IaC – Pipelines to provision OpenIDL node using Jenkins, AWX and Terraform Cloud/Enterprise

**Author**: Rajesh Sanjeevi

**Kind**: Draft

**Version**: 1.2

**Contents**

[1. Environment Required 4](#_Toc93340611)

[2. Source code repository (GitHub) 4](#_Toc93340612)

[3. Terraform Cloud/Enterprise 6](#_Toc93340613)

[3.1 Terraform User/Team Token 6](#_Toc93340614)

[3.1.1 TFC/TFE User Token 6](#_Toc93340615)

[3.1.2 TFC/TFE Team Token 6](#_Toc93340616)

[3.2 Workspaces 7](#_Toc93340617)

[3.2.1 Workspace to manage K8S Resources 7](#_Toc93340618)

[3.2.2 Workspace to manage AWS Resources 8](#_Toc93340619)

[3.3 Variable Set 10](#_Toc93340620)

[3.4 Team Access 12](#_Toc93340621)

[4. Jenkin’s Environment 13](#_Toc93340622)

[5.1 Plugins required 13](#_Toc93340623)

[5.2 Node labels 14](#_Toc93340624)

[5.3 Global tools configuration 14](#_Toc93340625)

[5.4 Configure System (AWX/Ansible Tower) 15](#_Toc93340626)

[5.5 Credentials 16](#_Toc93340627)

[5.5.1 Username and Password Type 17](#_Toc93340628)

[5.5.2 Secret Text Type 17](#_Toc93340629)

[5. Terraform code changes to adapt to Terraform Cloud/Enterprise 18](#_Toc93340630)

[**6.** Ansible Tower/AWX Environment 20](#_Toc93340631)

[6.1 User Account 20](#_Toc93340632)

[6.2 Credential Types 20](#_Toc93340633)

[6.2.1 OpenIDL-IAC 21](#_Toc93340634)

[6.2.2 OpenIDL-IAC-AWSUser-BAF 25](#_Toc93340635)

[6.2.3 OpenIDL-APP 26](#_Toc93340636)

[6.3 Inventory, Group and Host 30](#_Toc93340637)

[6.4 Credentials 32](#_Toc93340638)

[6.4.3 Machine Credential 32](#_Toc93340639)

[6.4.4 Source Control Credential 33](#_Toc93340640)

[6.4.5 OpenIDL-IAC 34](#_Toc93340641)

[6.4.6 OpenIDL-APP 36](#_Toc93340642)

[6.4.7 OpenIDL-IAC-AWSUser-BAF 37](#_Toc93340643)

[6.5 Projects 38](#_Toc93340644)

[6.4.8 openidl-main 38](#_Toc93340645)

[6.4.9 openidl-aais-gitops 39](#_Toc93340646)

[6.6 Templates 39](#_Toc93340647)

[6.6.1 Vault Install 40](#_Toc93340648)

[6.6.2 MongoDB Install 42](#_Toc93340649)

[6.6.3 BlockChain 44](#_Toc93340650)

[6.6.4 Register Users 46](#_Toc93340651)

[6.6.5 OpenIDL Application Install 48](#_Toc93340652)

[6.6.6 OpenIDL Application Secrets Install 50](#_Toc93340653)

[7. Jenkins Job Configuration 52](#_Toc93340654)

[7.1 Credentials 52](#_Toc93340655)

[7.2 Job Configurations 54](#_Toc93340656)

[7.2.1 Terraform Job 54](#_Toc93340657)

[7.2.2 Vault Job 55](#_Toc93340658)

[7.2.3 Blockchain Network Job 57](#_Toc93340659)

[7.2.4 MongoDB Job 58](#_Toc93340660)

[7.2.5 OpenIDL Application Job 59](#_Toc93340661)

[8. Executing Jenkins Pipeline 61](#_Toc93340662)

[8.1 Terraform Job 61](#_Toc93340663)

[8.2 Preparing Config file for Infra Job 63](#_Toc93340664)

