Automation Developer

TEKsystems Global Services

WORKFORCE DEVELOPMENT

Content Usage Parameters

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

1

Content is subject to copyright protection

2

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

3

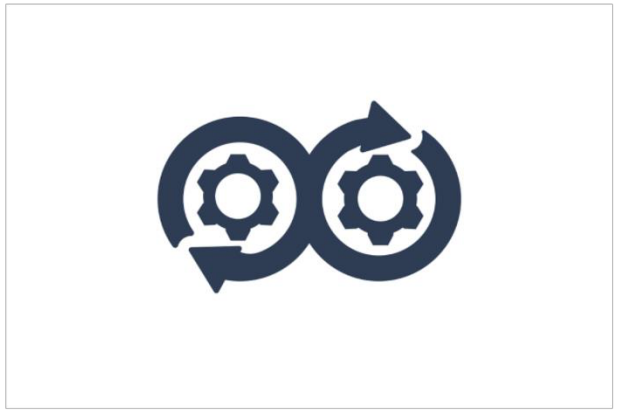
Students agree not to reproduce, make derivative works of, distribute, publicly perform and publicly display in any form or medium outside of the training program

4

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

Lab page

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



Ansible Vault

Ansible Vault

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

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

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

Ansible Vault

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

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

Ansible Vault

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

POP QUIZ: DISCUSSION

Things to consider:

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

Ansible Vault

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

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

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

Ansible Vault

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

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

Vault Id

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

Vault IDs provide labels to distinguish between individual vault passwords.

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

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

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

- ansible-vault
- ansible-playbook
- etc.

Ansible Vault

Create a new encrypted data file

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

Prompt for vault password

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

Use password file

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

Ansible Vault

Common vault commands

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

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

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

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

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

Lab: Ansible Vault, Async, Polling

Ansible Tags

Tags

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

Tags can be applied at multiple levels including:

- Tasks
- Roles
- Plays
- Blocks

Tags

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

Tags can be applied at multiple levels including:

- Tasks
- Roles
- Plays
- Blocks

Tags

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

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

Tags

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

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

Tags

This example shows the 'ntp' tag

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

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

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

Tags

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

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

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

Adding tags to roles allows you to run specific roles.

Tags

Define tags at the block level

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

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

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

Tags

Ansible reserves two tags:

- Always
  - Run the task/play unless specifically skipped
    - `--skip-tags always`
- Never
  - Skip unless specifically requested
    - `--tags never`

POP QUIZ: DISCUSSION

What are tags used for?

Ansible Galaxy

Ansible Collections

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

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

Ansible Collections

Modules:

Modules are reusable, standalone scripts that can be used by the Ansible API, the `ansible` command, or the `ansible-playbook` command. Modules provide a defined interface. Each module accepts arguments and returns information to Ansible by printing a JSON string to stdout before exiting. Modules execute on the target system (usually that means on a remote system) in separate processes.

Plugins:

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

Ansible Galaxy

Use ansible-galaxy to install collection

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

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

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

Use new collection (full namespace, collection, module)

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

Ansible Galaxy

Use ansible-galaxy to install collection or role

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

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

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

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

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

Ansible Galaxy

Use ansible-galaxy to install role

```
ansible-galaxy role install
```

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

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

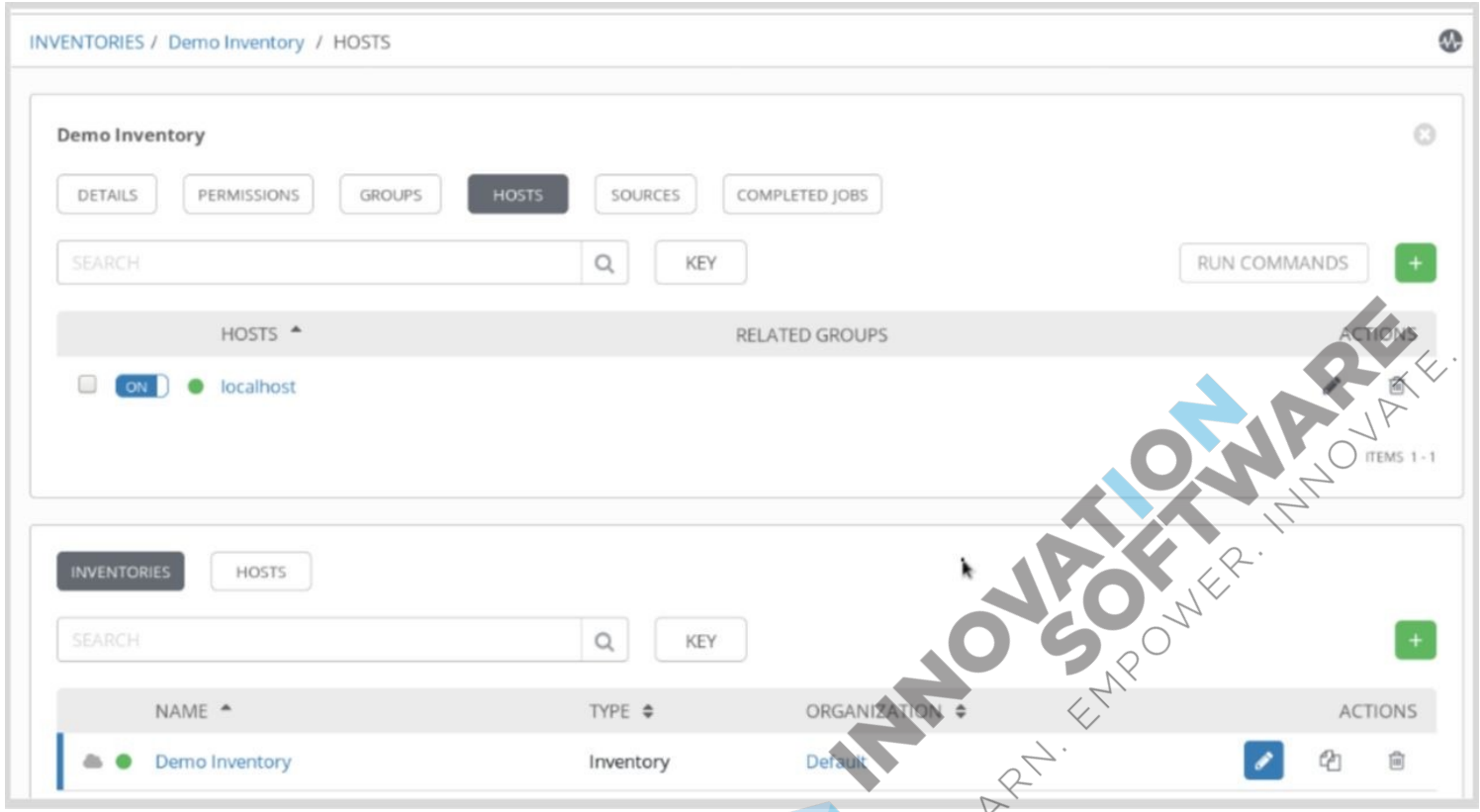
Use new role:

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

Lab: Ansible Roles

Ansible Inventory

- Manage inventory with Automation Platform
- This is a static inventory with no host groups and one host, called **`localhost`**.
- Clicking on that host in the inventory reveals that the inventory variable **`ansible_connection: local`** is set.



Accessing Ansible Docs

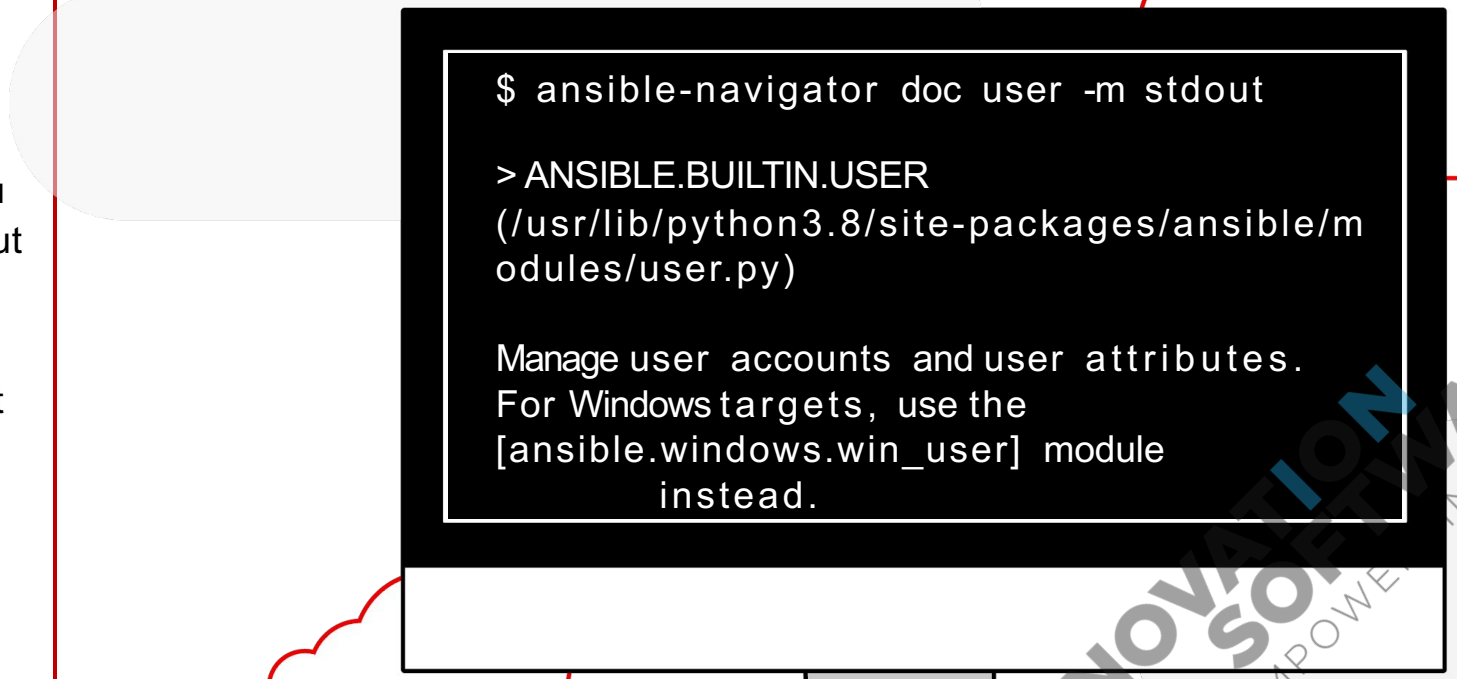
With the use of the latest command utility ansible-navigator, one can trigger access to all the modules available to them as well as details on specific modules.

