Inventory*

3. Provide and inventory ***Name*** and ***Organization*** and then click **Save**

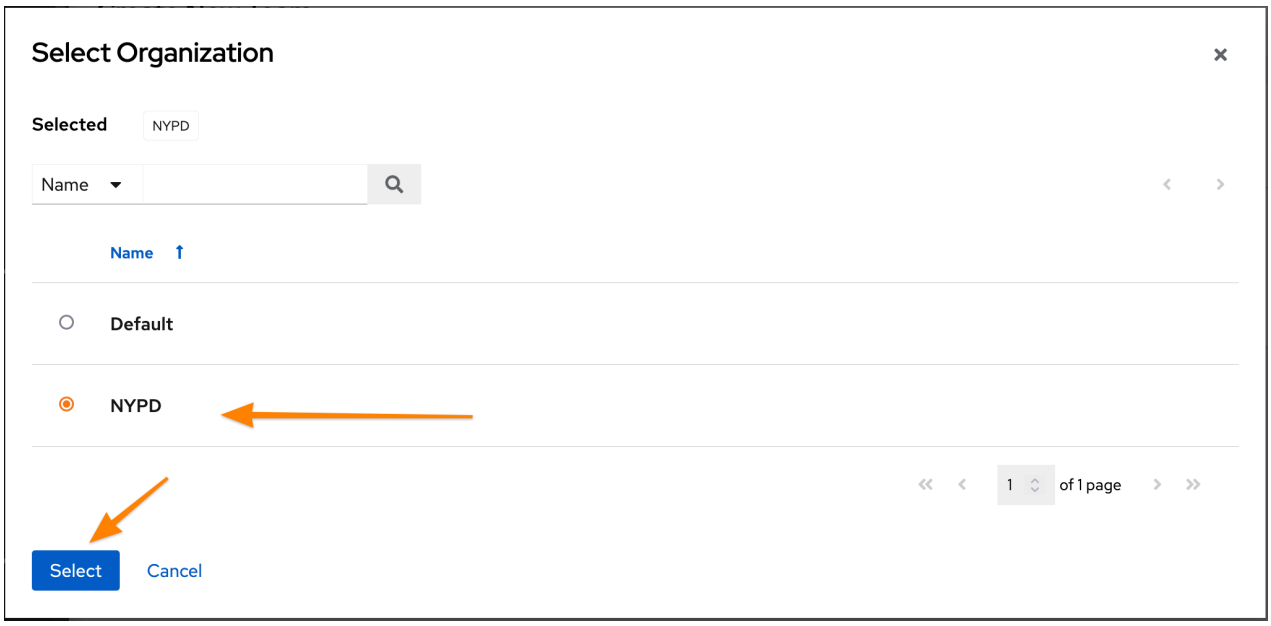


*Figure 18. Ansible Controller -* **Selecting an Organization**

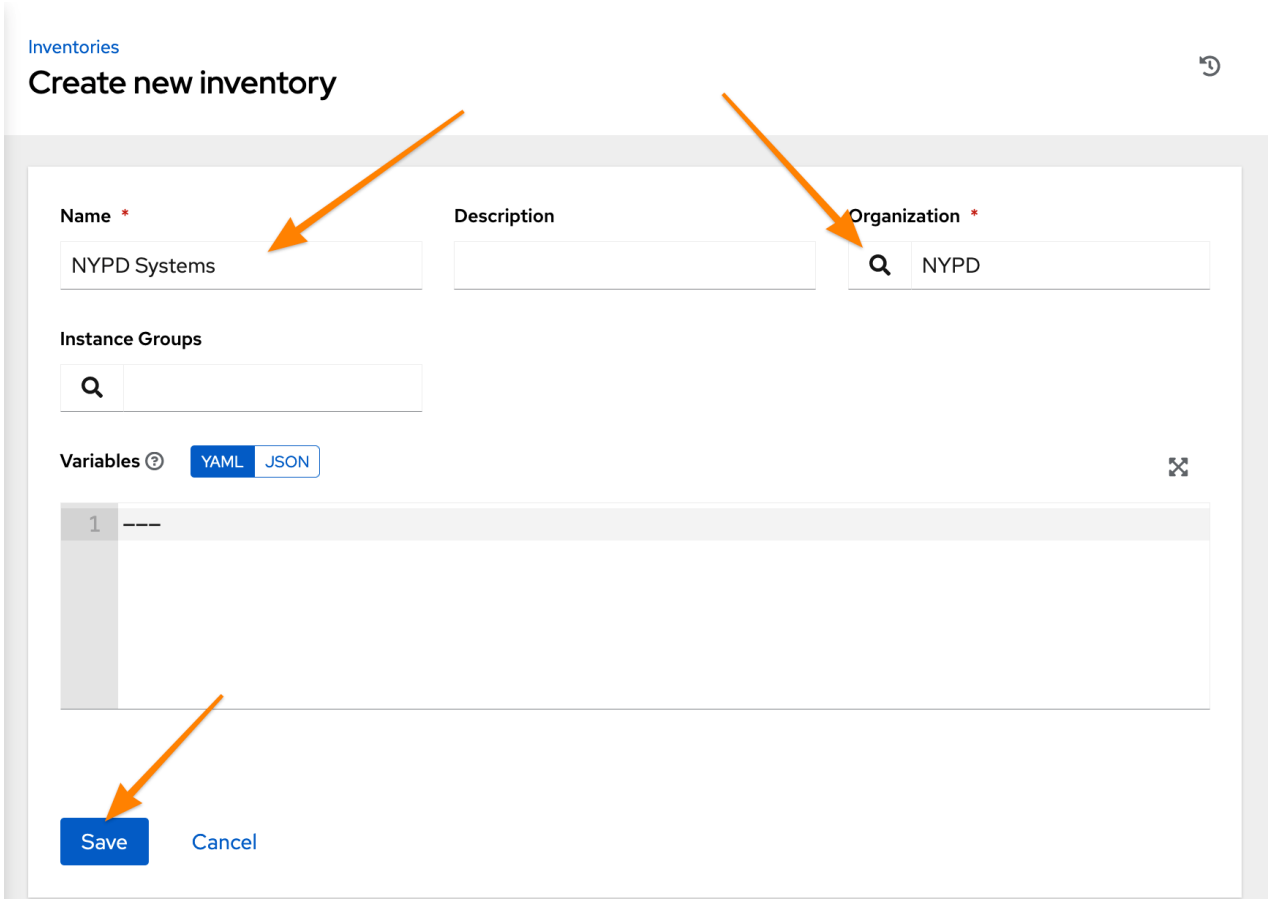


*Figure 19. Ansible Controller - New Inventory*

4. Add hosts to the inventory by clicking **Hosts** and then click **Add**

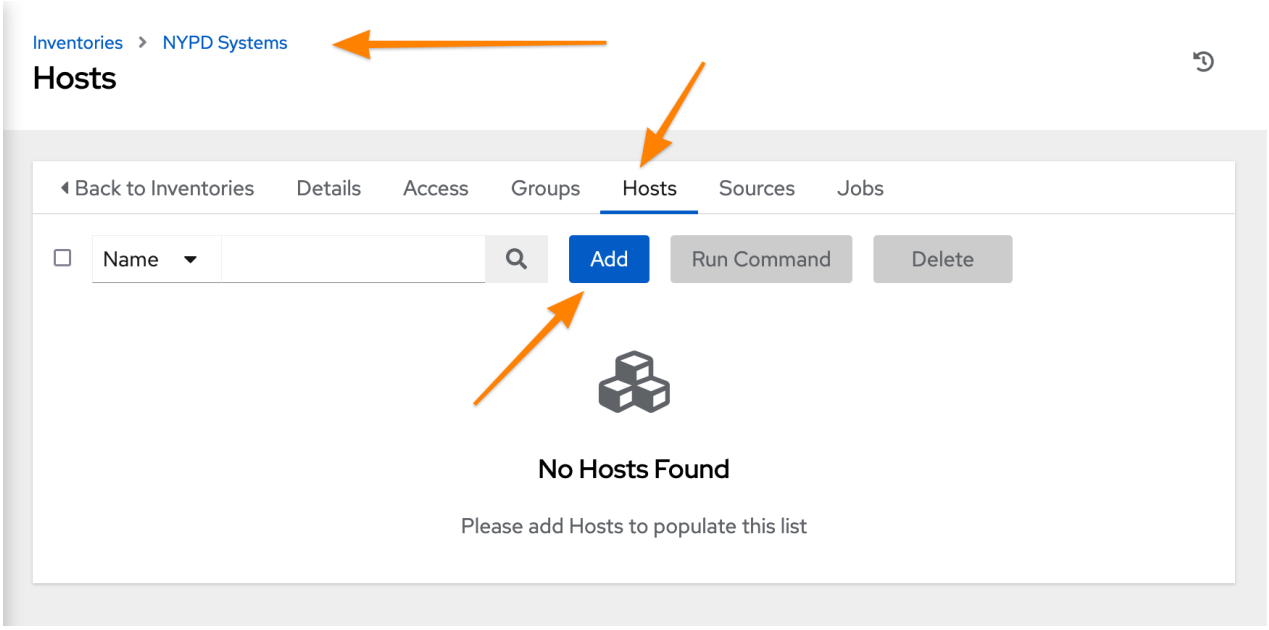


*Figure 20. Ansible Controller - Managed Hosts in Inventory*

5. Provide the inventory hostname of the host and any host-based variables if desired and click **Save**. Repeat for multiple hosts.

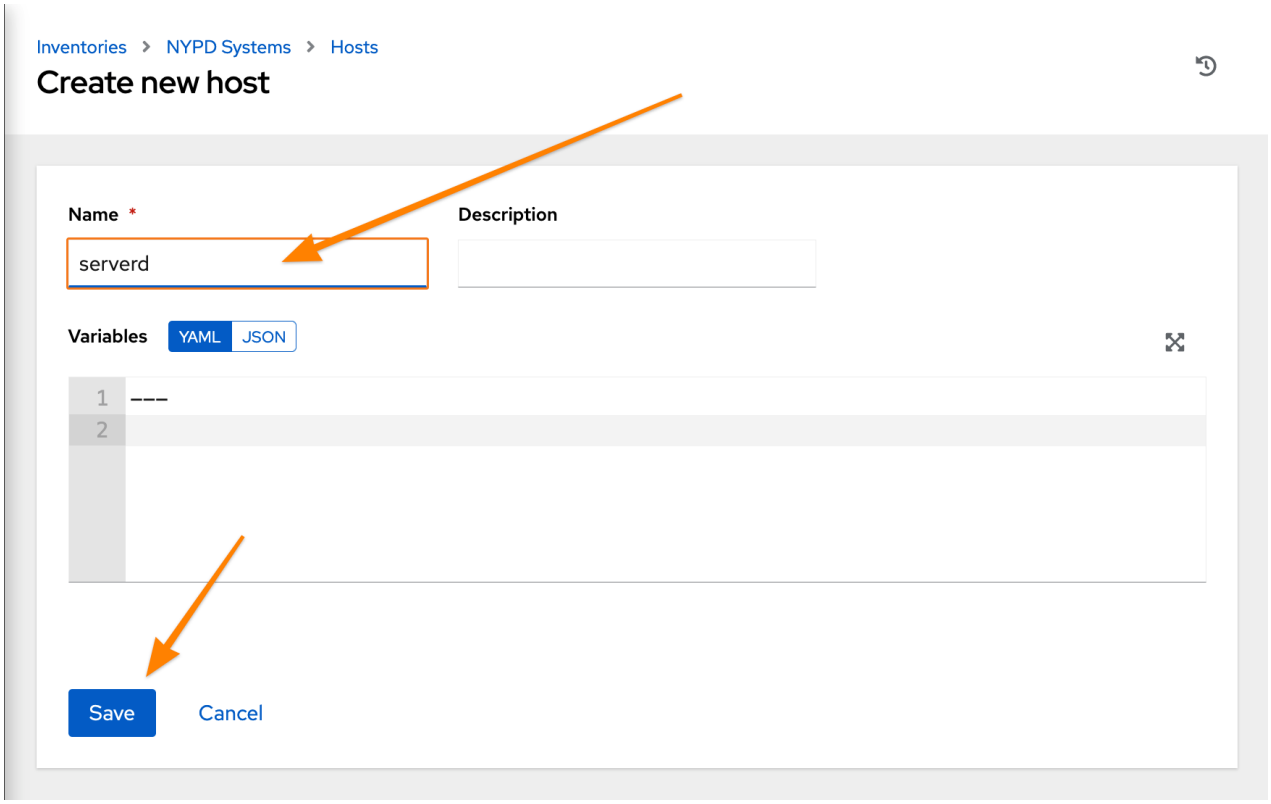


*Figure 21. Ansible Controller - Adding a Host to Inventory*

*Creating Credentials*

1. Click **Credentials** and then click **Add** to add a new credential

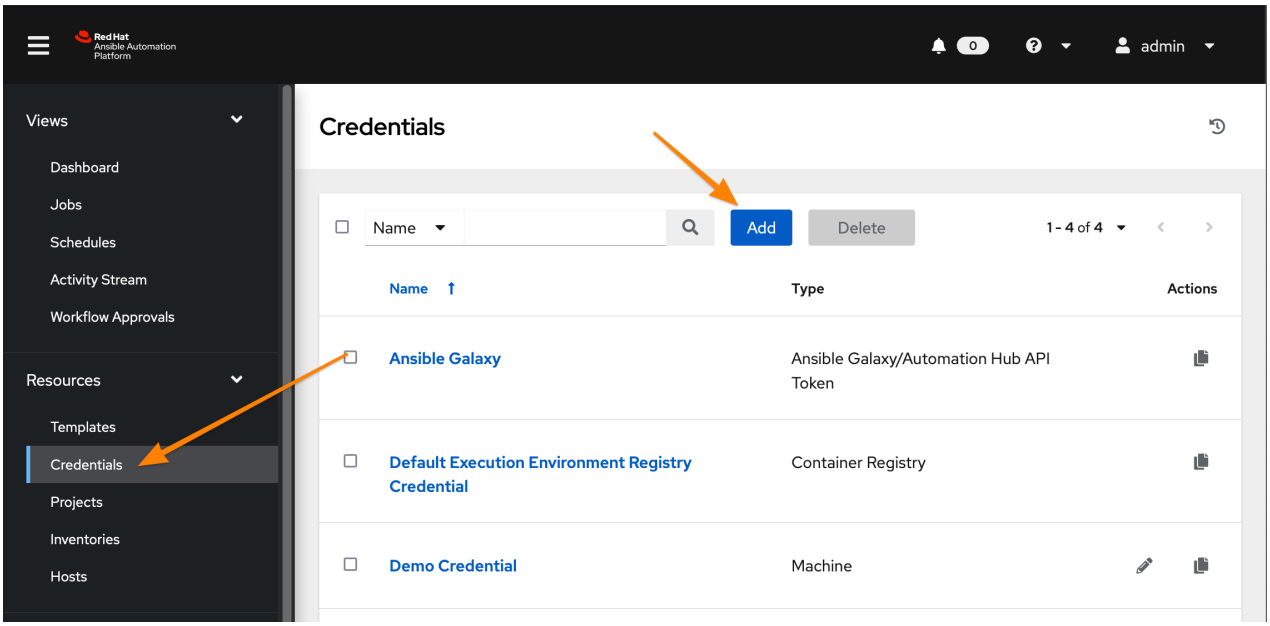


*Figure 22. Ansible Controller - Credentials*

2. Create the credential specifying the name, type, and bind to organization if desired and click **Save**.
    - NOTE - SSH credentials are **Machine Credentials**