[8.3 Vault Job 64](#_Toc93340665)

[8.4 Blockchain Network Job 64](#_Toc93340666)

[8.5 MongoDB Job 66](#_Toc93340667)

[8.6 Preparing Config files for OpenIDL application job 66](#_Toc93340668)

[8.7 OpenIDL Application Job 67](#_Toc93340669)

# Environment Required

1. Source code repository (GitHub)
2. Jenkins
3. Terraform Cloud/Enterprise
4. Ansible Tower /AWX (open source)

This document is an extension which details about Jenkins pipelines which works with Terraform Cloud/Enterprise and AWX to provision OpenIDL node. Hence follow the base document and as an extension work along these steps to setup Jenkins, AWX, Terraform Cloud, Git repository to use these pipelines.

# Source code repository (GitHub)

The following are the two repositories involved.

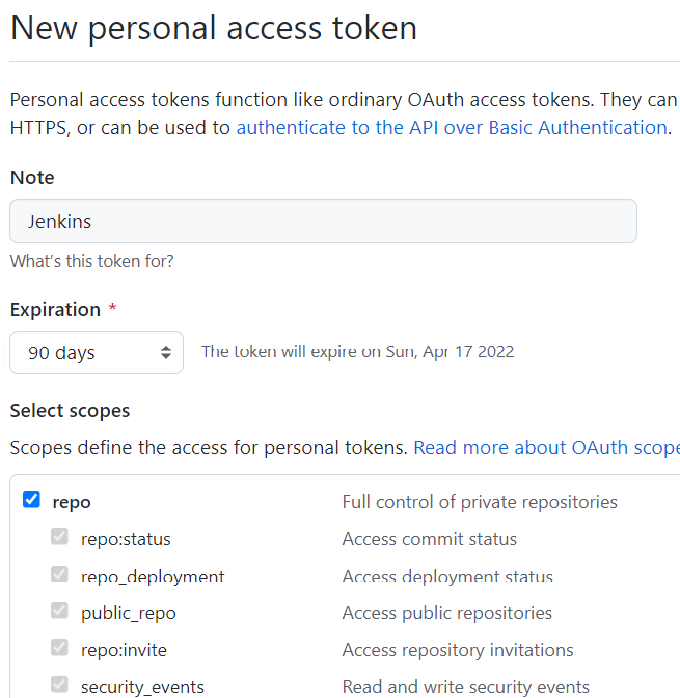
|  |  |  |
| --- | --- | --- |
| **S No** | **Repository** | **Description** |
| 1 | openidl-aais-gitops | Infrastructure as a code repository. This repository is used to provision Infrastructure using related pipelines. |
| 2 | openidl-main | Application specific code repository. This repository is used to provision application using its specific pipelines. |

A user account with necessary permissions to manage these repositories is required. Further provision a Personal Access Token with Selected scopes as **“repo”**.

The following are the areas the token is used. A single PAT or multiple PAT can be provisioned and used according to each organization decisions. Either provision one or below listed number of tokens and use accordingly.

|  |  |  |
| --- | --- | --- |
| **S No** | **Personal Access Token** | **Description** |
| 1 | PAT 1 | A personal access token which will be used by Jenkins to connect to GitHub. This token will be added as a username/password secret in Jenkins to allow it to connect to repositories successfully. |
| 2 | PAT 2 | A personal access token which will be used by AWX/Tower to connect to source control to sync project (playbooks). This will be added as a source control credential in AWX/Tower and further used to sync playbooks. |
| 3 | PAT 3 | A personal access token used by ansible playbooks to download content from the repository during playbook run on remote host. |

1. To provision PAT in GitHub (Source control) login to GitHub, go to settings => Developer settings => Personal access tokens => Generate new token.
2. Name the token, set expiration as either no expiration or required number of days if decided to refresh on a specific interval.
3. Set the selected scopes as “repo”



**NOTE:** Once the necessary tokens are provisioned, please get them recorded to enable them as secrets/credentials in Jenkins/AWX in next steps.