A formal introduction to ansible-navigator and how it can be used to run playbooks in the following exercise.

Accessing Ansible Docs

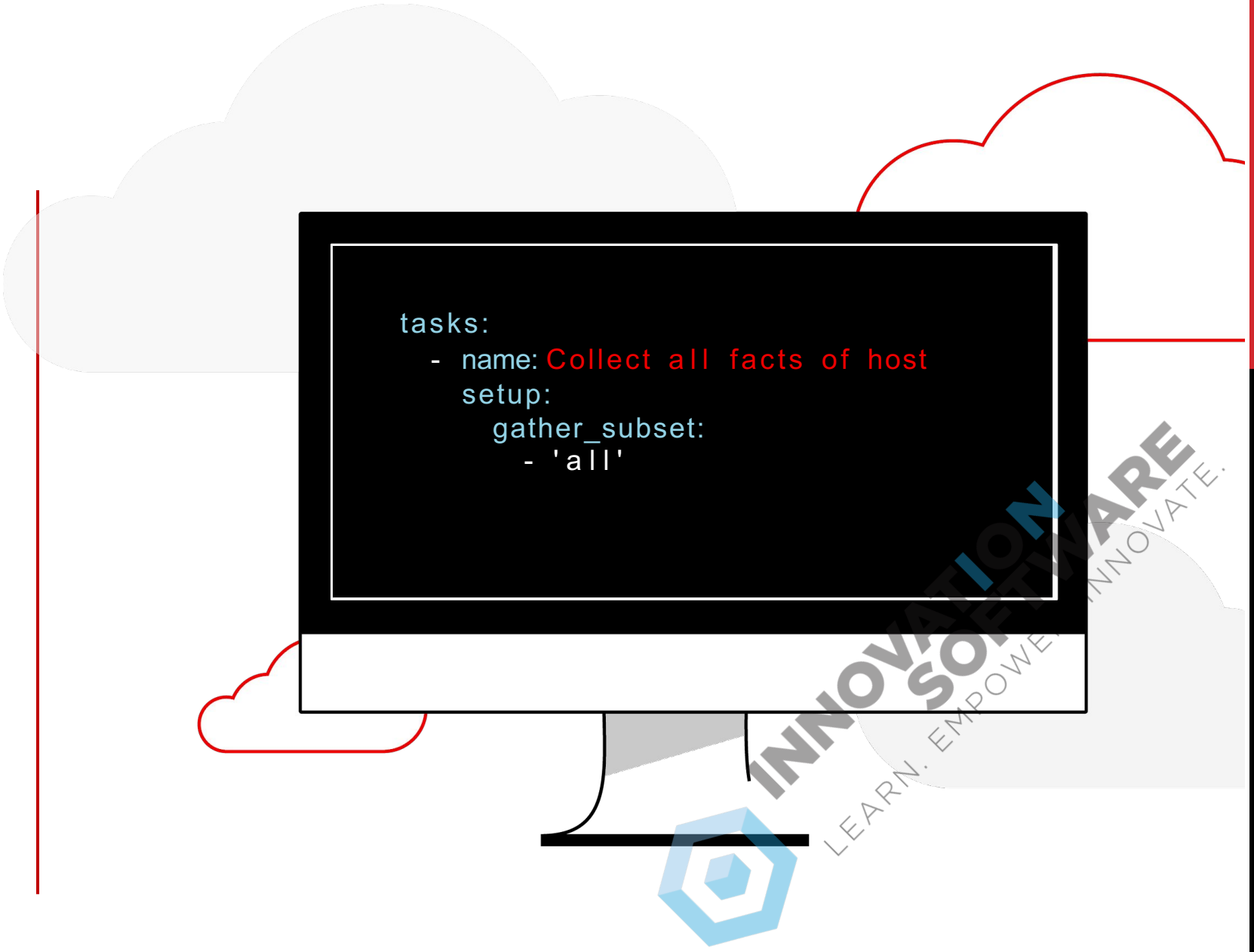
Aside from listing a full list of all the modules, you can use ansible-navigator to provide details about a specific module.

In this example, we are getting information about the user module.



Ansible Facts

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



Ansible Facts

Ansible playbooks

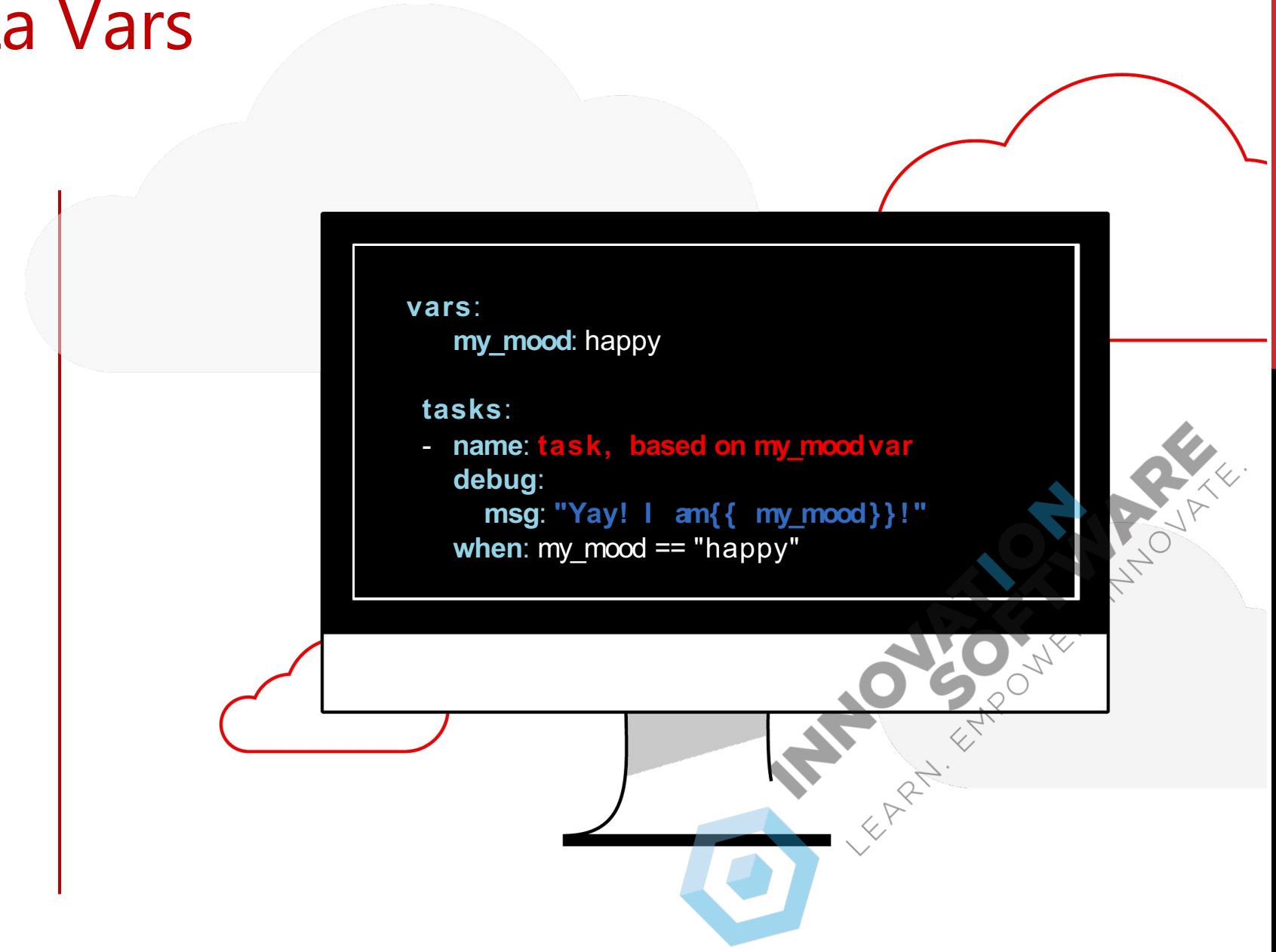
```
---
- name: facts playbook
  hosts: localhost

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

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

Conditionals Via Vars

Example of using a variable labeled *my_mood* and using it as a conditional on a particular task.



Conditionals Via Vars