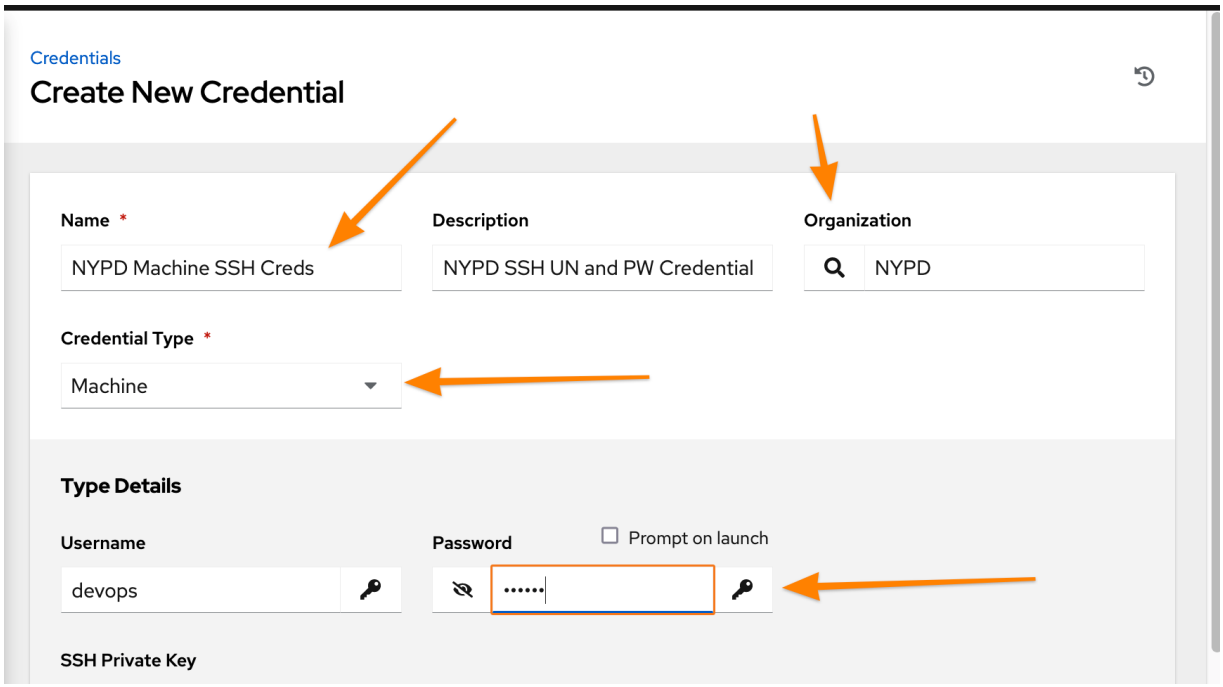


*Figure 23. Ansible Controller - Machine Credentials*

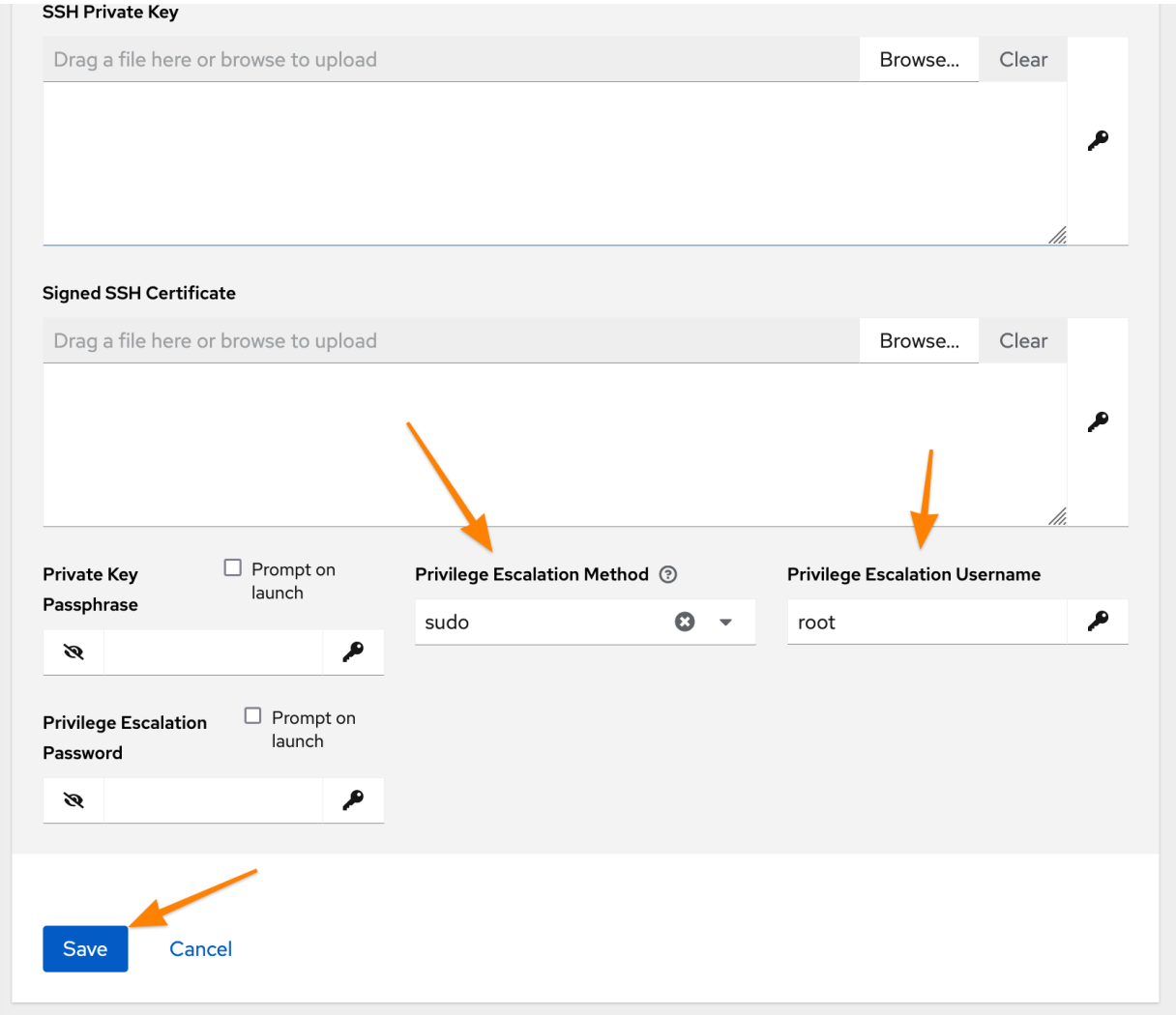


*Figure 24. Ansible Controller - Credentials - Privileged User*

*Privilege Escalation*

It is necessary to provide privilege escalation information as with Ansible Controller, this is where the information and configuration must come from for execution environments (EEs).

### 3.3.3. Projects and Job Templates

*Example 9. DEMONSTRATION - Creating an Projects and Job Templates*

*Creating a Project*

1. Login to **Ansible Controller**

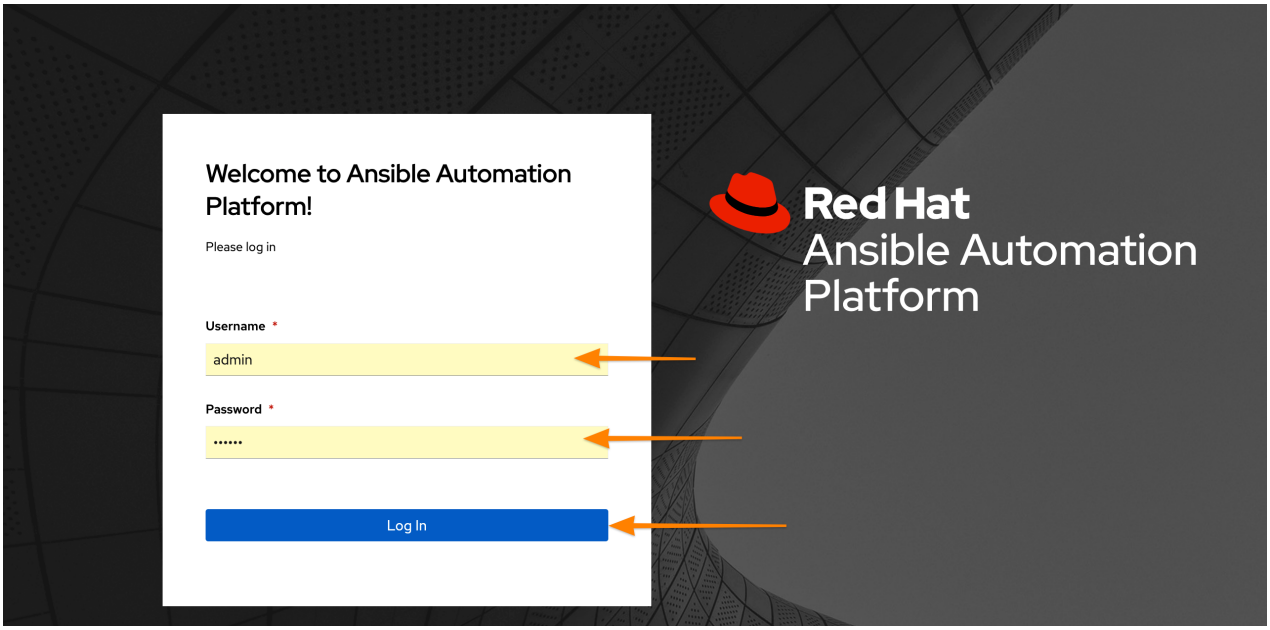


*Figure 25. Ansible Controller Login*

2. Click **Projects** and then click **Add** to Add a Project

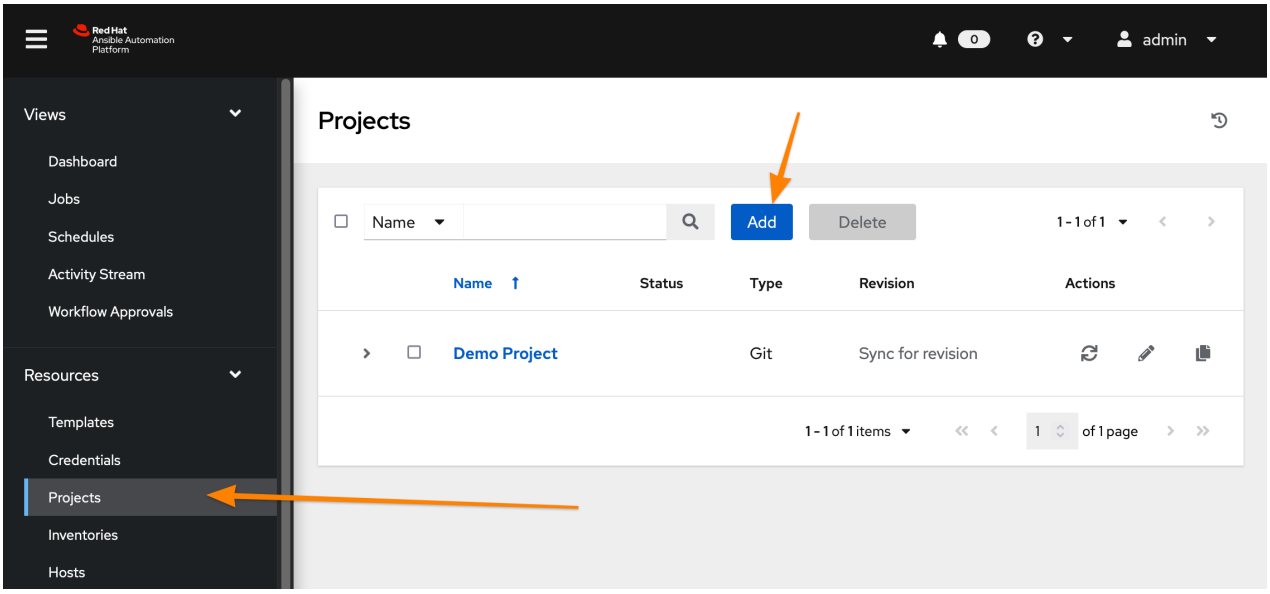


*Figure 26. Ansible Controller - Projects*

3. Create a New Project with a Name, Organization and Source Control Credential
    - URL: git@github.com:tmichett/AAP_Webinar.git

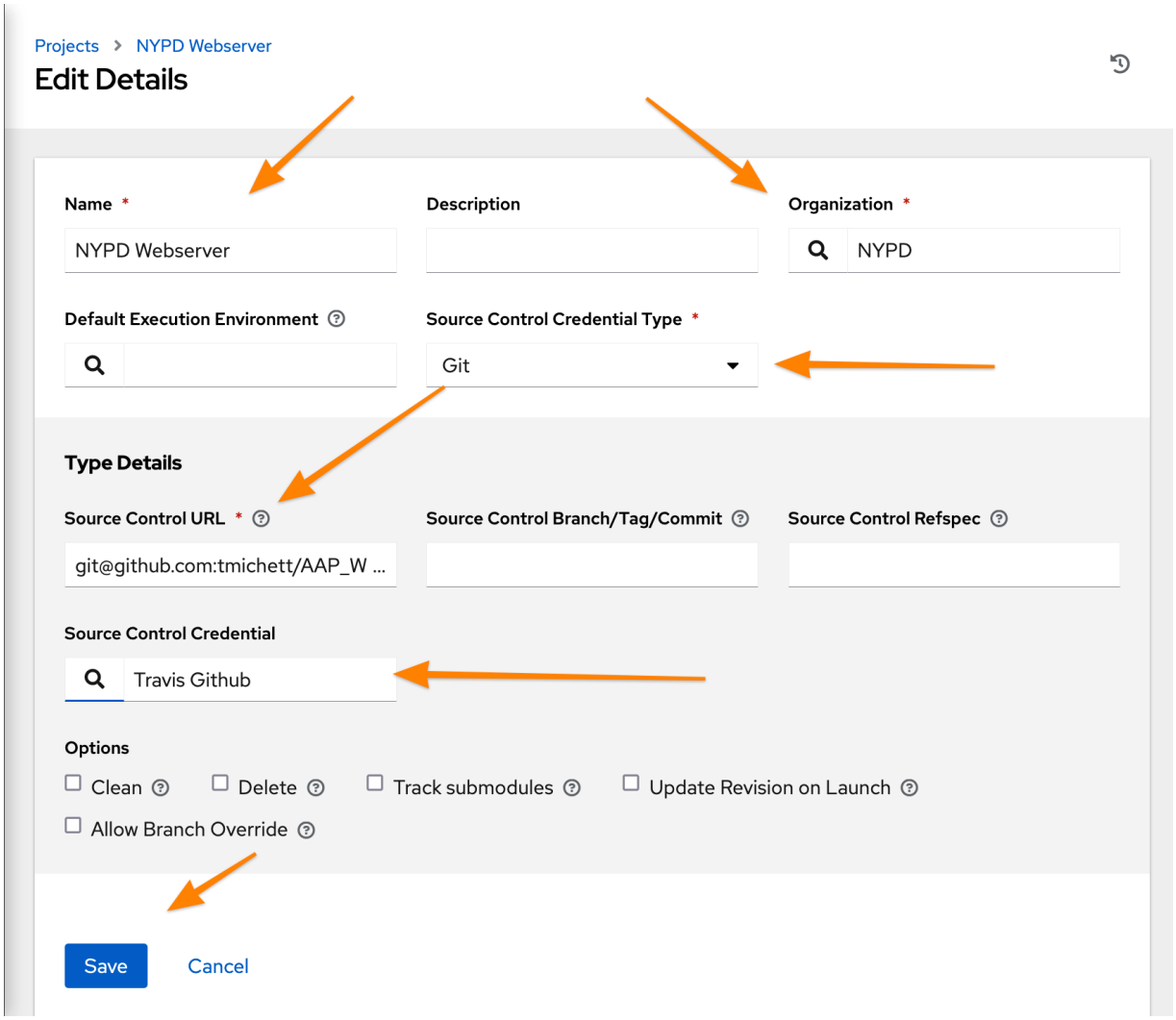


*Figure 27. Ansible Controller - Creating Project from Github Source*

*Creating a Job Template*

1. Click **Templates** and then click **Add** to Add (*Add job template) to create a job template.

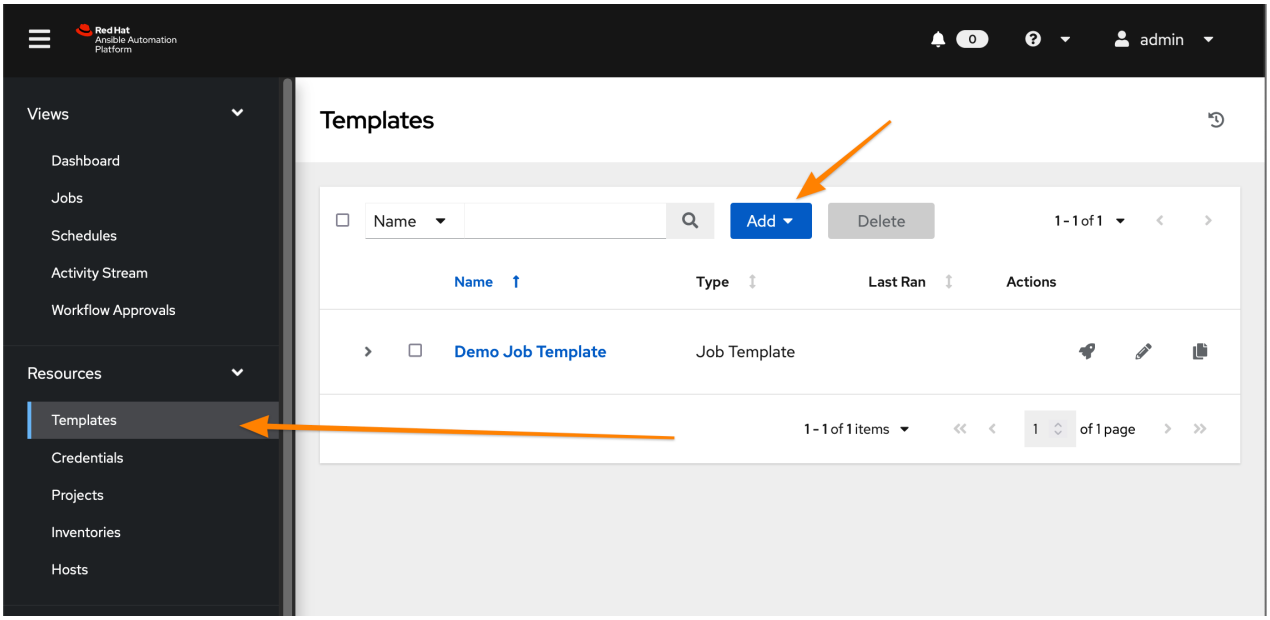


*Figure 28. Ansible Controller - Job Templates*

2. Create the new job template and then click **Save**
    - Provide a name
    - Provide job type (**run**)
    - Provide **Inventory**
    - Provide **Project**
    - Select **Playbook**
    - Select **Credentials**
    - Select **Privilege Escalation** option