# Terraform Cloud/Enterprise

The following are items are required to setup in Terraform.

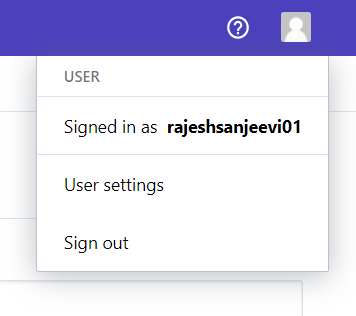
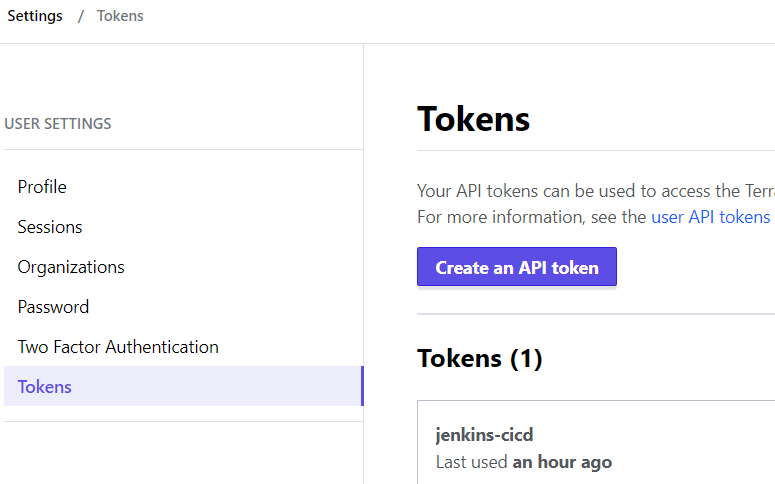
1. User Token/Team Token
2. Workspaces
3. Variable Set

## Terraform User/Team Token

A user token/team token is required to allow Jenkins to authenticate and successfully communicate with Terraform. It depends on an organization to choose between the type of token used according to their need.

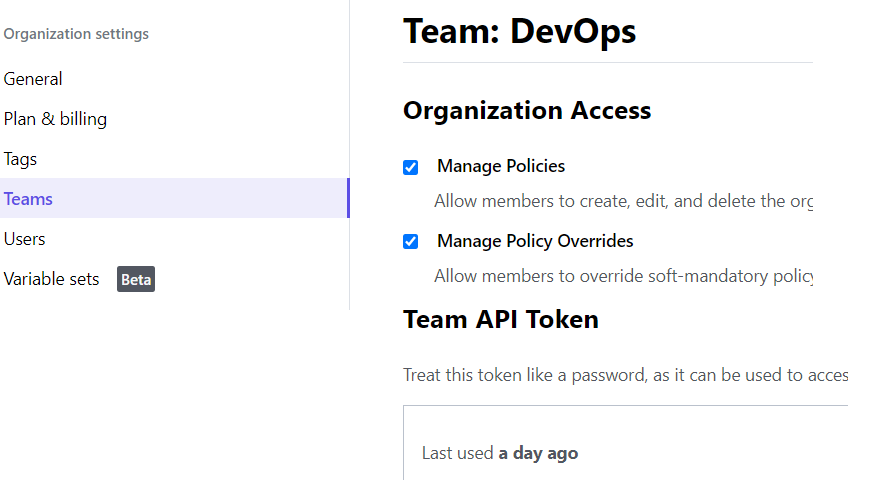
### 3.1.1 TFC/TFE User Token

1. Login to Terraform Cloud/Enterprise go to User settings
2. Create an API token

 ****

### 3.1.2 TFC/TFE Team Token

1. Login to Terraform Cloud/Enterprise, go to Organization settings
2. Go to Teams to setup up a new team and provision a team token or go to existing team and provision a team API token.