```
---
- name: variable playbook test
  hosts: localhost

  vars:
    my_mood: happy

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

Alternatively

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

Conditionals With Facts

Ansible Conditionals w/ Facts

```
---
- name: variable playbook test
  hosts: localhost

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

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

Task State

Using Previous Task State

```
---
- name: variable playbook test
  hosts: localhost

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

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

Variables And Loops


```yaml
---
- name: Ensure users
  hosts: node1
  become: yes

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

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

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

Variables And Loops


```yaml
---
- name: Ensure users
  hosts: node1
  become: yes

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

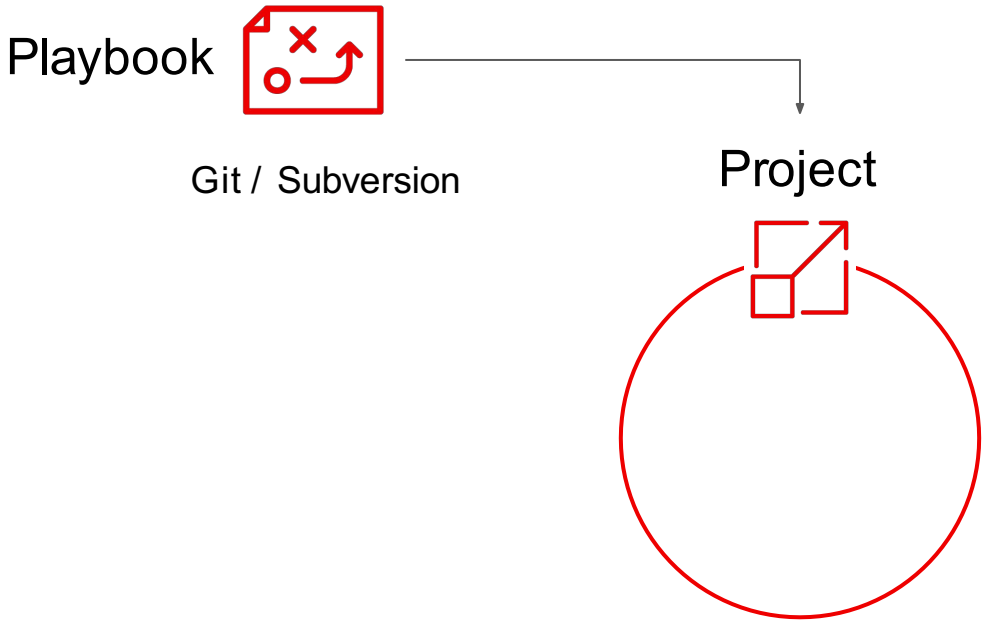
Automation Controller

What is Ansible Automation Controller?

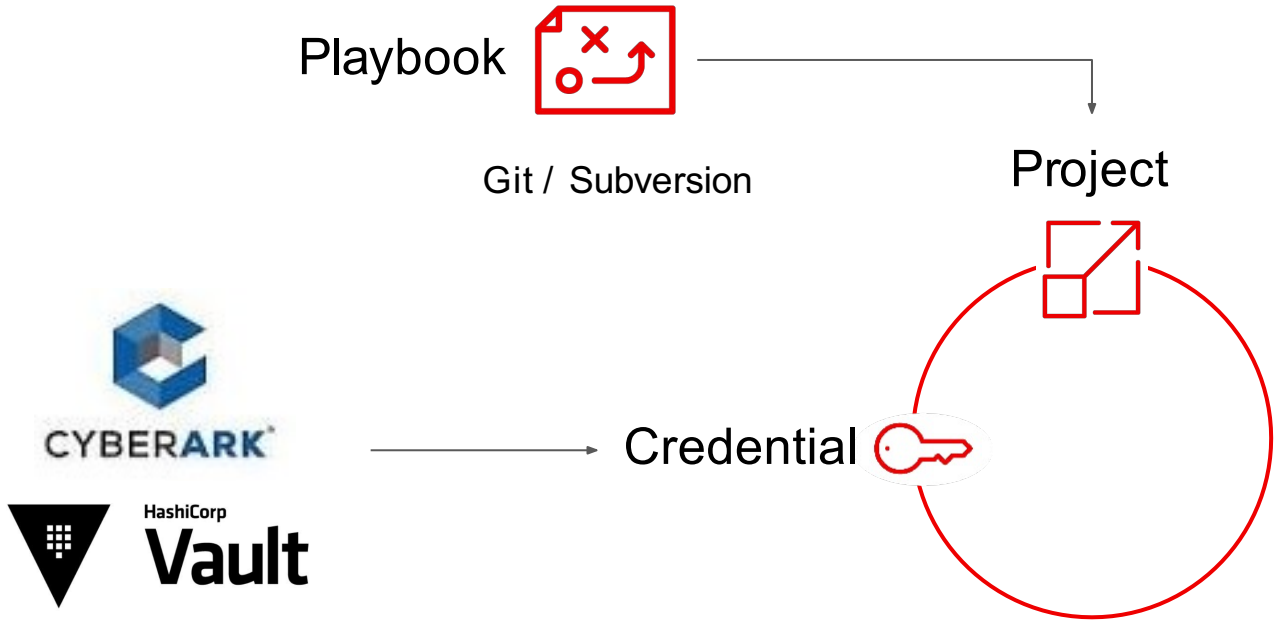
Ansible Automation Controller is a UI and RESTful API allowing you to scale IT automation, manage complex deployments and speed productivity.

- Role-based access control
- Deploy entire applications with push-button deployment access
- All automations are centrally logged
- Powerful workflows match your IT processes