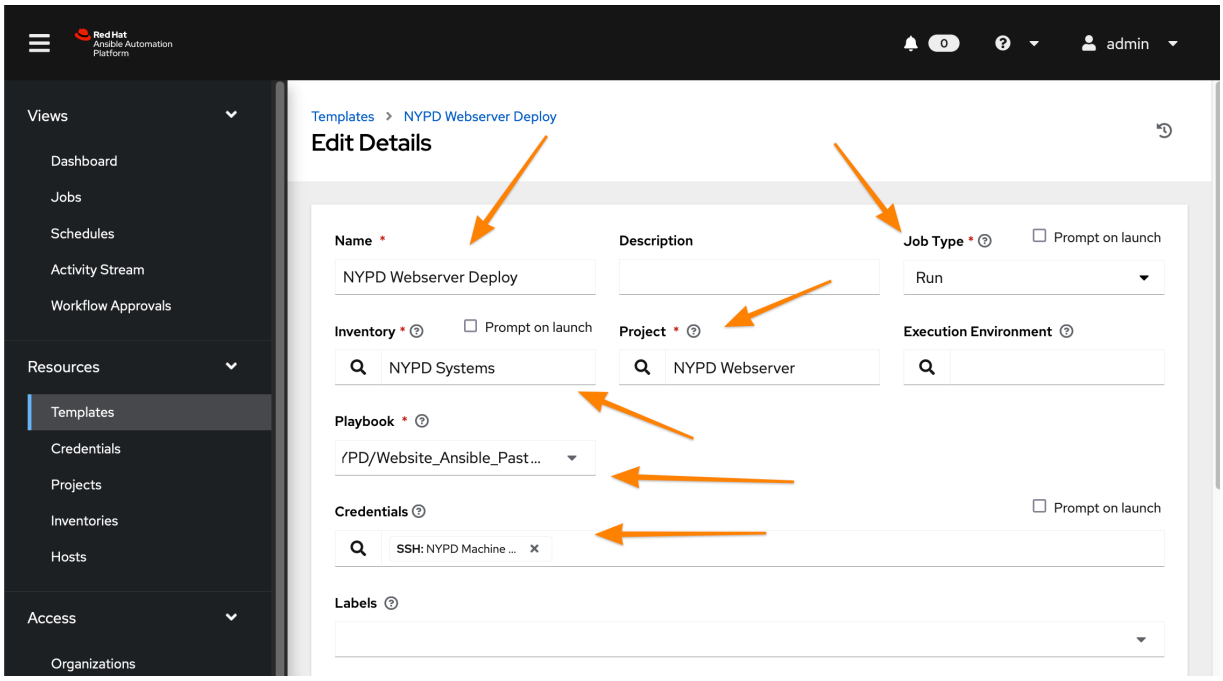


*Figure 29. Ansible Controller - Job Template Details*

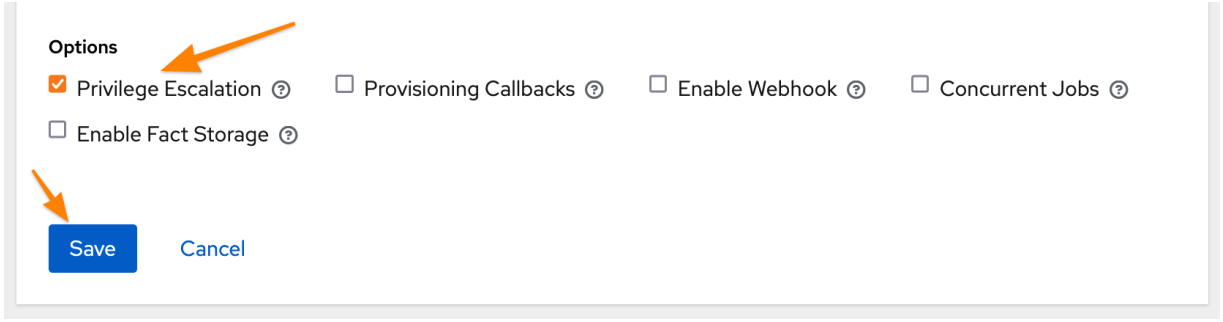


*Figure 30. Ansible Controller - Job Template Details cont.*

*Privilege Escalation*

It is important to know what the playbook does and whether it requires privilege escalation. A proper playbook might already have this defined, but it also allows you to assign it to the job from this menu.

3. Launch the job by clicking **Launch**

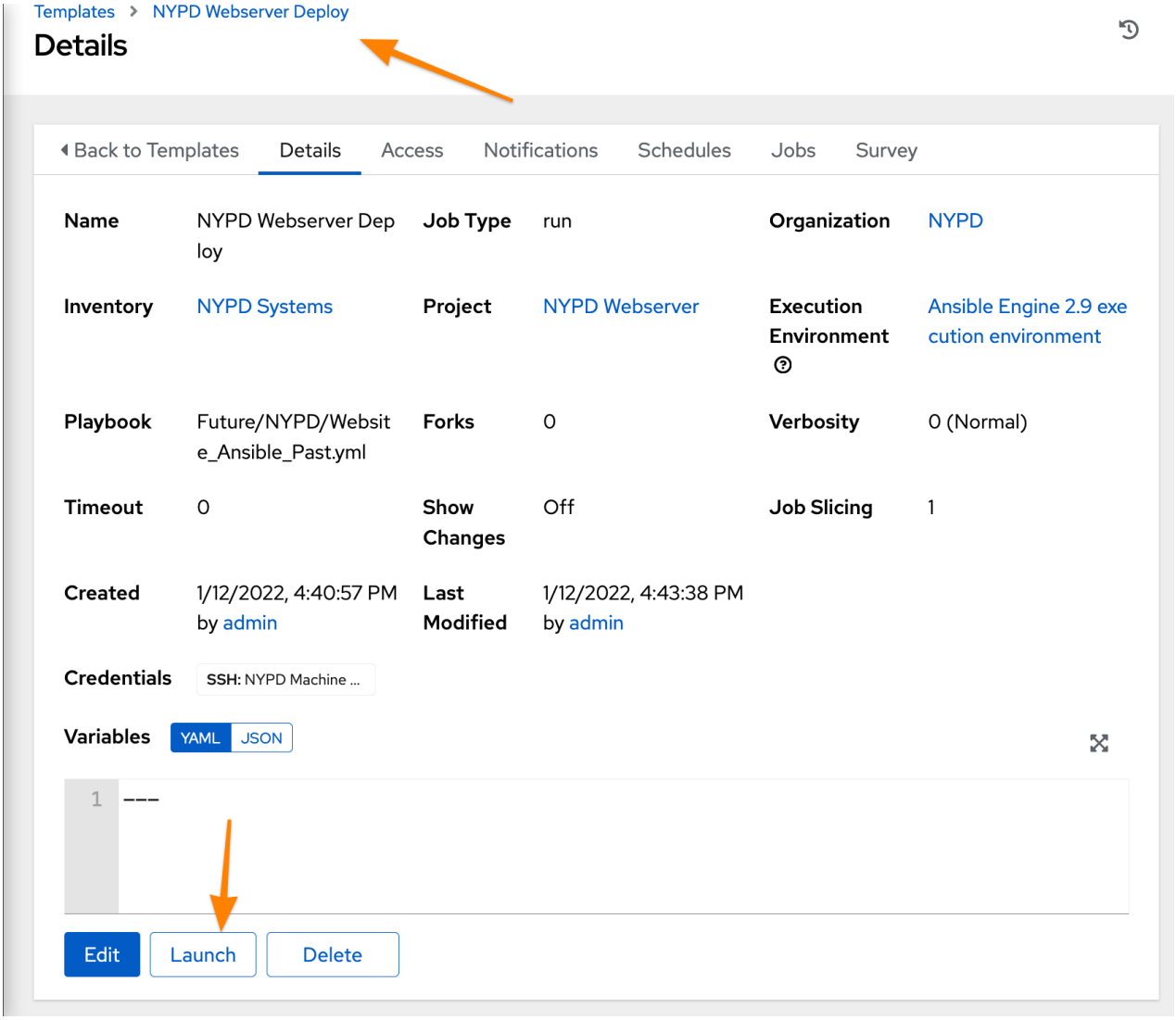


*Figure 31. Ansible Controller - Job Template Launch*

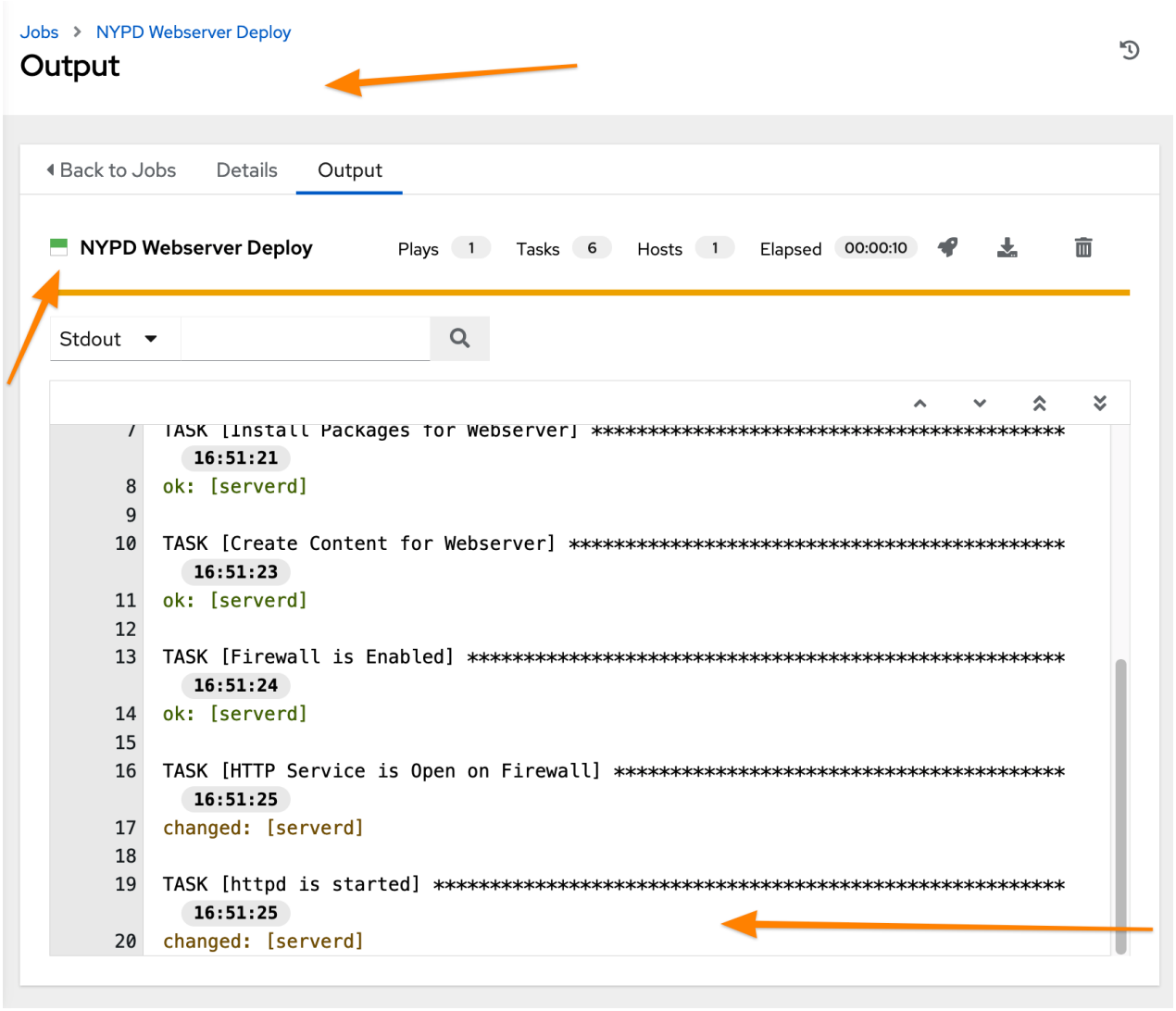


*Figure 32. Ansible Controller - Job Results Output Verification*

4. Verify webserver is running and accessible.

```
[student@workstation ~]$ curl serverd
I'm an awesome webserver for the NYPD and I know Castle!!
```

### 3.3.4. Workflows

In order to create job workflows, projects and existing job templates must already be created before they can be put together as a job workflow template.

*Example 10. DEMONSTRATION - Job Workflow Templates*

For this demonstration, it will be necessary to create two new **Job Templates** that will be linked together in a **Job Workflow Template**. We will be leveraging the already created project **NYPD Webserver** for existing playbooks and inventories. We will also create a dynamic inventory based on imported inventory from the project.