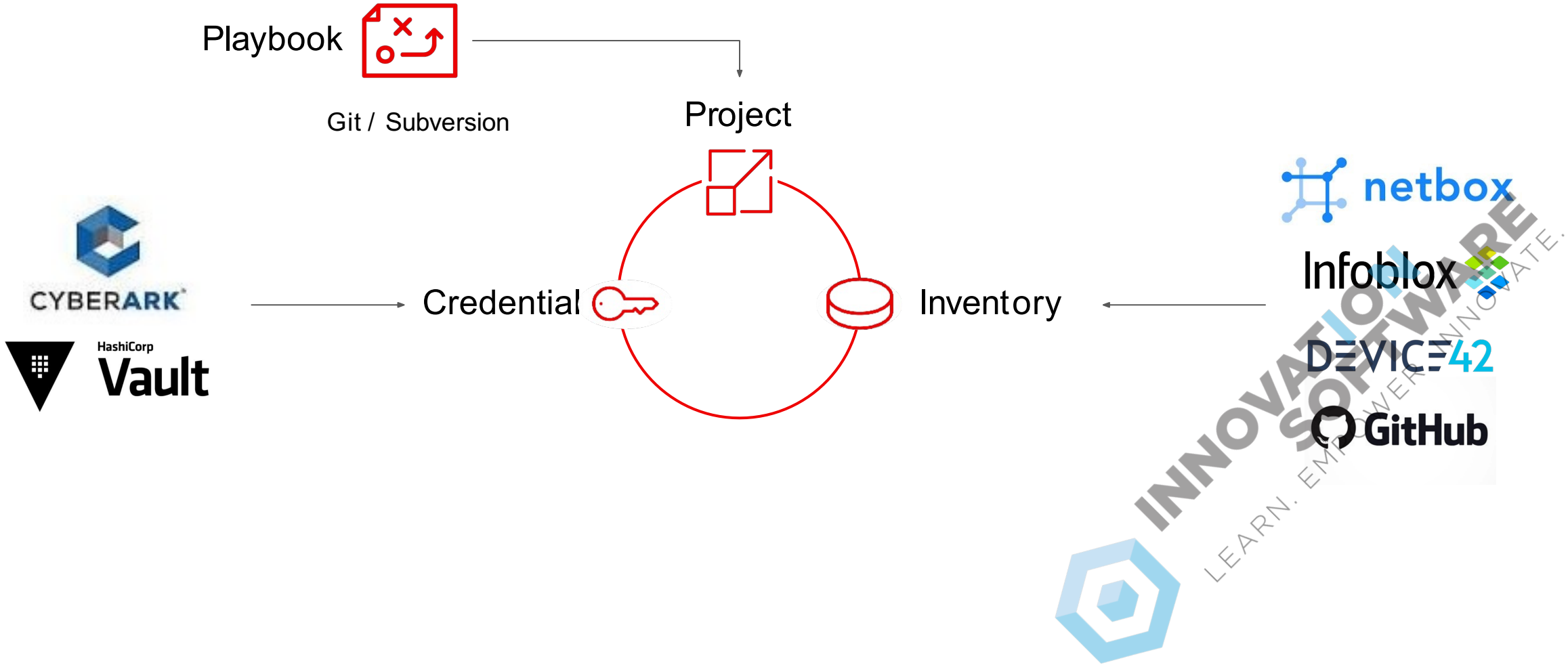
Anatomy of An Automation Job



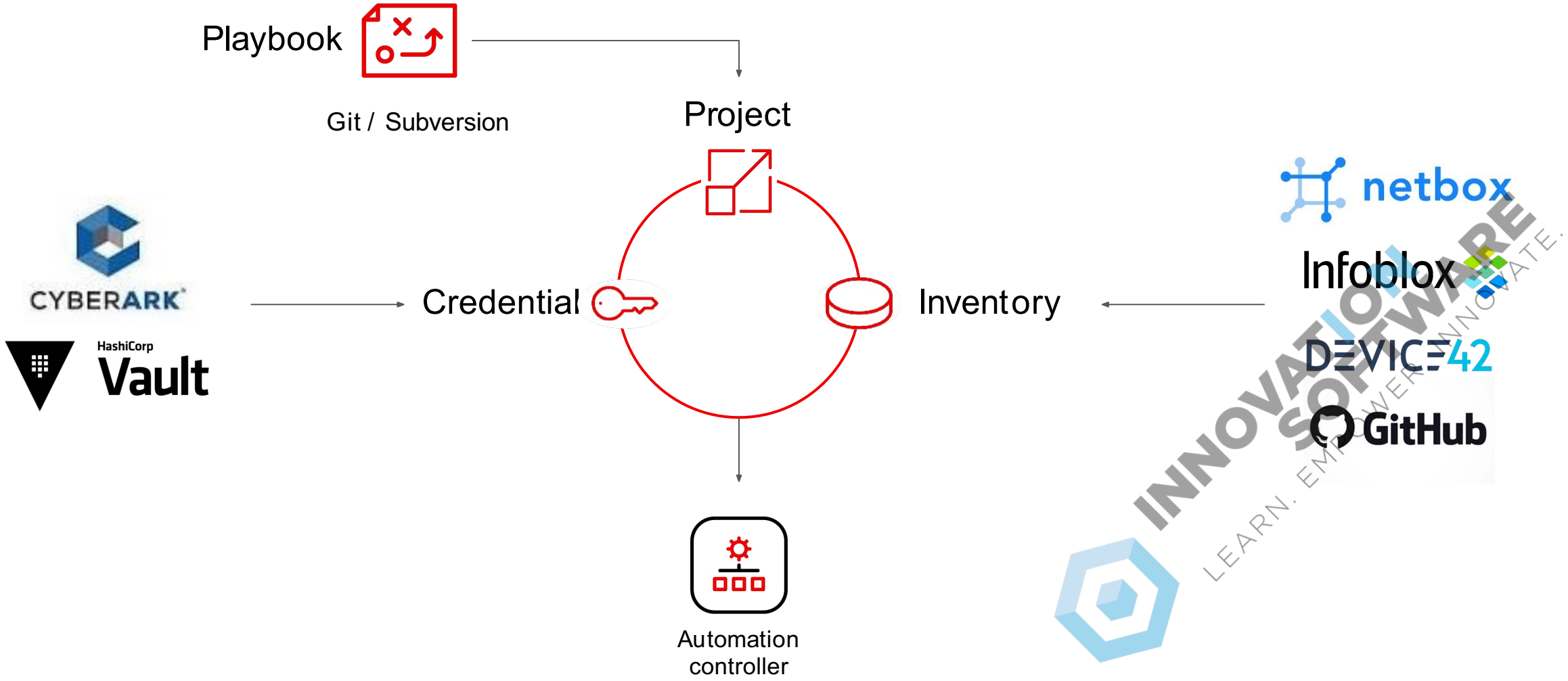
Anatomy of An Automation Job



Anatomy of An Automation Job

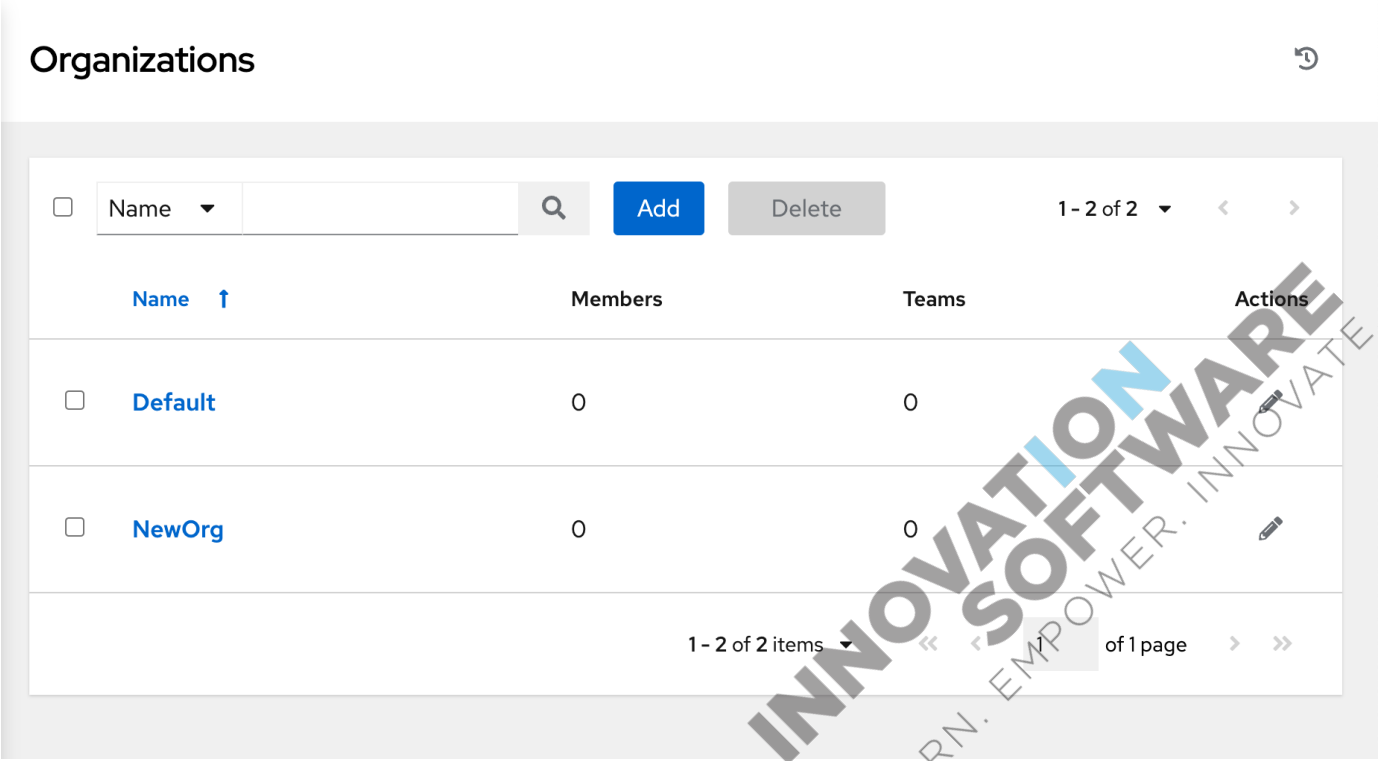


Anatomy of An Automation Job



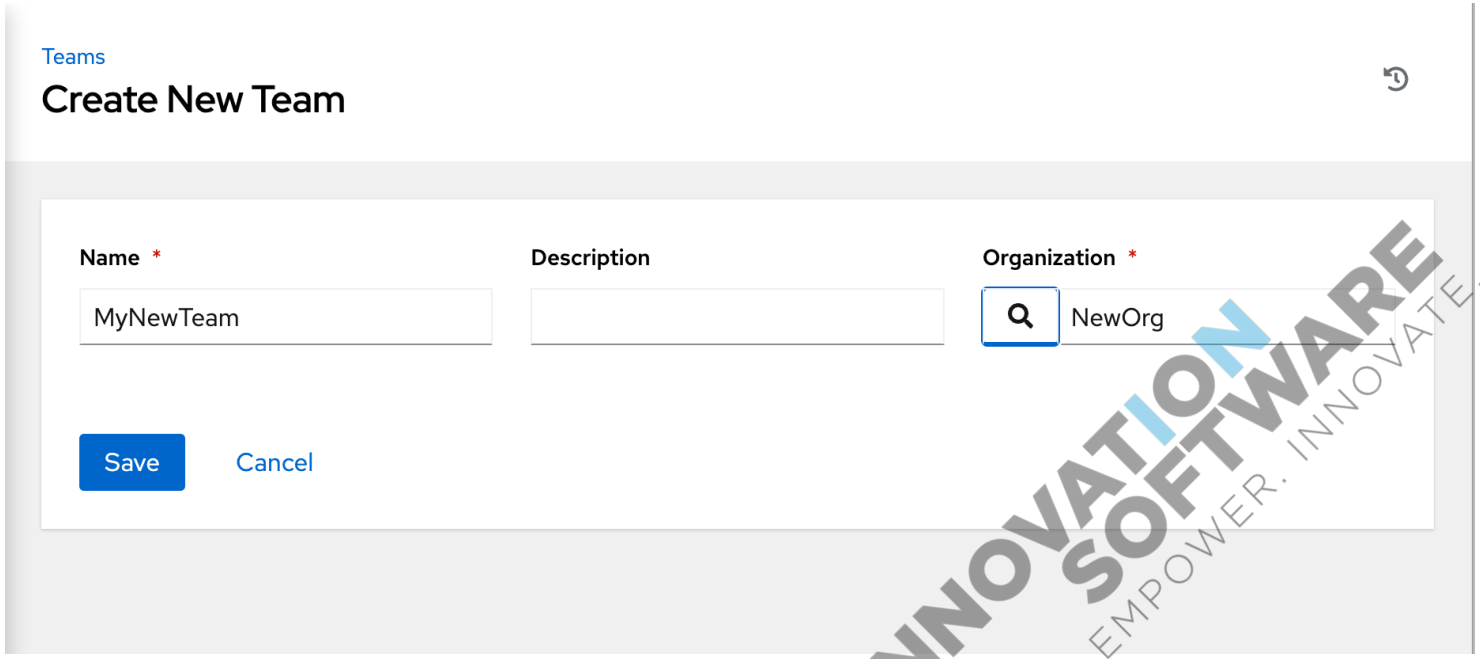
Organization

- 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.



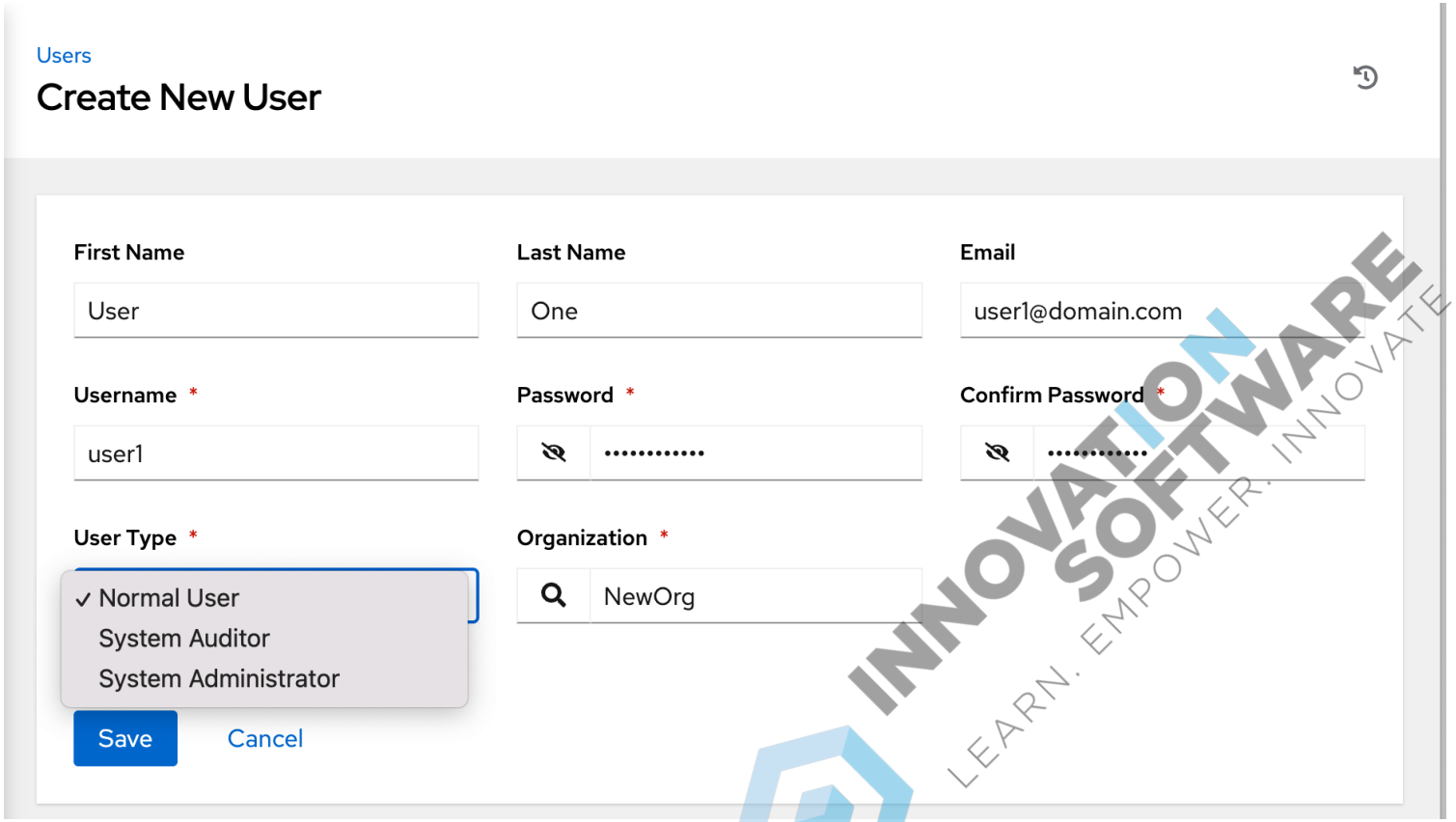
Team

- You can apply permissions at the team level.



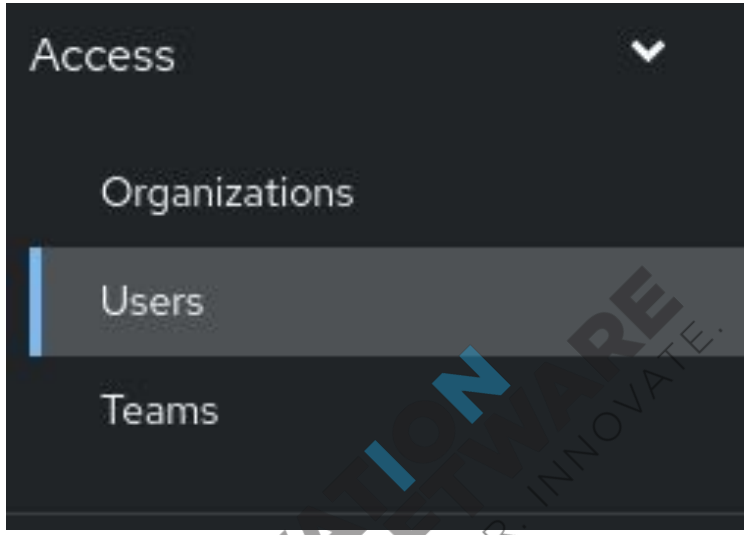
User Roles

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



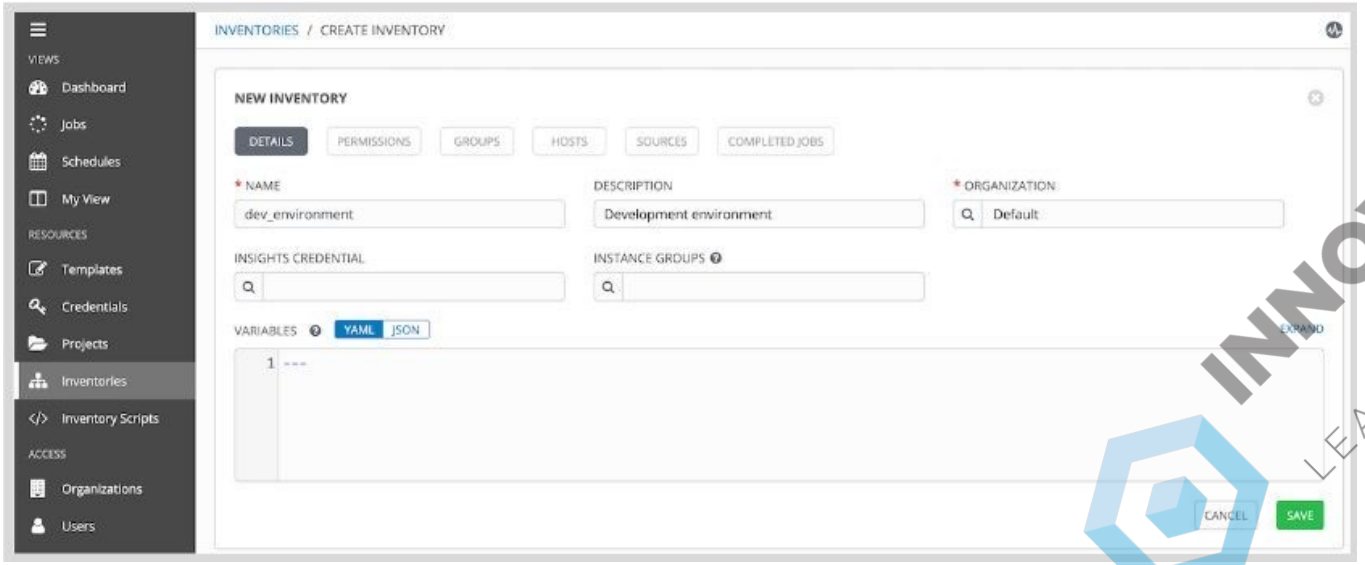
User Management

- 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.