*Creating a Project-Based Inventory Source*

1. Login to **Ansible Controller**

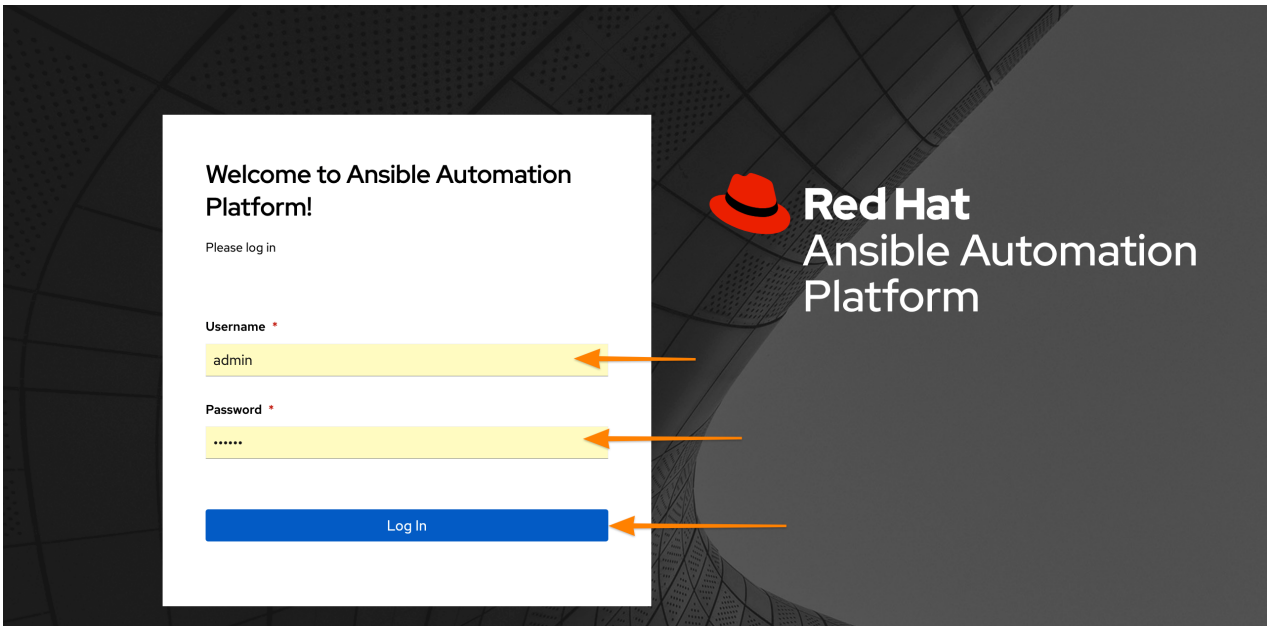


*Figure 33. Ansible Controller Login*

2. Click **Inventories** and then click **Add**

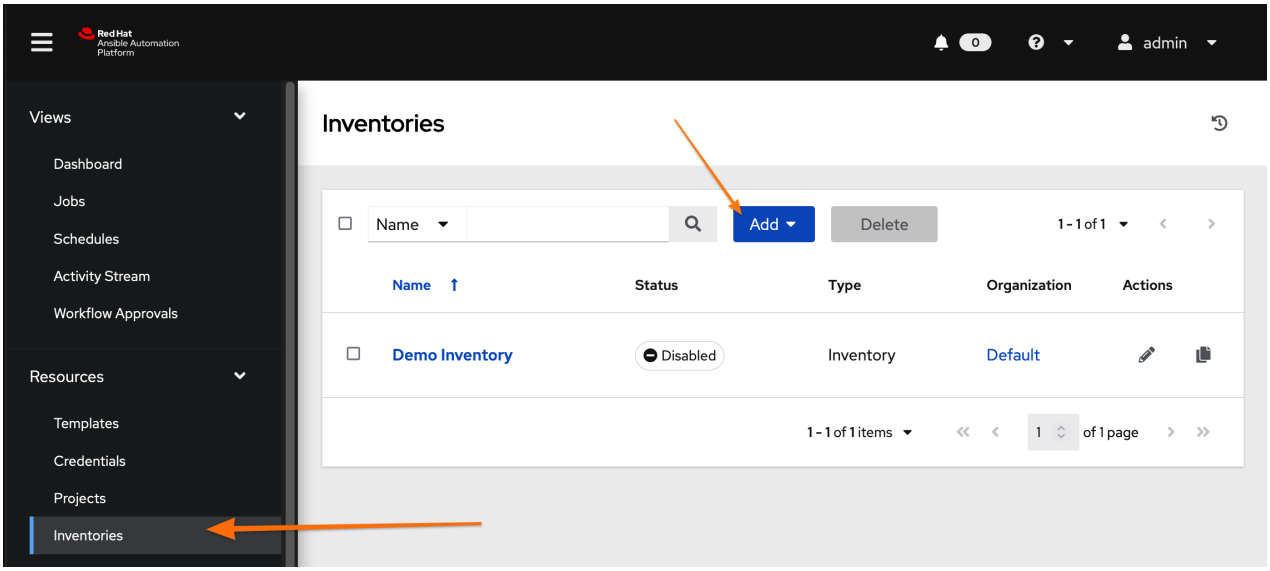


*Figure 34. Ansible Controller - Inventory*

3. Assign a **Name** and **Organization** to the Inventory and then click **Save**

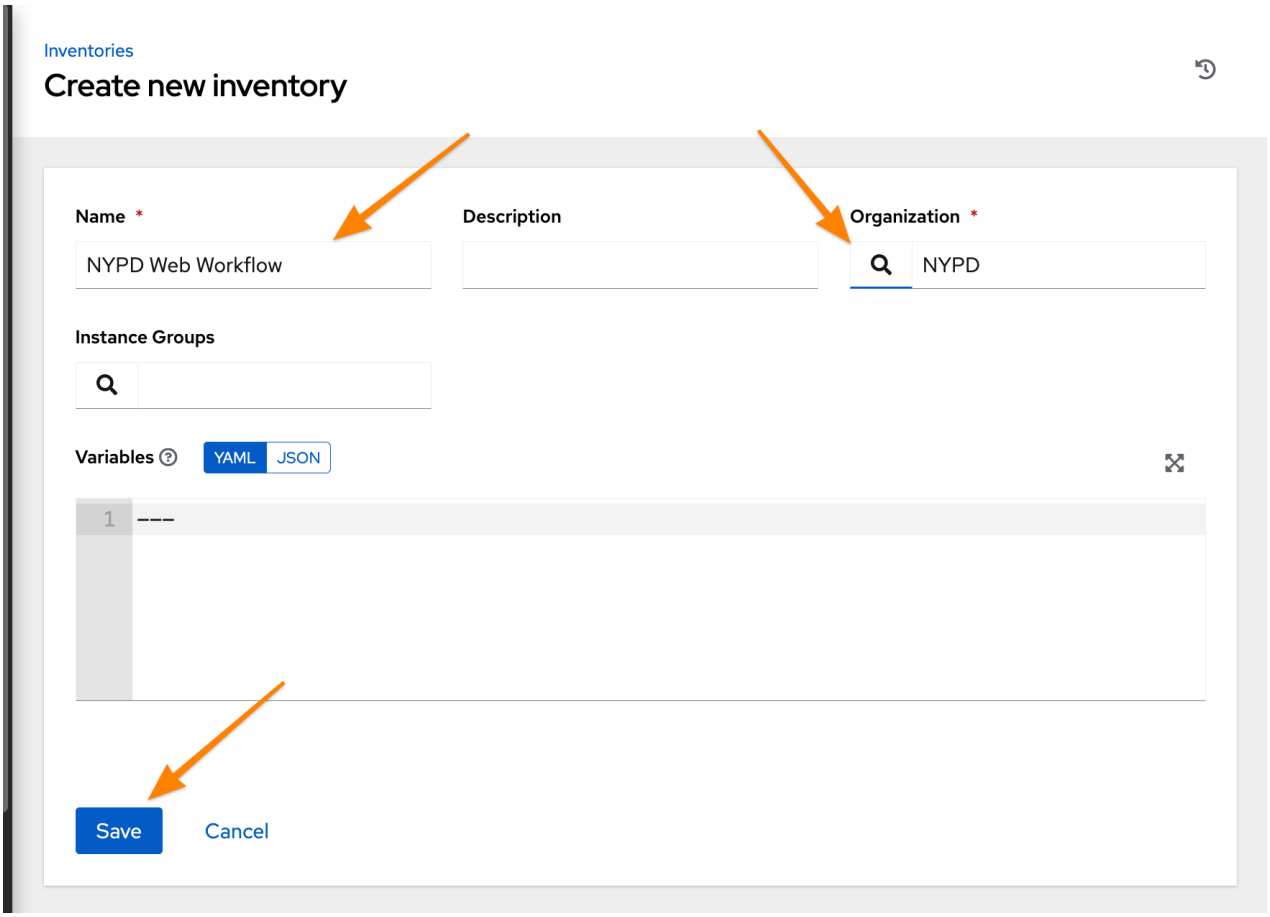


*Figure 35. Ansible Controller - Inventory Creation*

4. Click **Sources** to create an inventory source

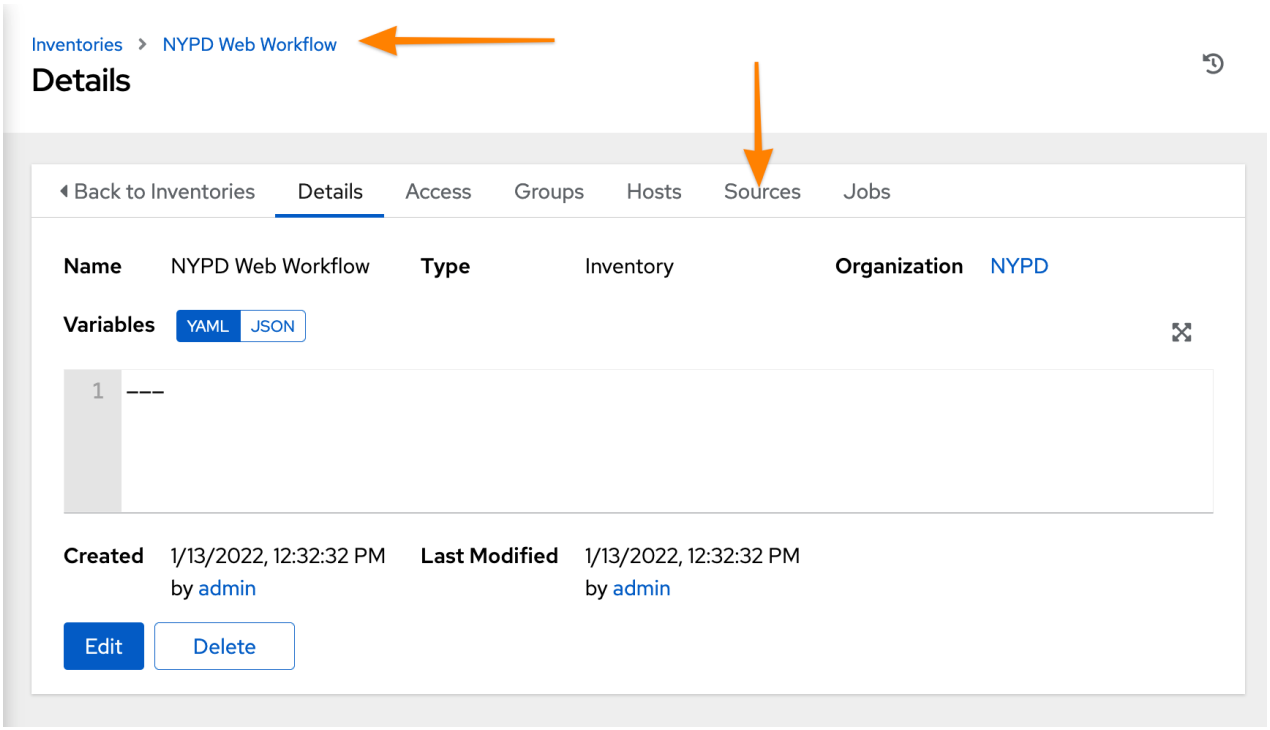


*Figure 36. Ansible Controller - Inventory Sources*

5. Click **Add** to add an inventory source

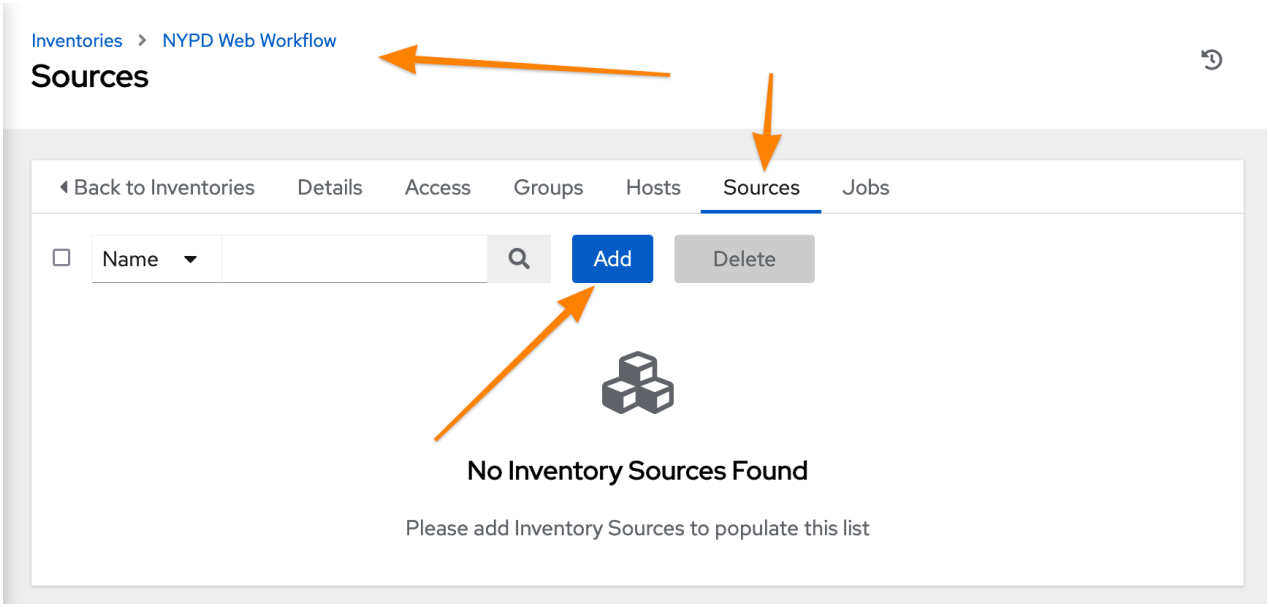


*Figure 37. Ansible Controller - Adding Inventory Sources*

6. Provide a **Name** and choose **Sourced from a Project** as source and click **Save**

a. Select the **Project** and **Inventory file**

b. Check **Update on launch**

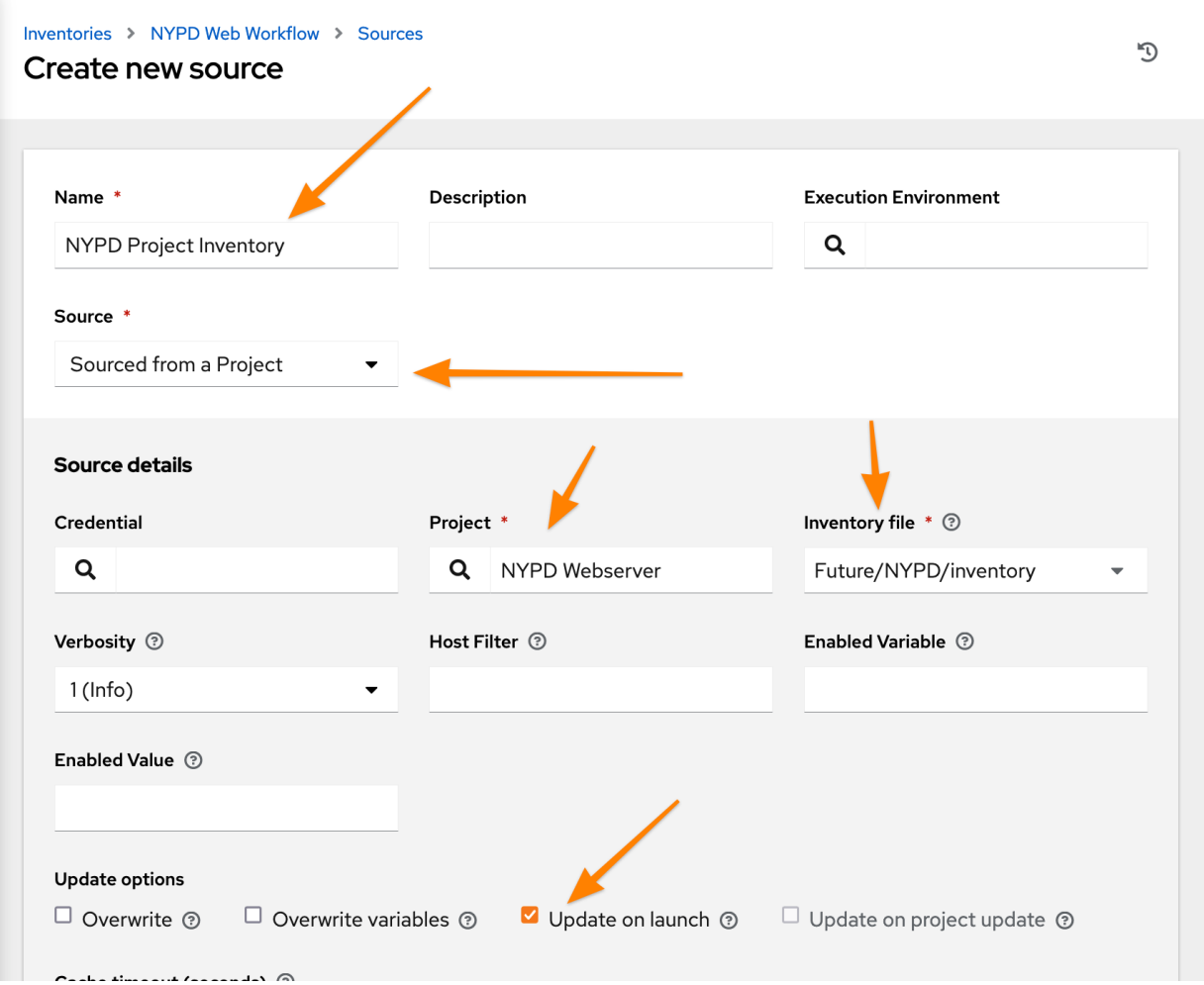


*Figure 38. Ansible Controller - Configuring Inventory Sources*

7. Click **Sync** to perform a synchronization

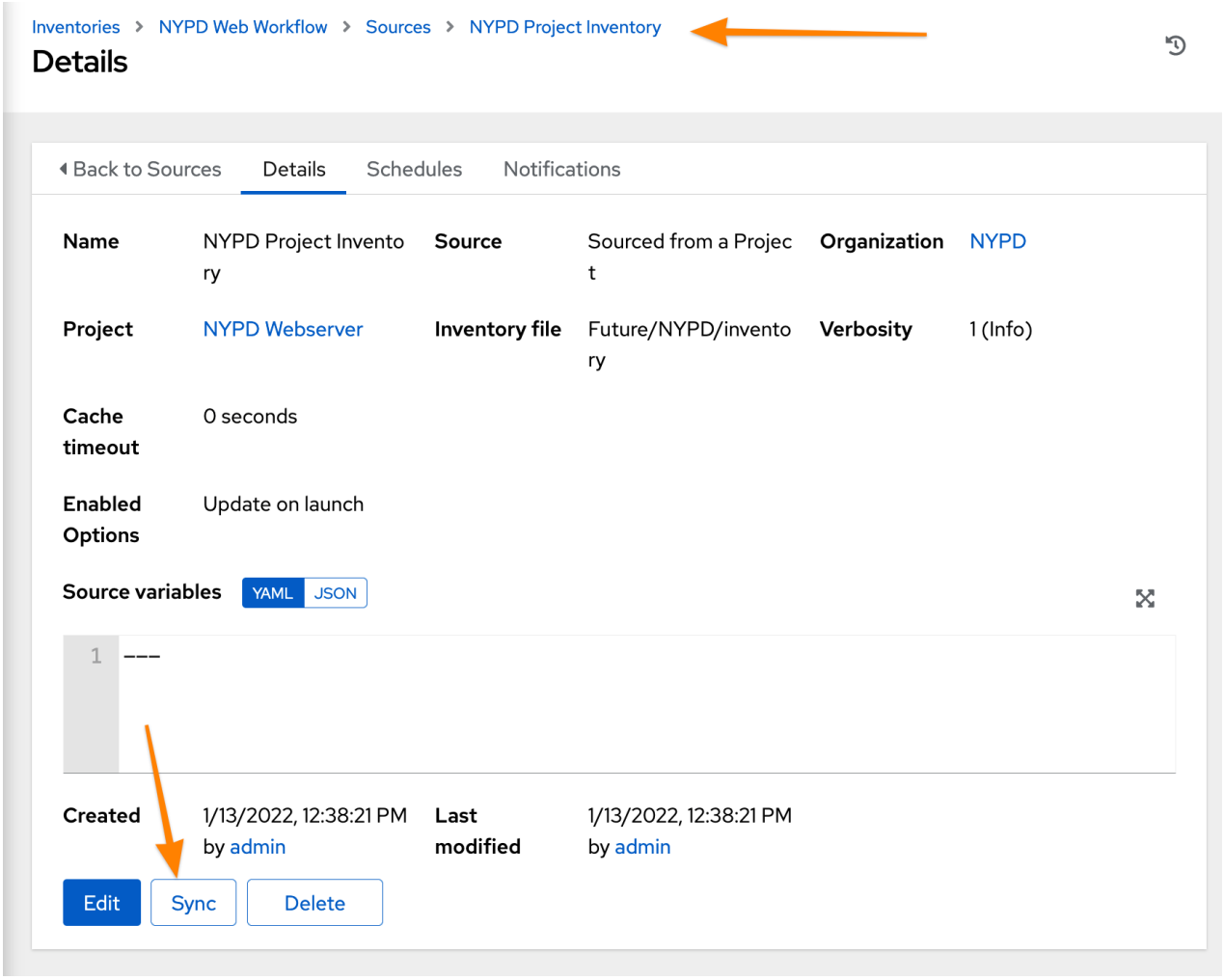


*Figure 39. Ansible Controller - Synchronizing Inventory Sources*

8. Click **Inventories** to verify the inventory and select **NYPD Web Workflow**

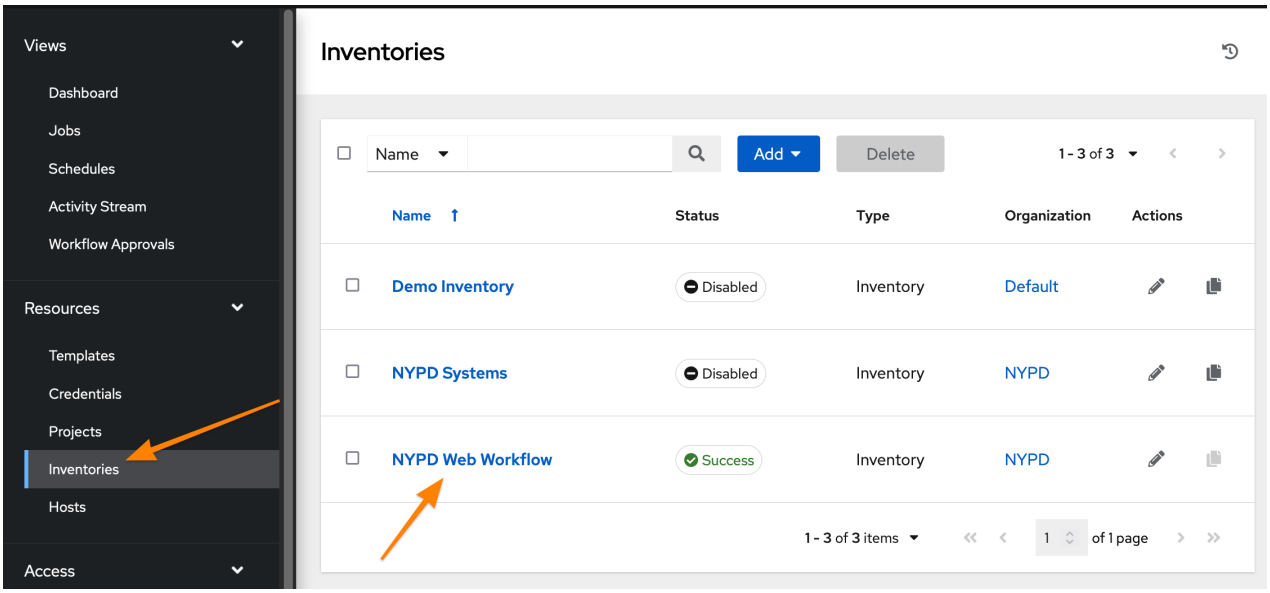


*Figure 40. Ansible Controller - Verifying Inventory Sources*

9. Click **Hosts** to view hosts

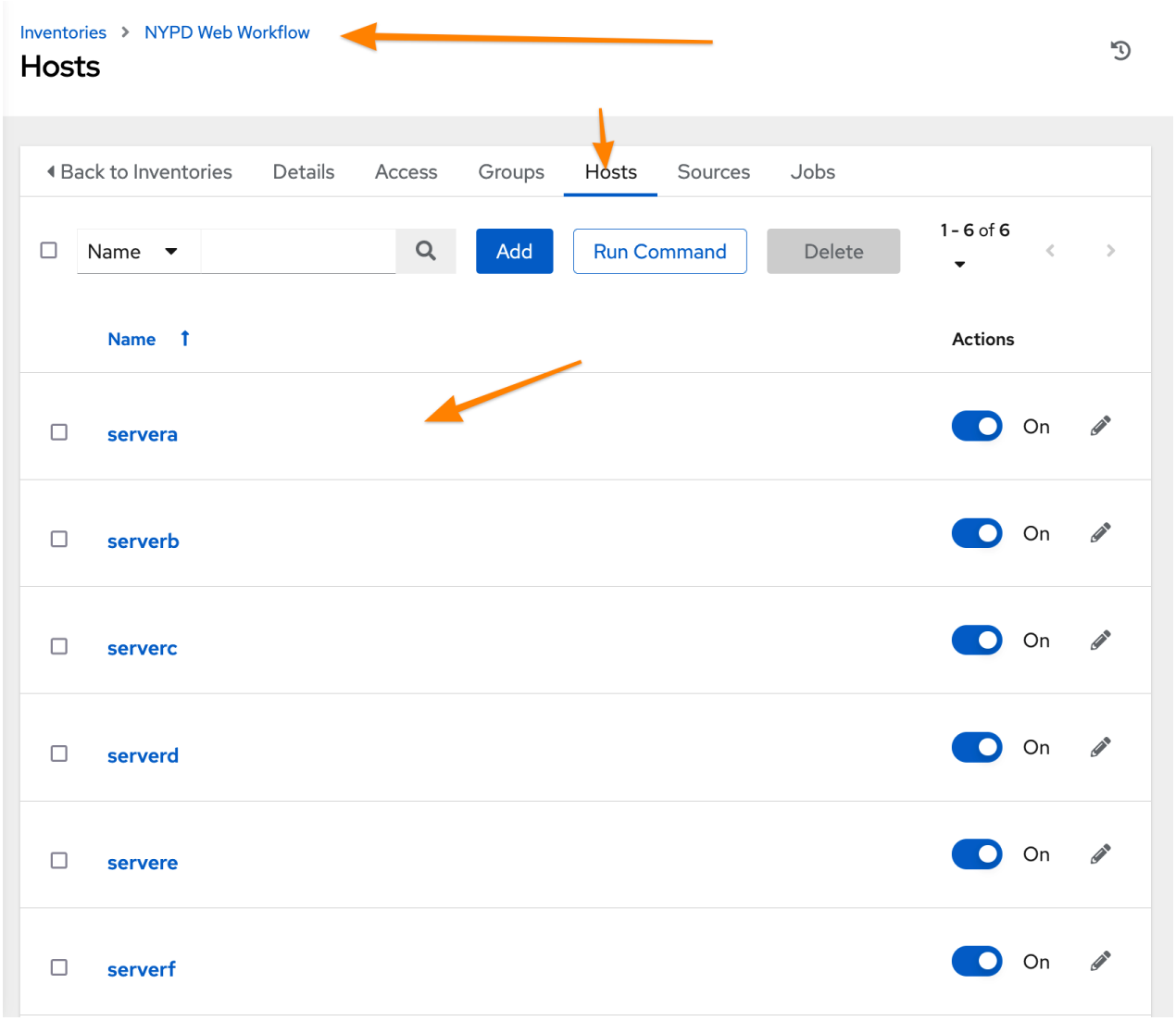


*Figure 41. Ansible Controller - Verifying Inventory Hosts from Project*