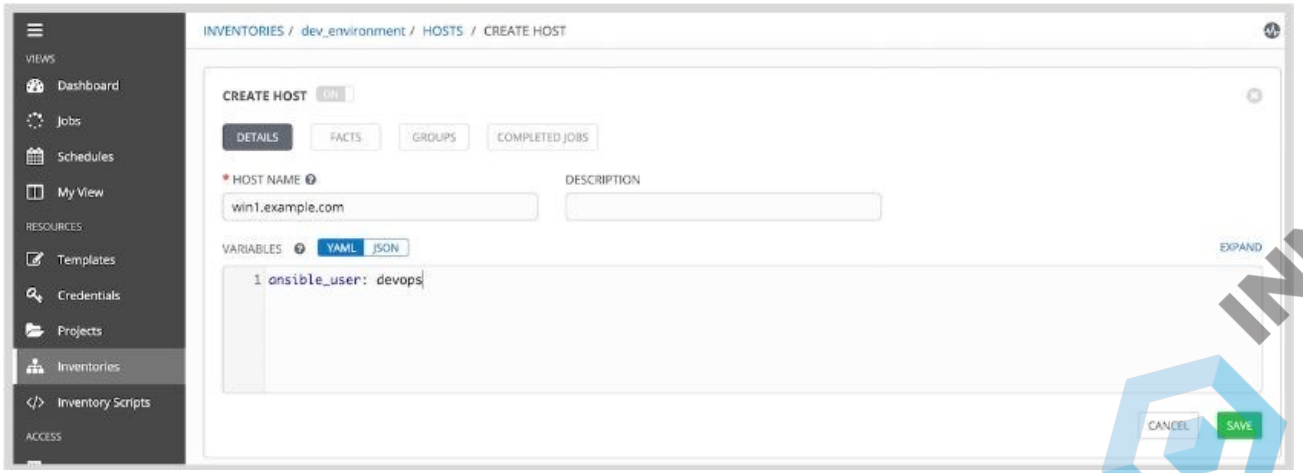
Inventory

- 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**.

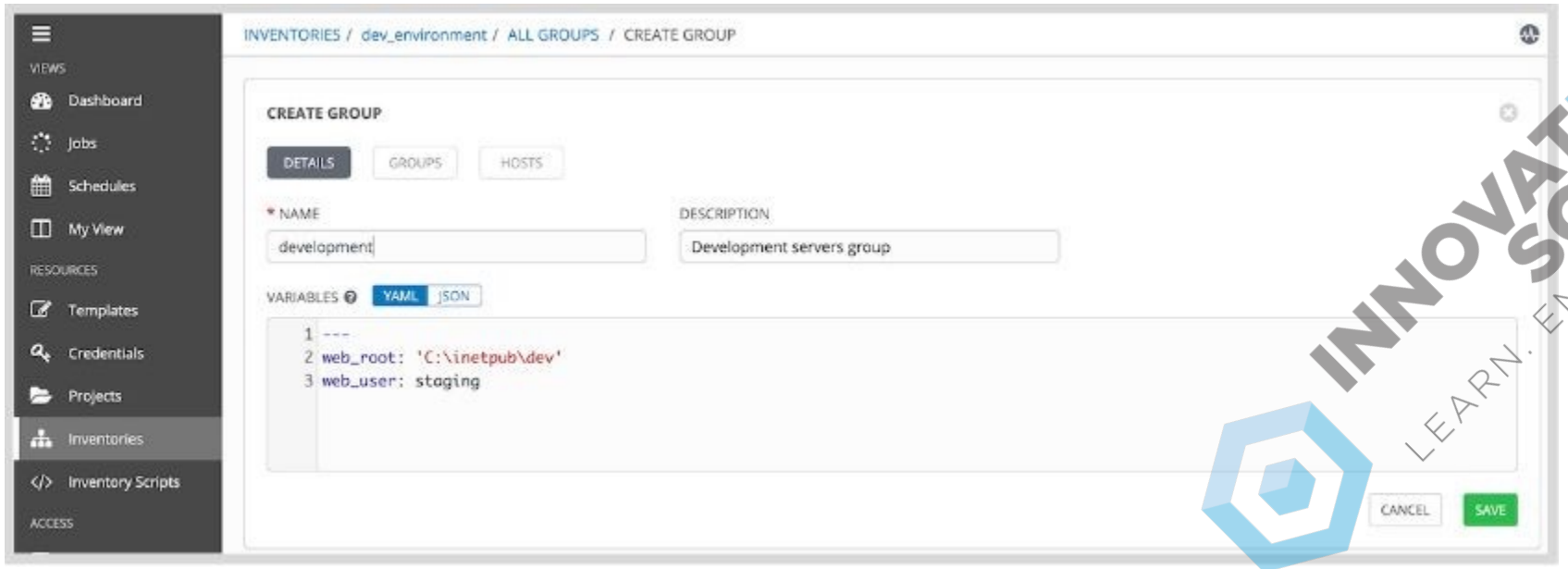


Inventory Groups

- 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**.

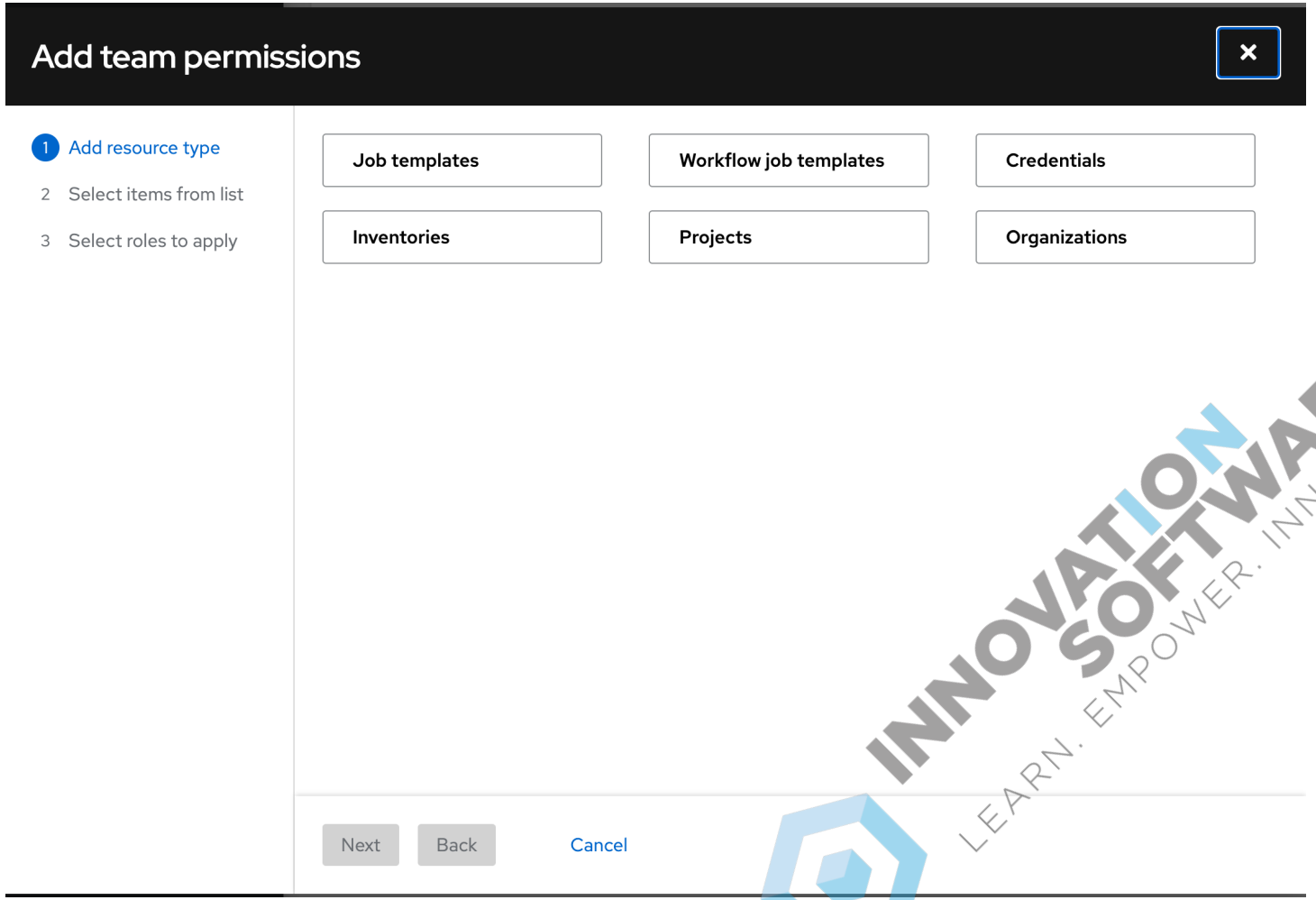


Inventory Groups

- 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

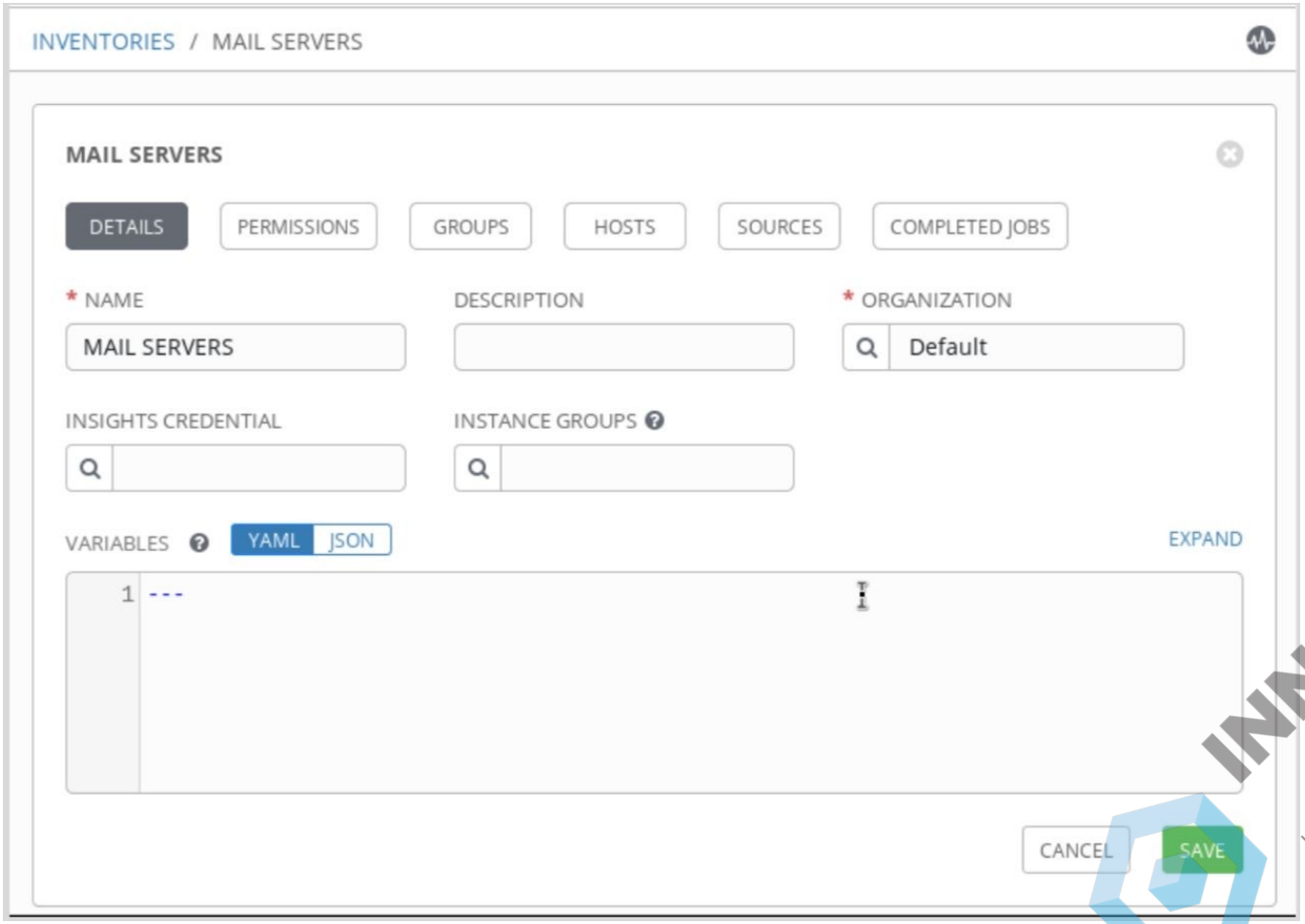
Role	Description
Admin	The inventory **Admin** role grants users full permissions over an inventory. These permissions include deletion and modification of the inventory. In addition, this role also grants permissions associated with the inventory roles **Use**, **Ad Hoc**, and **Update**.
Use	The inventory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

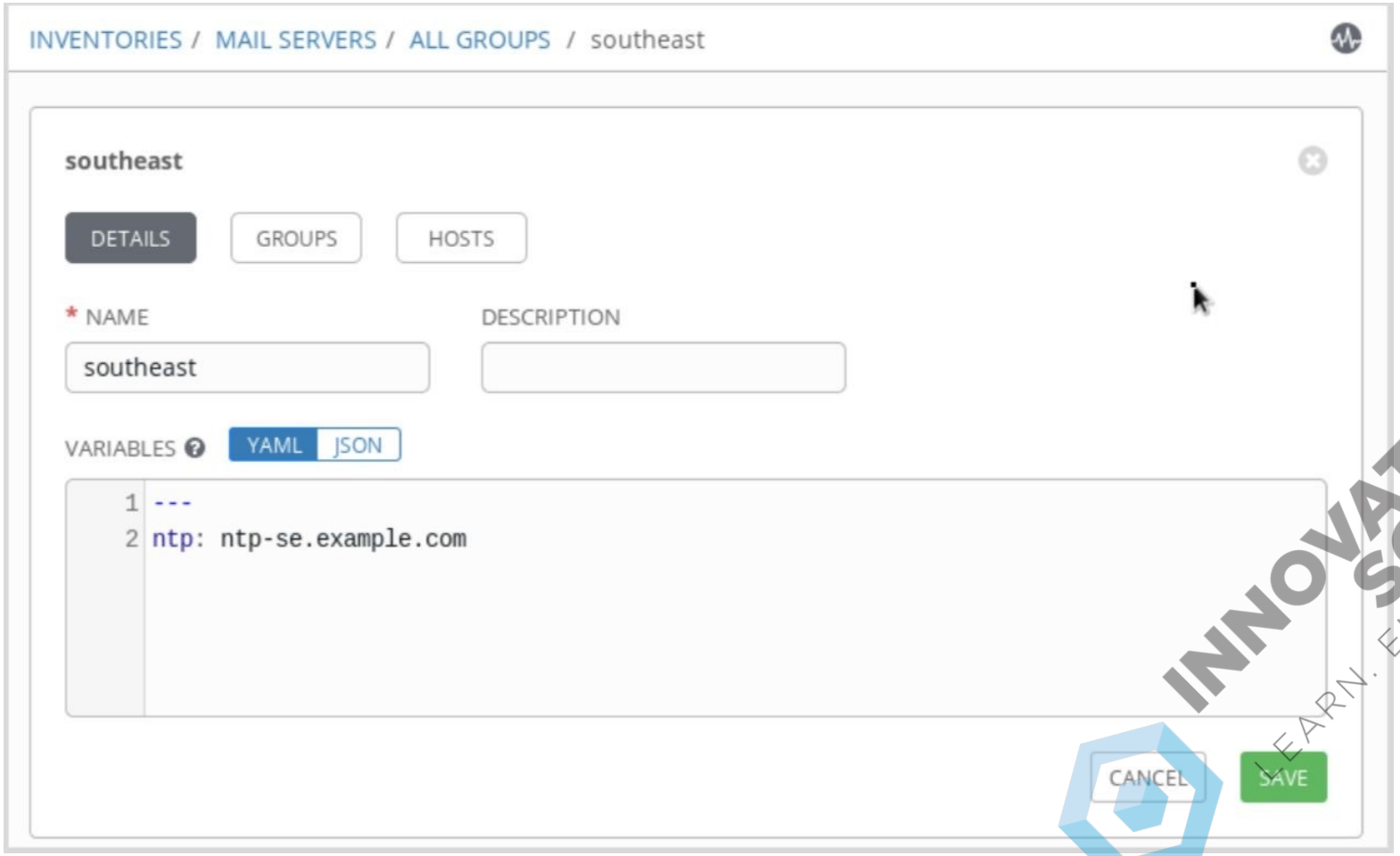
When you manage a static inventory in the Ansible Platform web UI, you may define inventory variables directly in the inventory objects.

- 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`**.