10. Click **Groups** to view host groups
    a. Click on a group name to see hosts in group and click **Hosts**

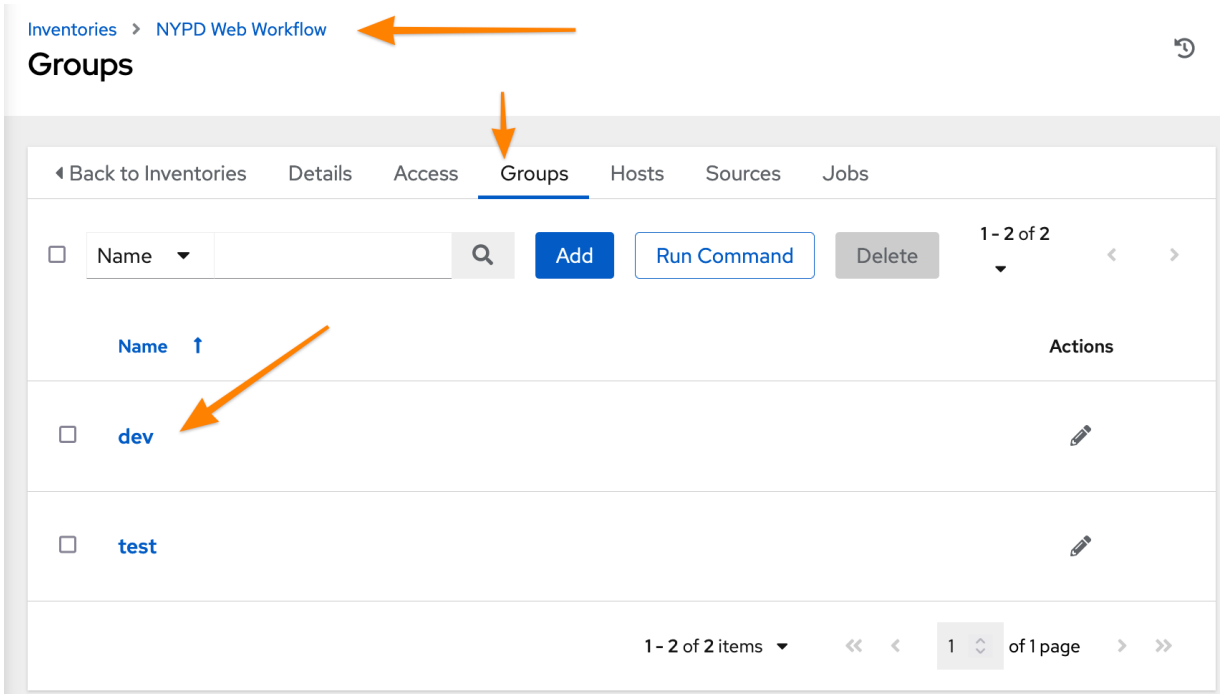


*Figure 42. Ansible Controller - Verifying Inventory Group from Project*

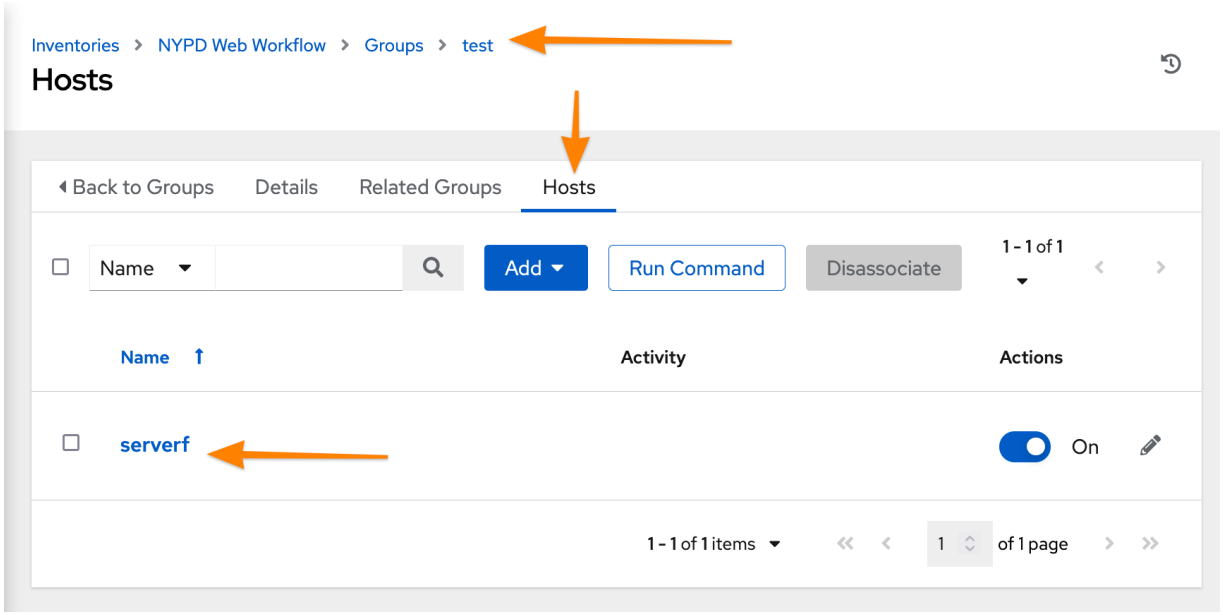


*Figure 43. Ansible Controller - Verifying Inventory Group* **(Hosts)** *from Project*

*Project Based Inventory*

The above example shows how to create a dynamic inventory that is sourced from a project. This would can be done to ensure that you have the same inventory and host systems as the developers. It is not 100% necessary to have inventory in the projects, but some people prefer to keep host inventory in projects and this is a great method in keeping developer inventory in sync with what is stored in Ansible Controller.

*Creating a Job Workflow Template*

1. Login to **Ansible Controller**

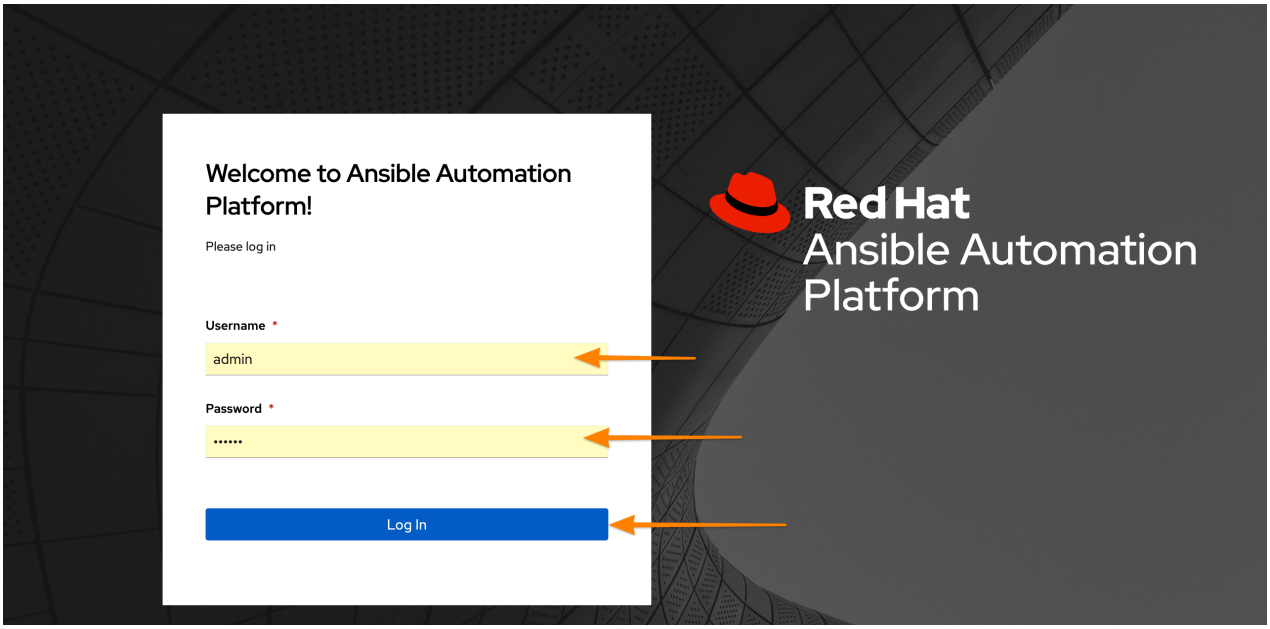


*Figure 44. Ansible Controller Login*

2. Click **Templates** and then click **Add** and **(Add job template)** to create a new Job Template

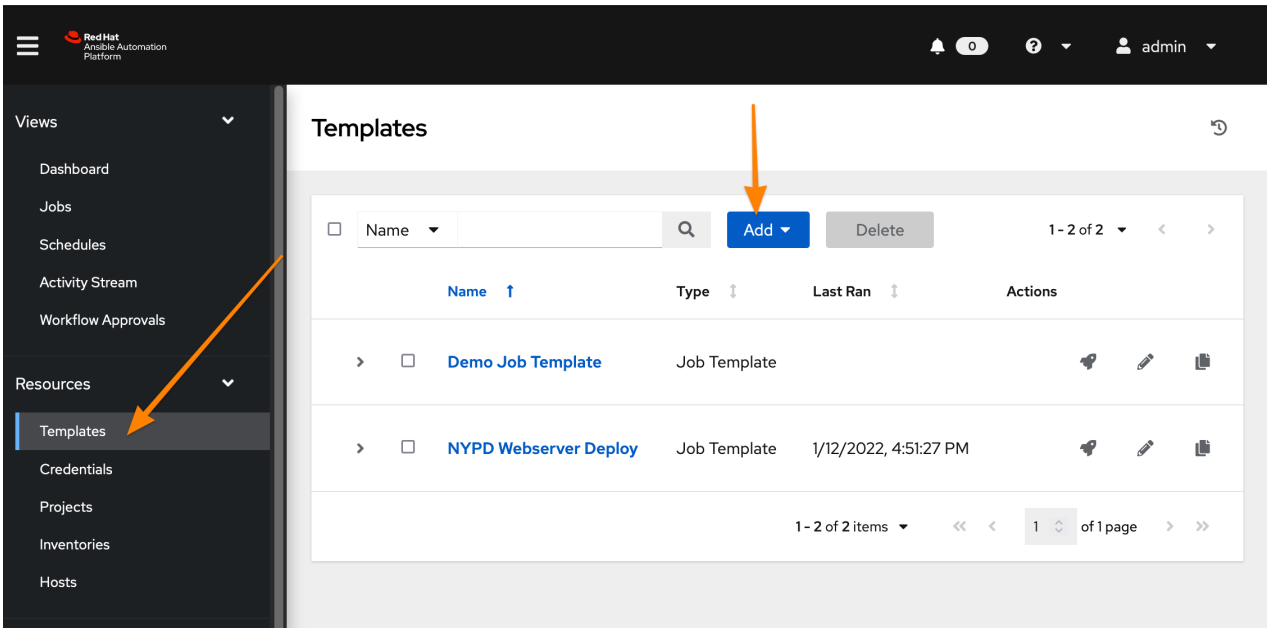


*Figure 45. Ansible Controller - Job Templates*

3. Complete the form for the NYPD Dev Webserver and click **Save**
    a. **Name**: *NYPD Dev Webserver*
    b. **Job Type**: *run*
    c. **Inventory**: *NYPD Web Workflow*
    d. **Project**: *NYPD Webserver*
    e. **Playbook**: *Future/NYPD/NYPD_Web_Workflow.yml*
    f. **Credentials**: *NYPD Machine SSH Creds*
    g. **Variables**: *inv_host_var: servere*
    h. **Privilege Escalation**: *Checked*