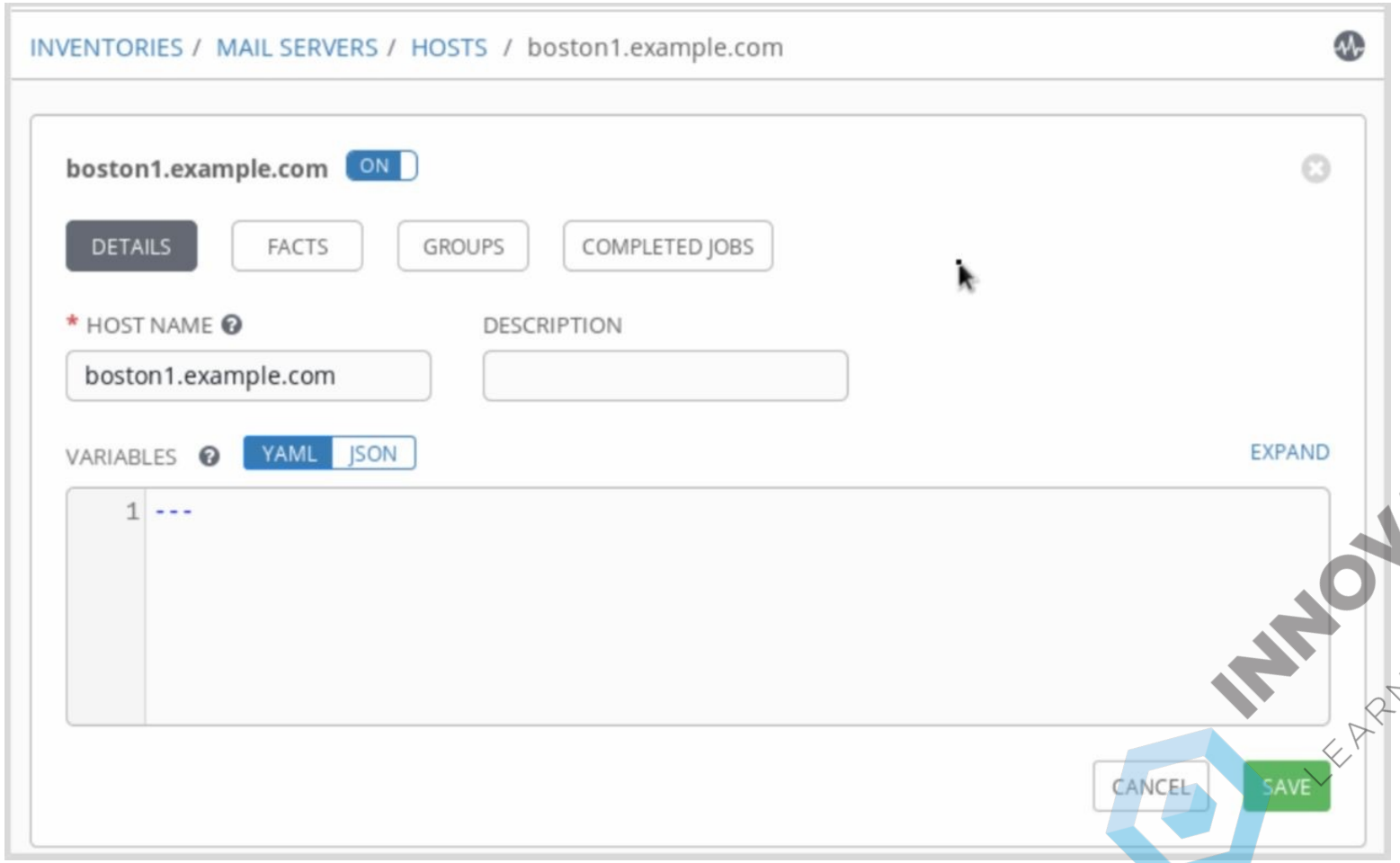
Inventory Variables



Inventory Group Variables



Inventory Host Variables

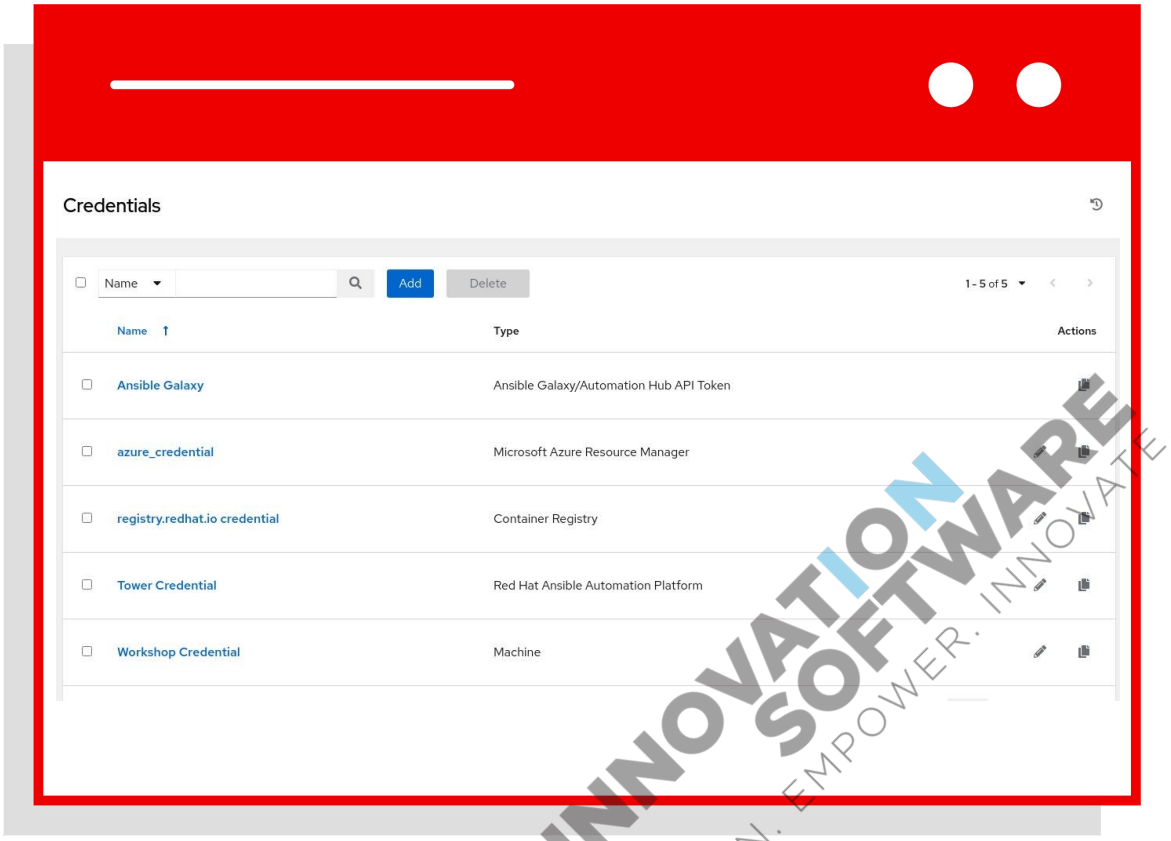


Credentials

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

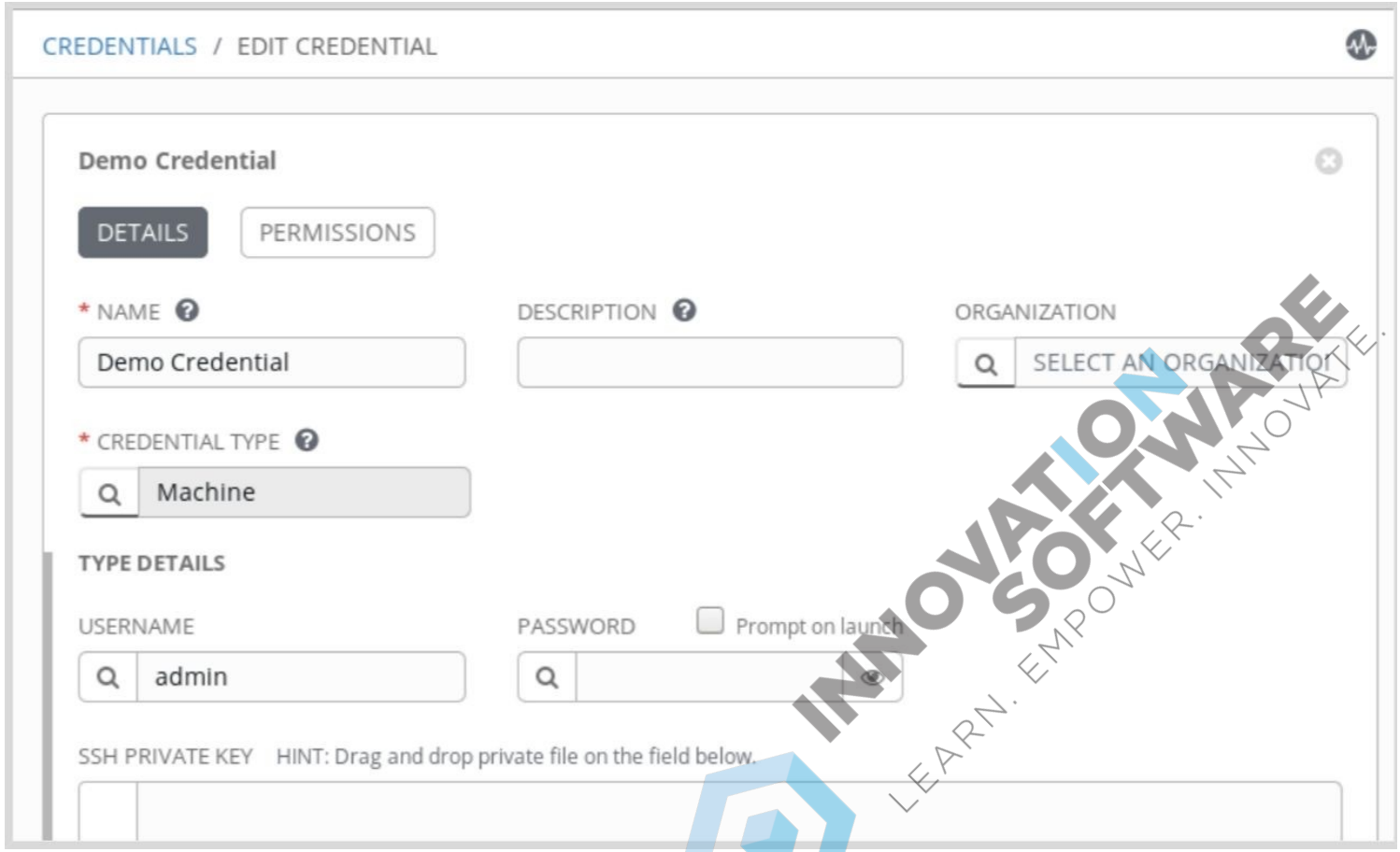
- Connecting to remote machines to run jobs
- Syncing with inventory sources
- Importing project content from version control systems
- Connecting to and managing network devices

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

Rolling Updates

Batch size:

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

Rolling Updates

Example playbook utilizing `serial` keyword.

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

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

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

Rolling Updates

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

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

TASK [task one] ******************************************
changed: [web2]
changed: [web1]

TASK [task two] ******************************************
changed: [web1]
changed: [web2]


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

Rolling Updates

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

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

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

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

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

Rolling Updates

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

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

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

Rolling Updates

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

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

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

Rolling Updates

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

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

Rolling Updates

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

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

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