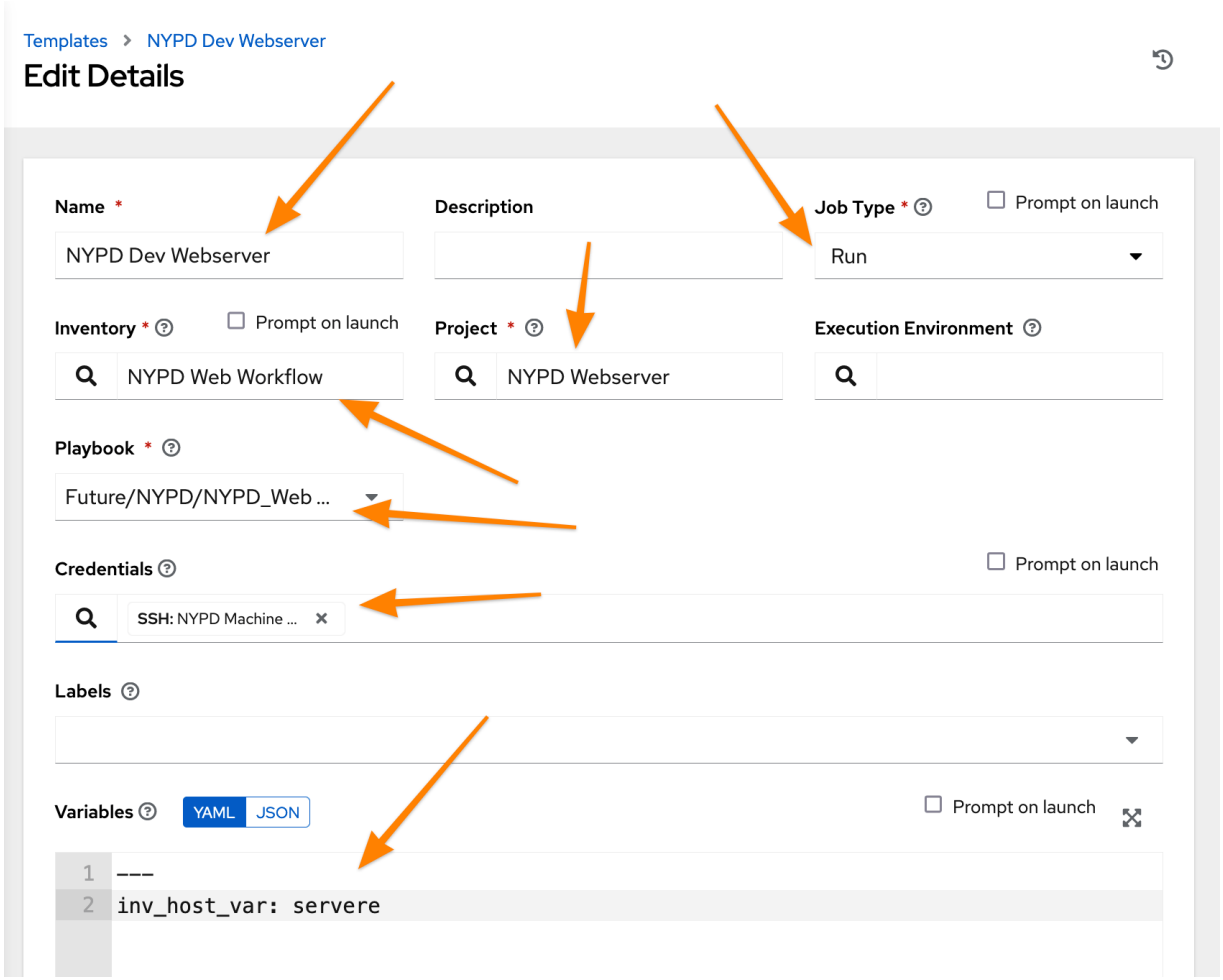


*Figure 46. Ansible Controller - Job Template Parameters*

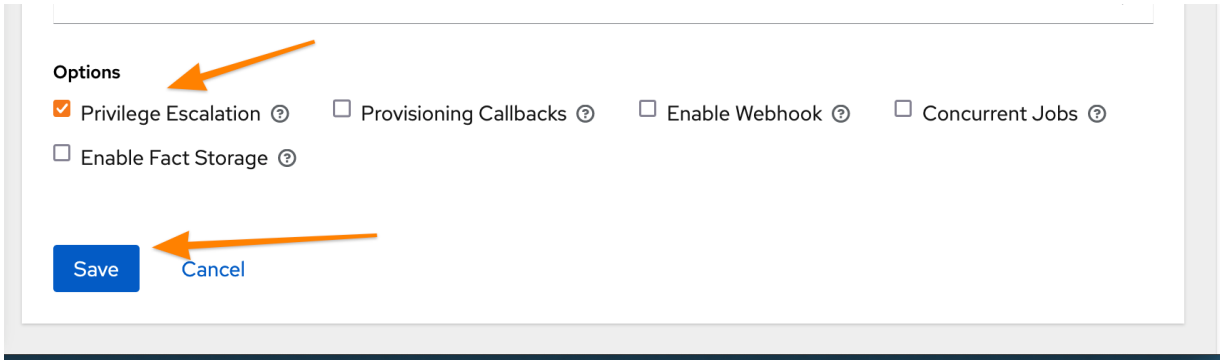


*Figure 47. Ansible Controller - Job Template Parameters cont.*

4. Create a new Job template using steps above with the following values.
    a. **Name**: *NYPD Test Webserver*
    b. **Job Type**: *run*

c. **Inventory**: *NYPD Web Workflow*

d. **Project**: *NYPD Webserver*

e. **Playbook**: *Future/NYPD/NYPD_Web_Workflow.yml*

f. **Credentials**: *NYPD Machine SSH Creds*

g. **Variables**: *inv_host_var: serverf*

h. **Privilege Escalation**: *Checked*

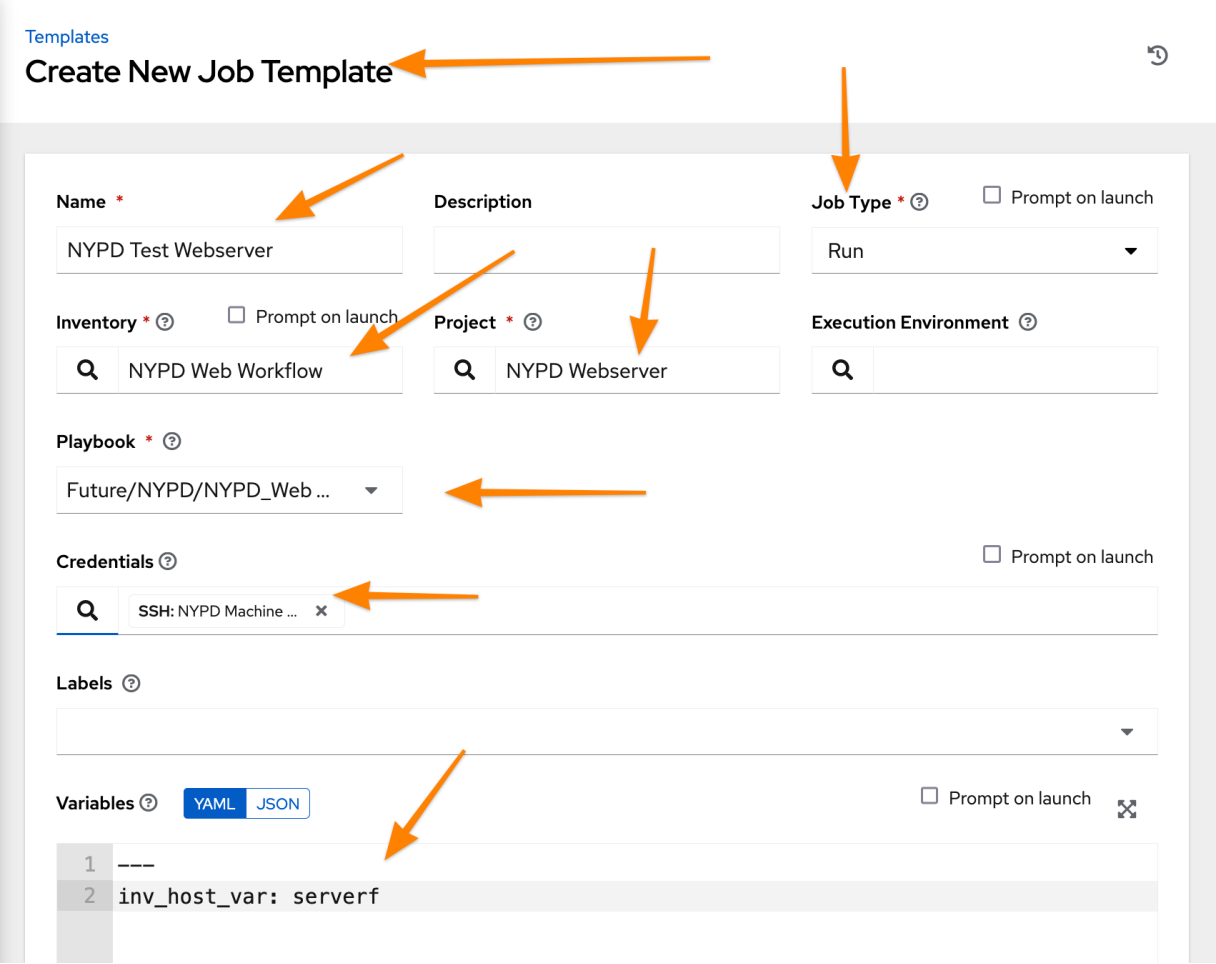


*Figure 48. Ansible Controller - Job Template Parameters for NYPD Test Webserver*

5. Click **Templates** then click **Add** and select **Add workflow template**

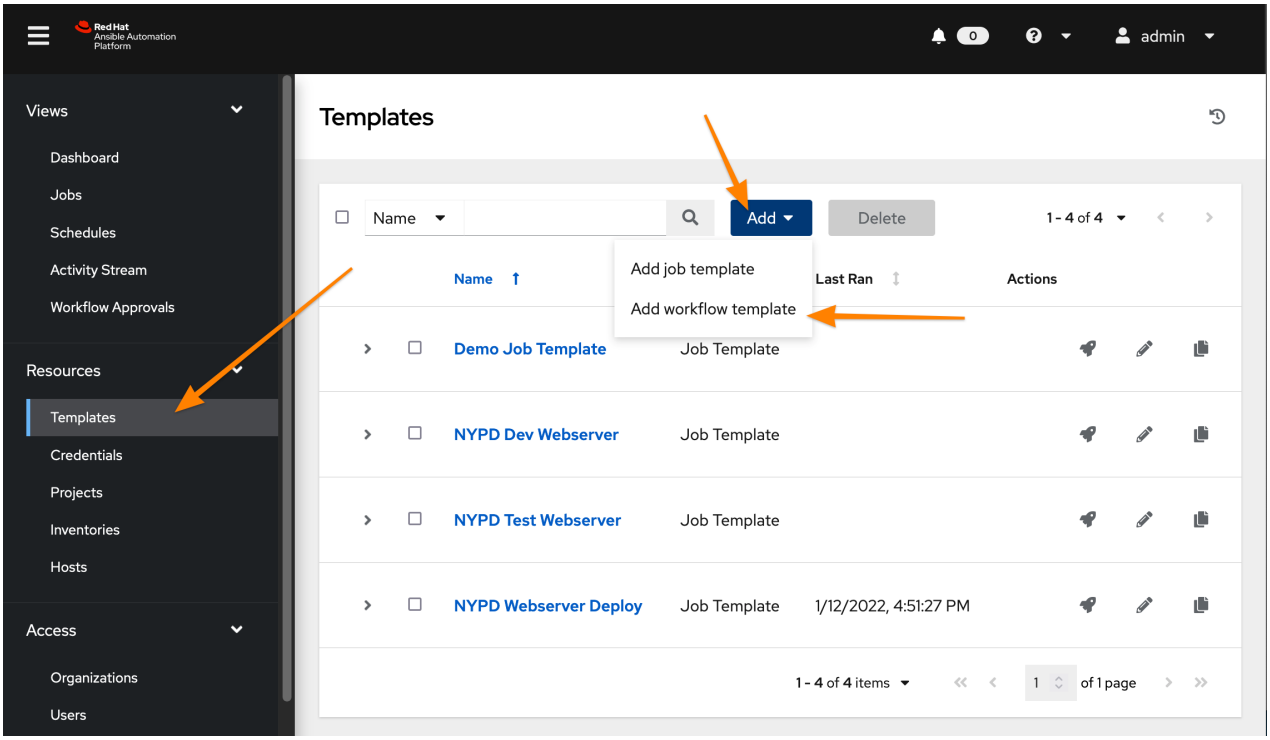


*Figure 49. Ansible Controller - Job Workflow Template*

6. Provide a **Name** and select the appropriate items
    a. **Inventory**: *Leave Blank* (Will use inventory specified for Job Templates)
    b. **Organization**: *NYPD*

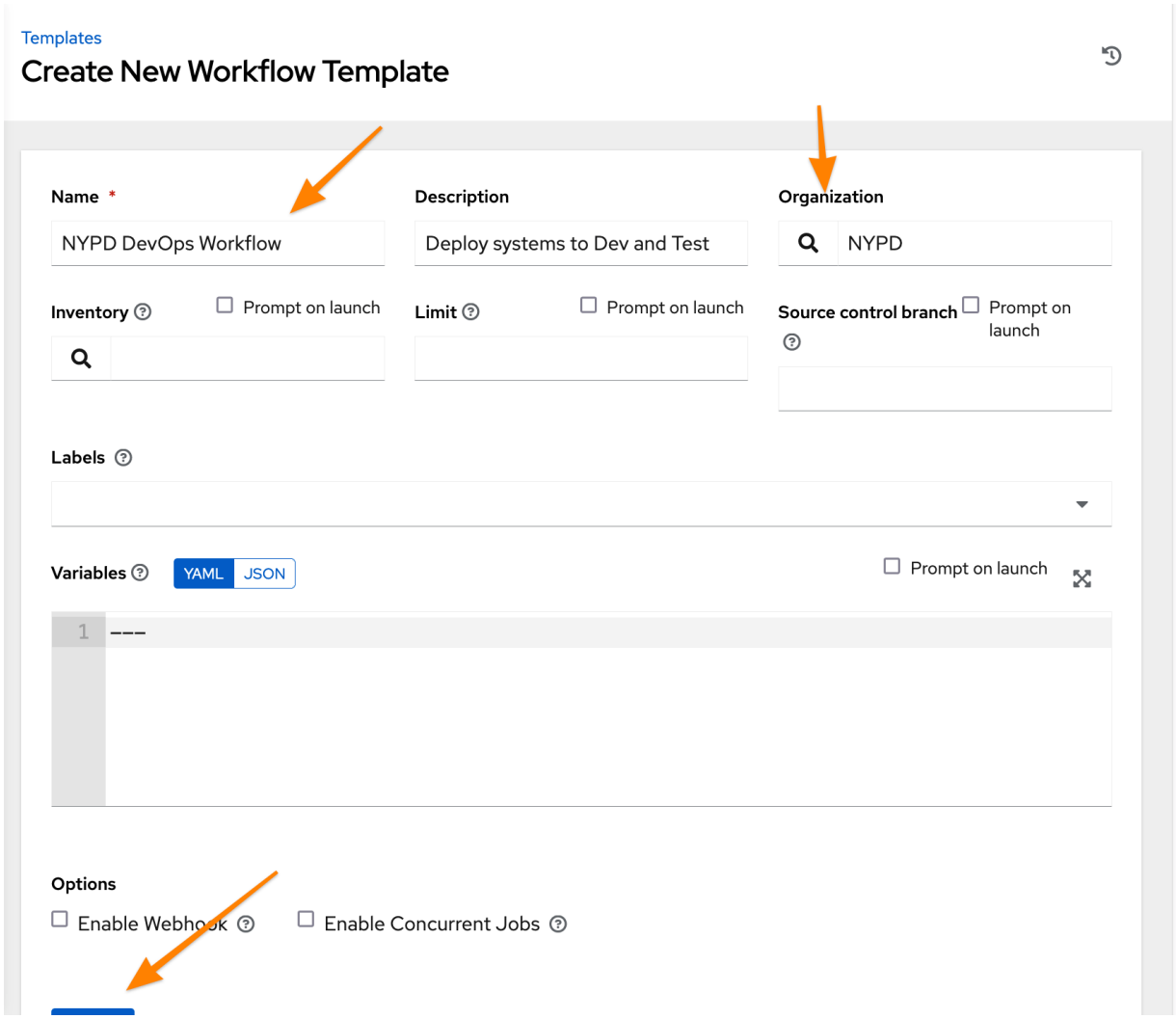


*Figure 50. Ansible Controller - Job Workflow Template Details*

7. The **Workflow Visualizer** will open and click **Start** to define first task in workflow

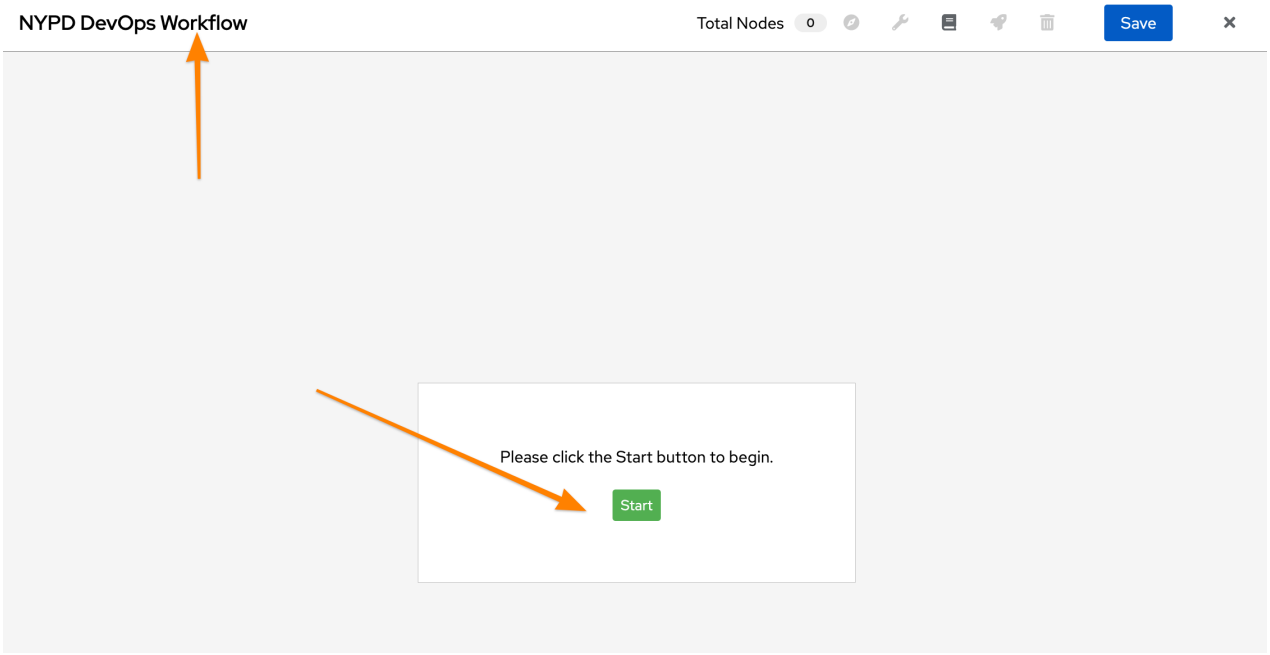


*Figure 51. Ansible Controller - Job Workflow Visualizer*

8. Start by selecting **Node Type** of **Project Sync** and select the **NYPD Webserver** Project then click **Save**

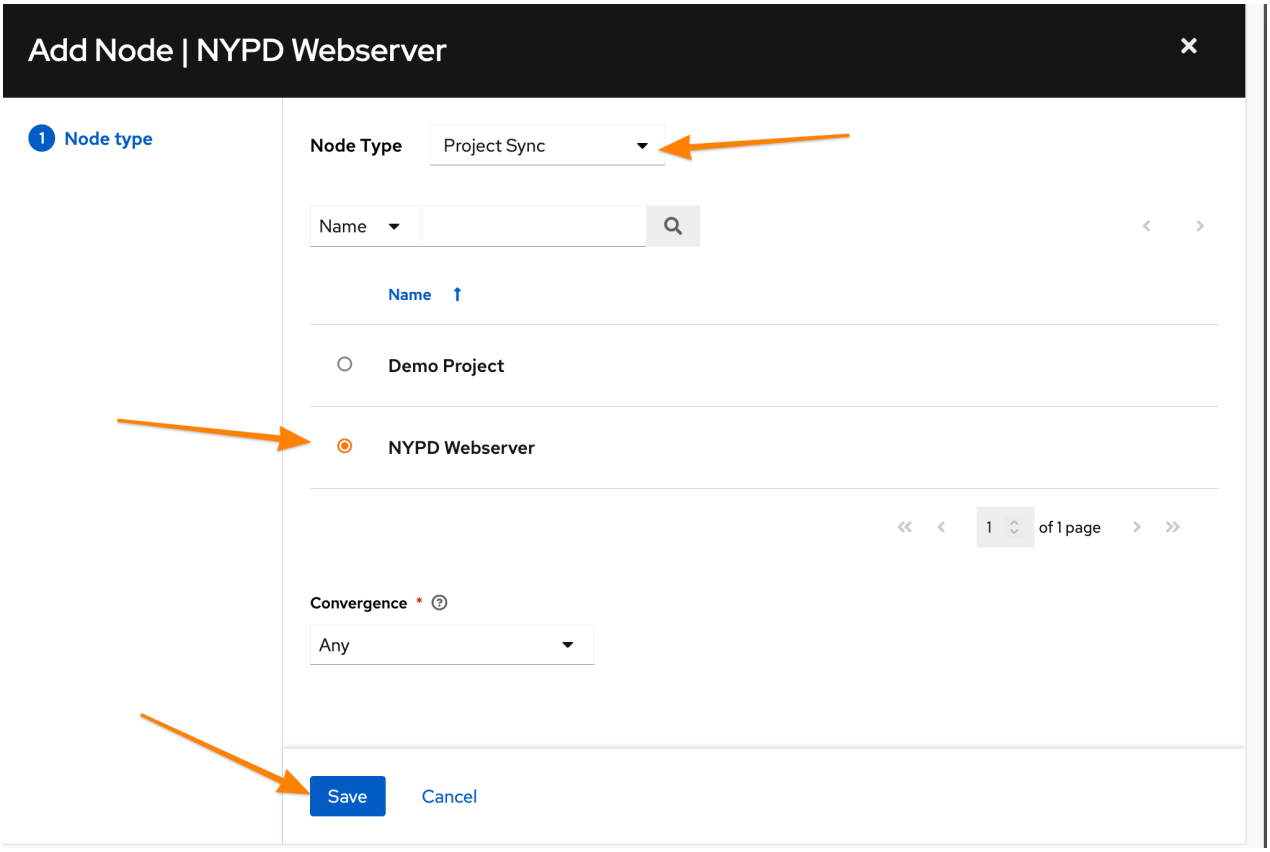


*Figure 52. Ansible Controller - Job Workflow Task #1 Synchronize Project Data*

9. Add Step #2 to run on **Success** by selecting the NYPD Webserver Project and clicking the **+**.
   a. Select **Run** and **On Success** then click **Next**
   b. Select **Node Type** to be **Job Template** and select the **NYPD Dev Webserver** then click **Save**

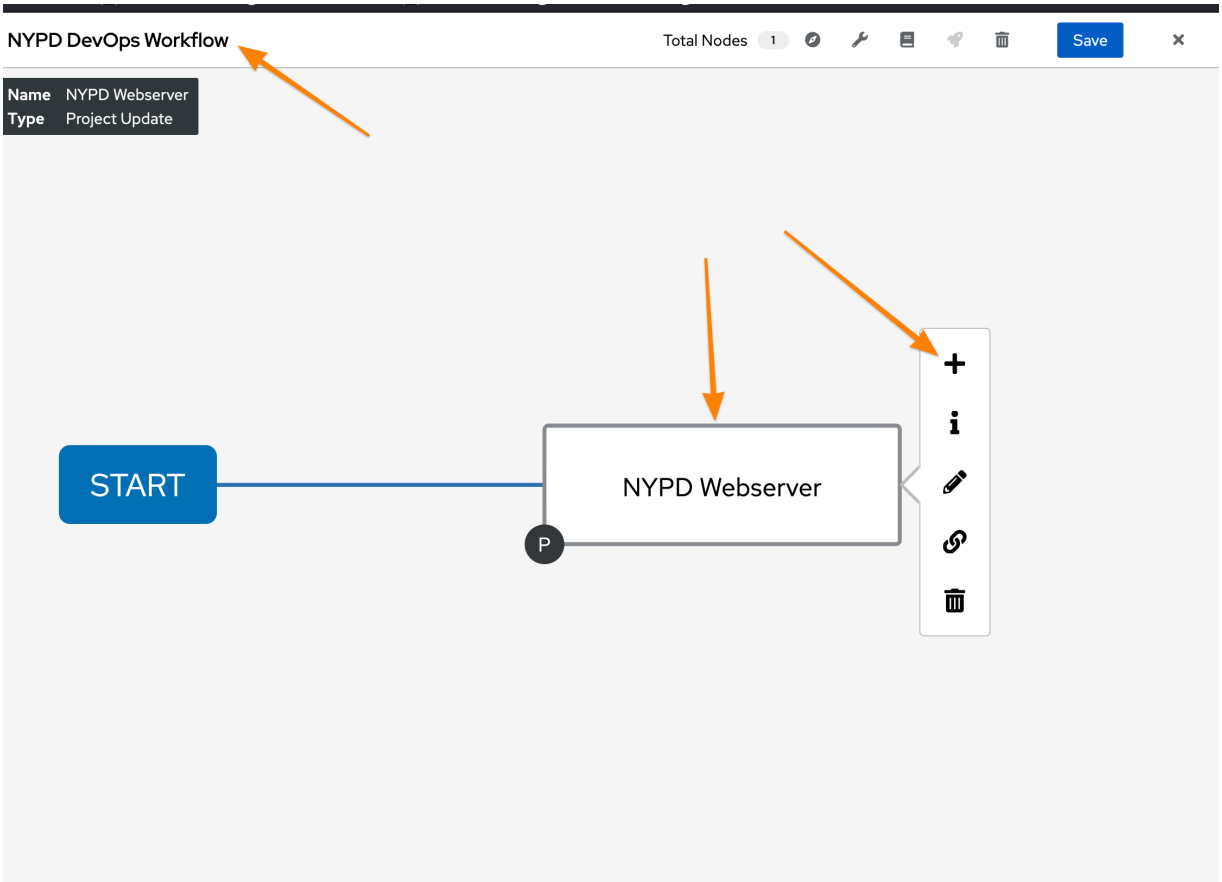


*Figure 53. Ansible Controller - Job Workflow Task #2 Launch Development Job*

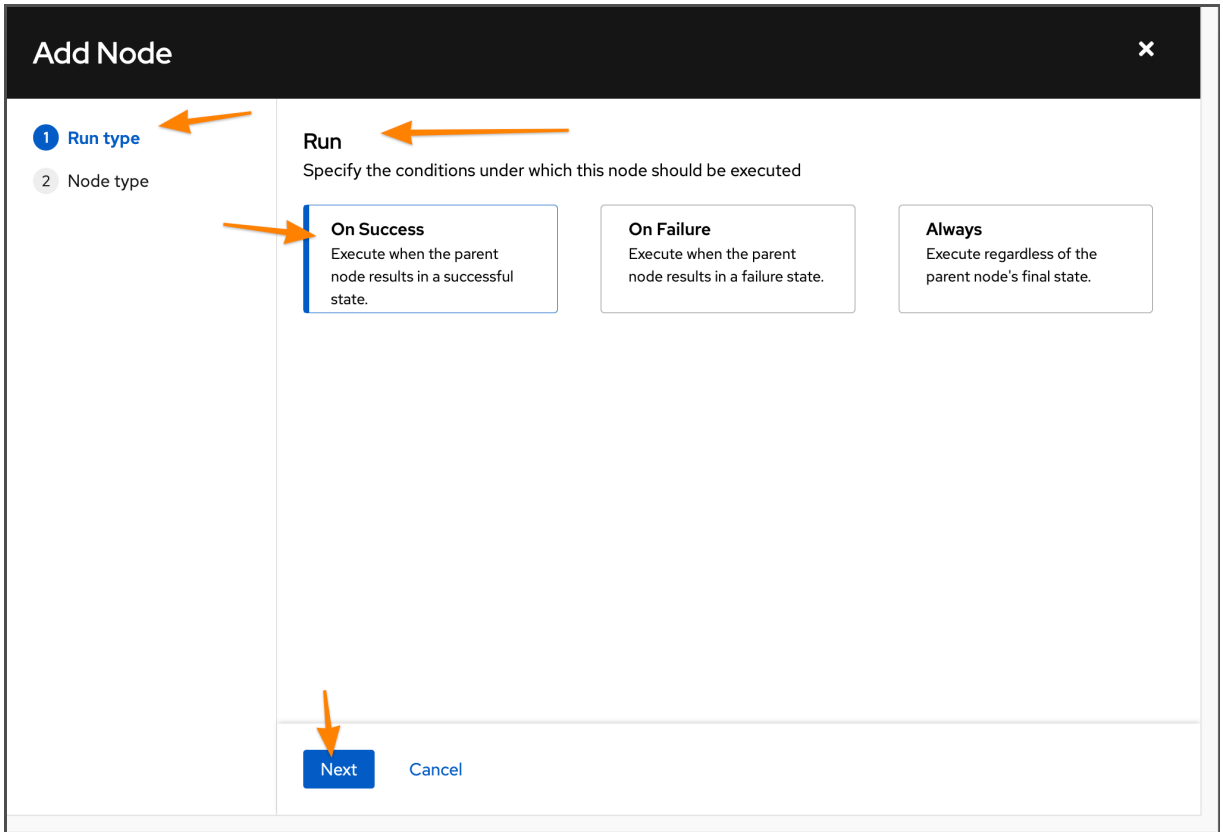


*Figure 54. Ansible Controller - Job Workflow Task #2 Selecting Job Run Parameters*

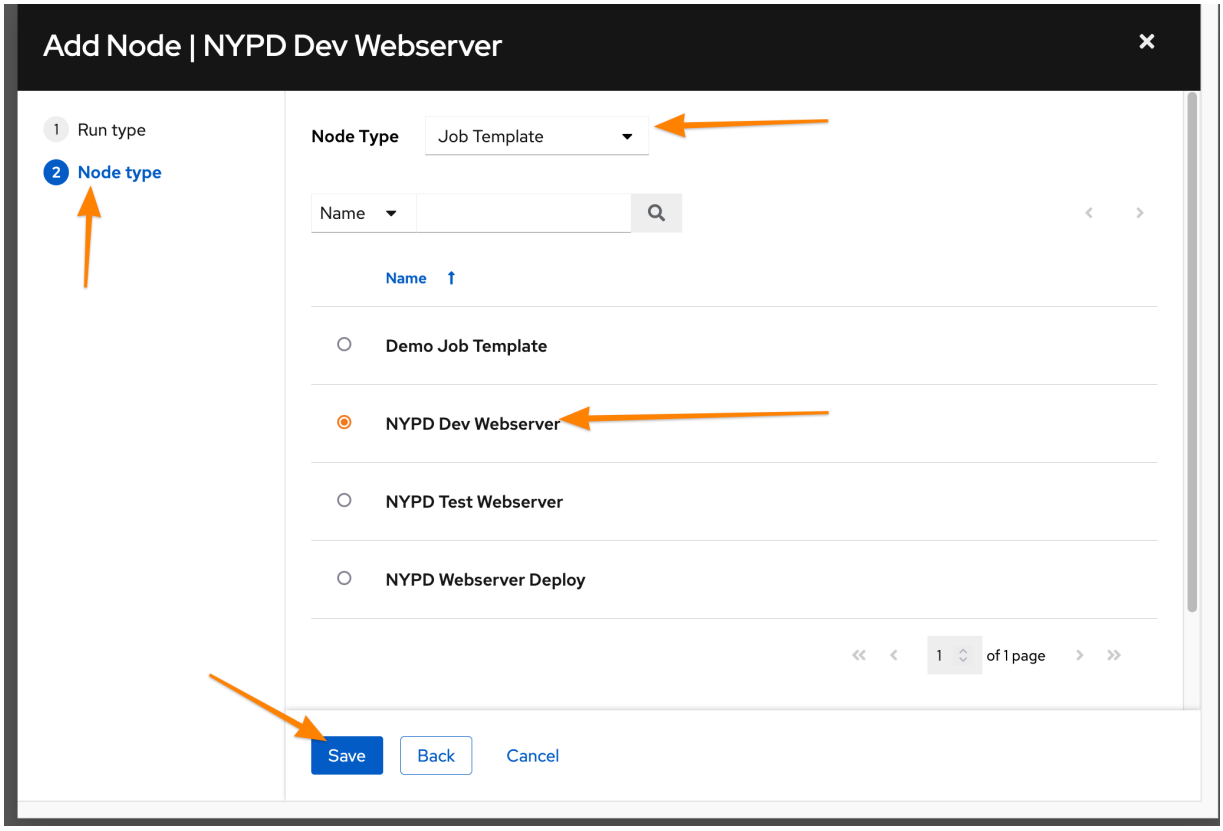


*Figure 55. Ansible Controller - Job Workflow Task #2 Selecting Job Template to Run*

10. Add Step #3 to run on **Success** by selecting the NYPD Webserver Project and clicking the **+**.
    a. Select **Run** and **On Success** then click **Next**
    b. Select **Node Type** to be **Job Template** and select the **NYPD Test Webserver** then click **Save**

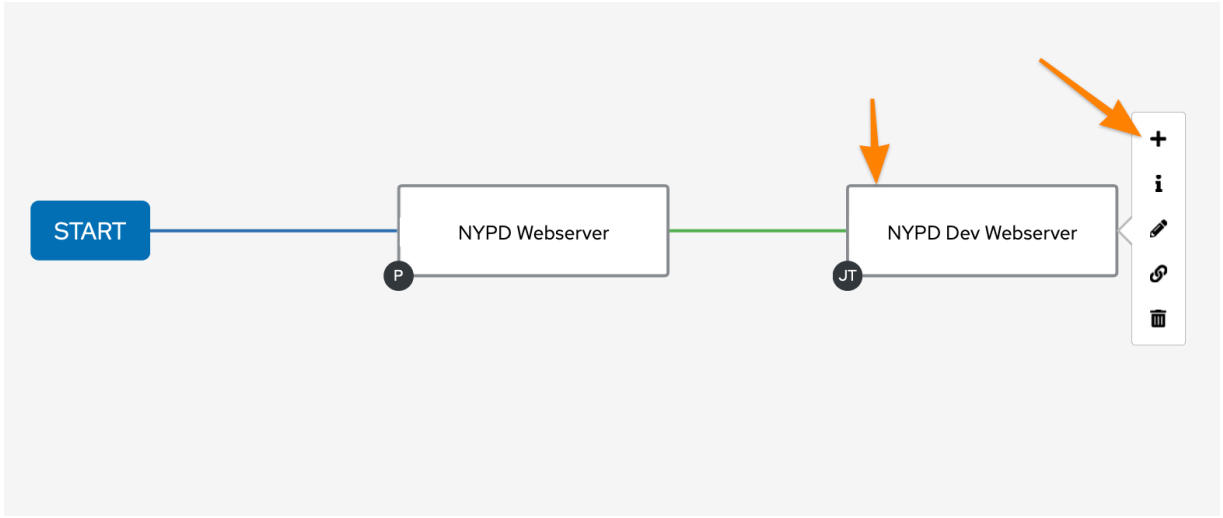


*Figure 56. Ansible Controller - Job Workflow Task #3 Launch Development Job*

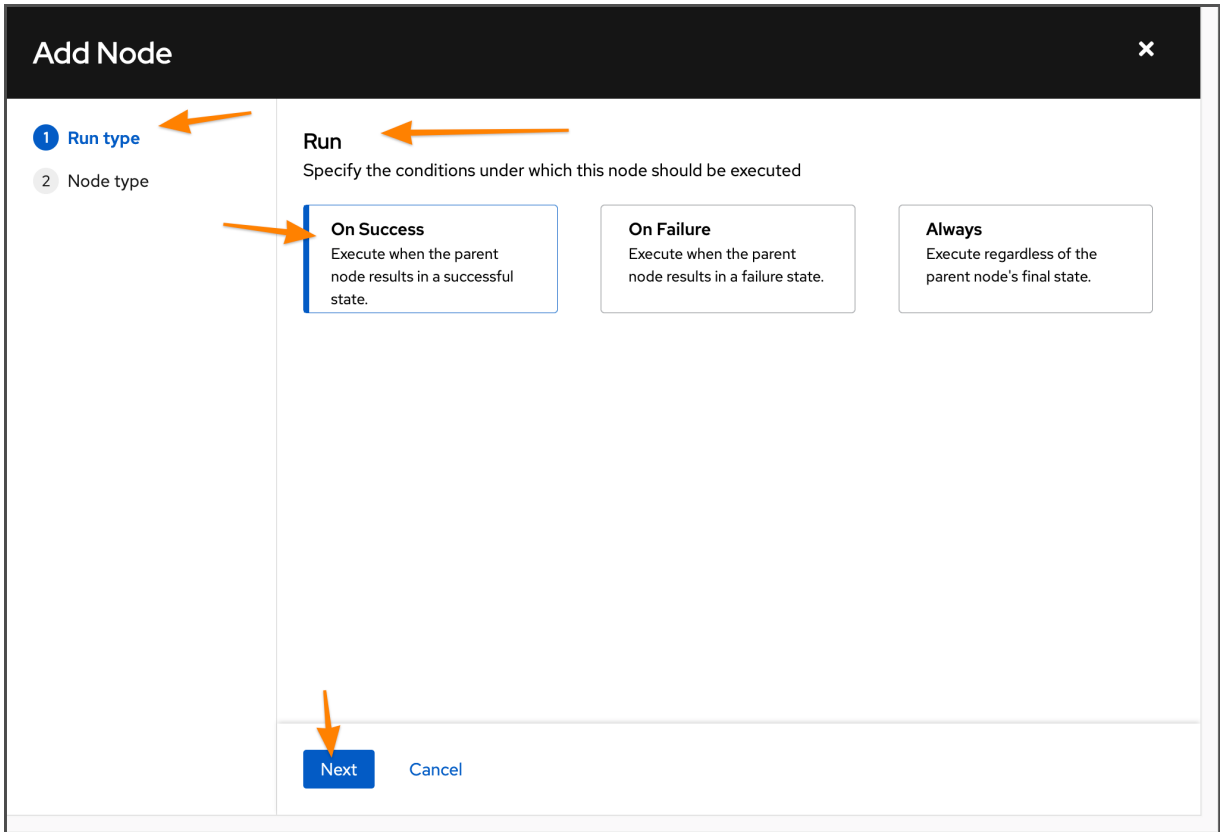


*Figure 57. Ansible Controller - Job Workflow Task #3 Selecting Job Run Parameters*

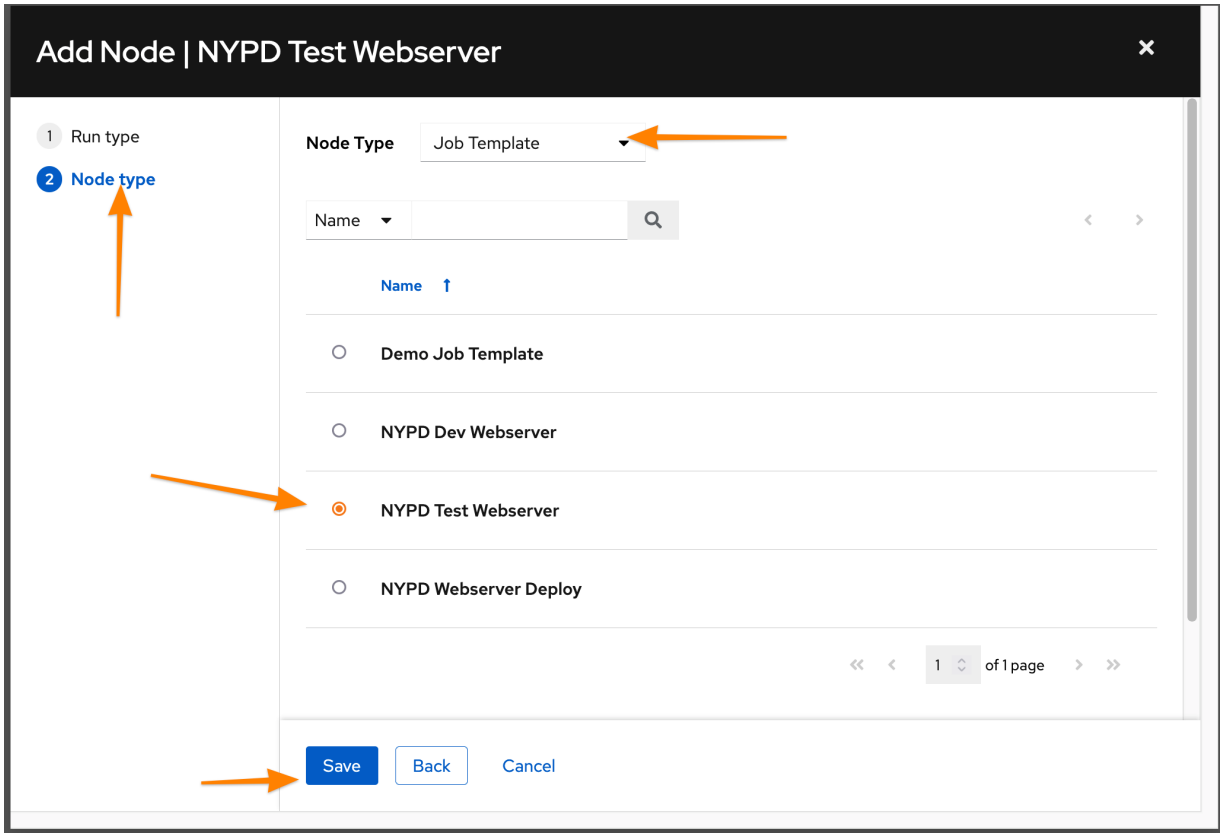


*Figure 58. Ansible Controller - Job Workflow Task #3 Selecting Job Template to Run*

11. View the complete Job Workflow and click **Save** when done adding steps

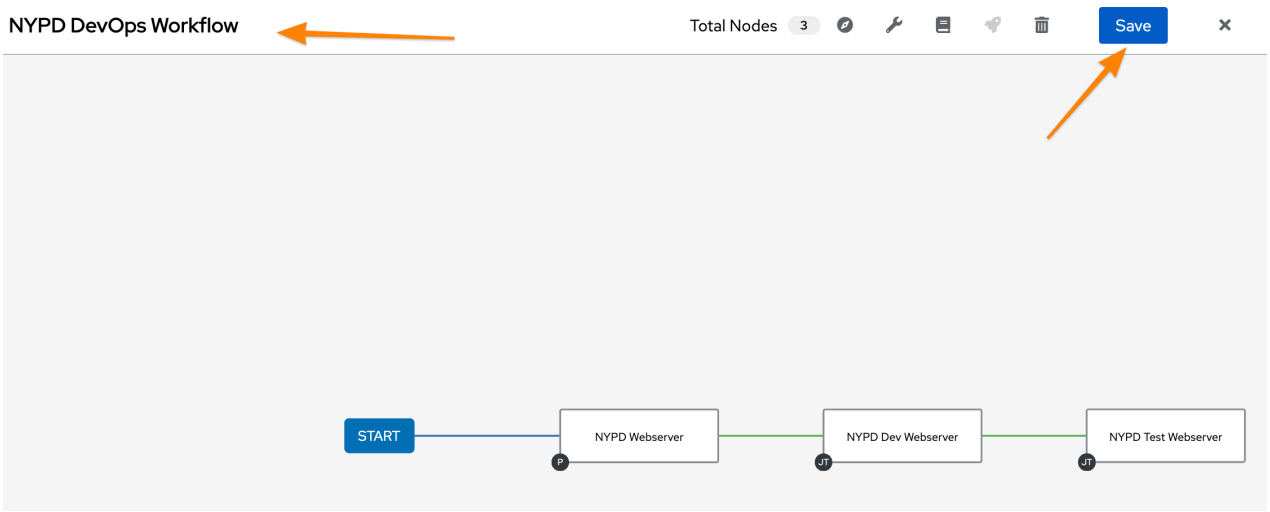


*Figure 59. Ansible Controller - Job Workflow Complete Overview*

*Job Workflows* - **Success**, **Failure**, *and* **Always**

It is important to architect workflows properly. In this example, we defined the **SUCCESS** path. In general, you will most likely have failure paths to cleanup environments from failed workflows. This has been left out based on time.

12. Launch job workflow to test by clicking **Launch**

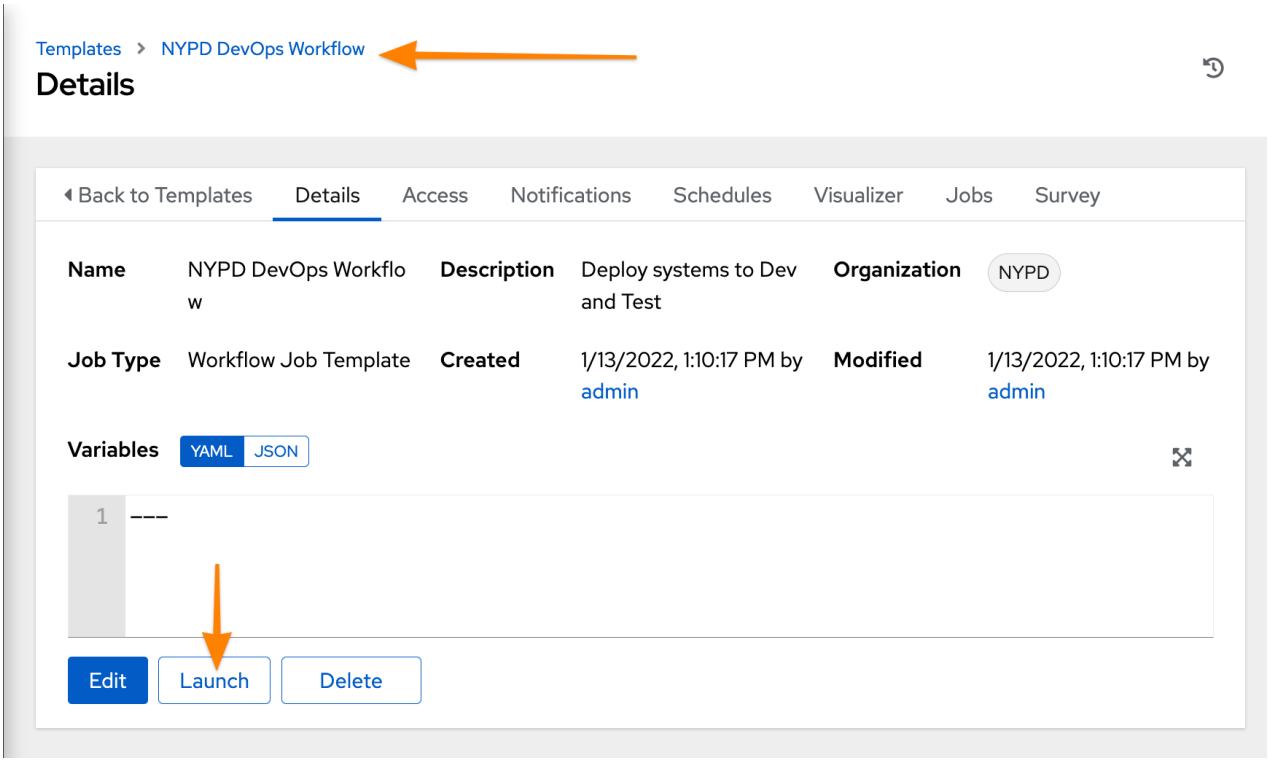


*Figure 60. Ansible Controller - Job Workflow Launching*

13. View workflow output

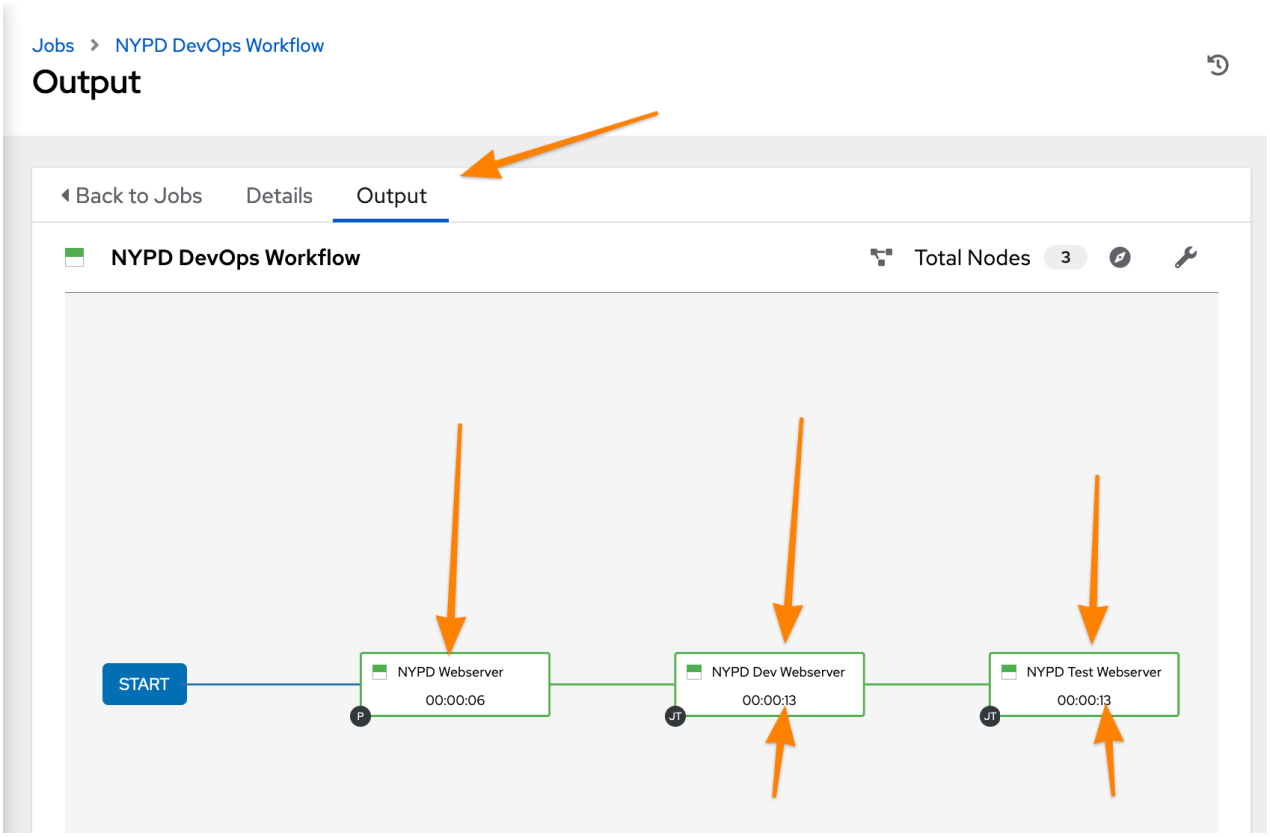


*Figure 61. Ansible Controller - Job Workflow Output*

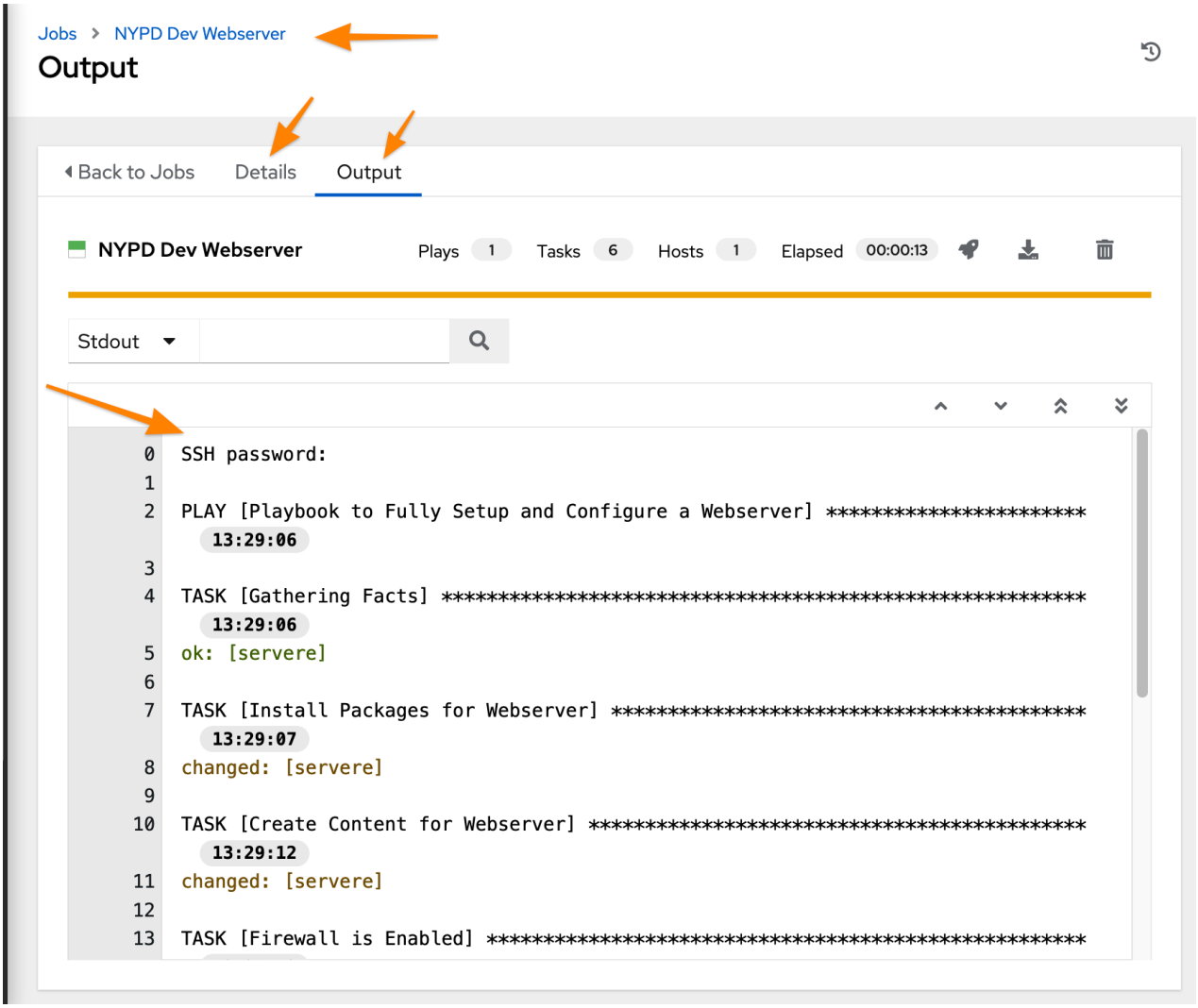


*Figure 62. Ansible Controller - Job Workflow Output - Details for Job Template*

14. Test from workstation

*Listing 11.* **Dev Server** *test*

```
[student@workstation ~]$ curl servere
I'm an awesome webserver for the NYPD and I know Castle!!
```

*Listing 12.* **Test Server** *test*

```
[student@workstation ~]$ curl serverf
I'm an awesome webserver for the NYPD and I know Castle!!
```