****

## Workspaces

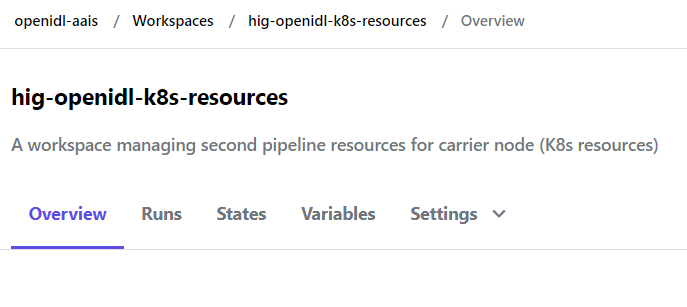
The terraform code to provision necessary Infrastructure resources for OpenIDL node is provisioned into two independent set. The first set is used to provision AWS resources and the other one to provision K8s resources. There is dependency in provisioning K8s which are addressed in first set of code and before provisioning K8s.

For example, K8s resources like config-map, storage class and ha proxy has dependency with EKS cluster which gets provisioned before these resources. Hence two set of codes are managed which requires two different terraform workspaces in the environment to manage and configure. The details are below.

### Workspace to manage K8S Resources

1. A workspace to manage K8s resources is required. Create a new workspace and choose workflow as “API-Drive workflow” and give a meaningful name.
2. Open the workspace go to settings => General and set the execution mode to Remote, Apply method as Manual and Terraform version above 1.0.5

***Note****: This workspace refers to the state file of AWS resources workspace.*



**A screenshot of a computer

Description automatically generated**

**Graphical user interface, text, application

Description automatically generated**

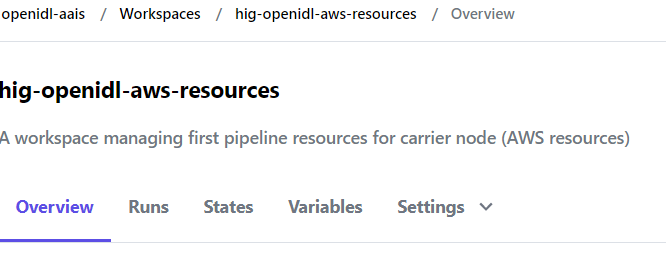
**Graphical user interface, text, application

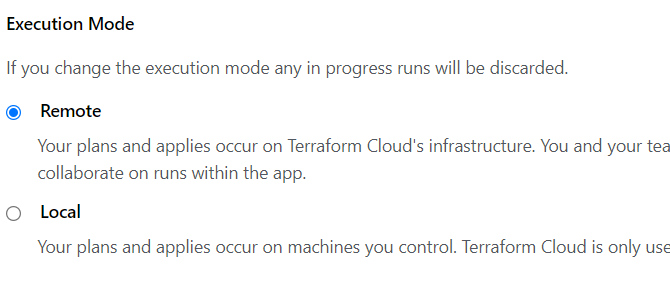
Description automatically generated**

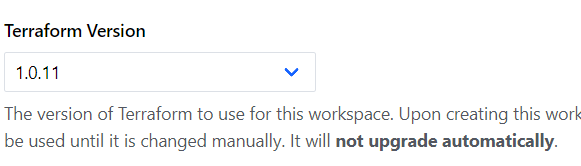
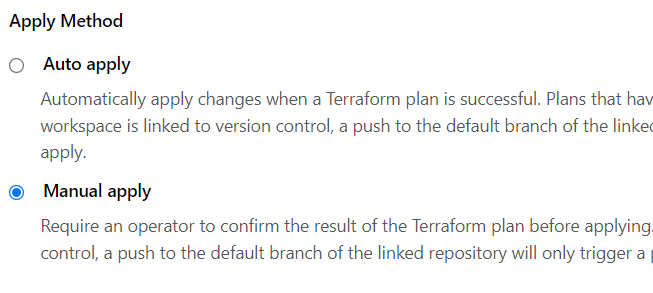
### Workspace to manage AWS Resources

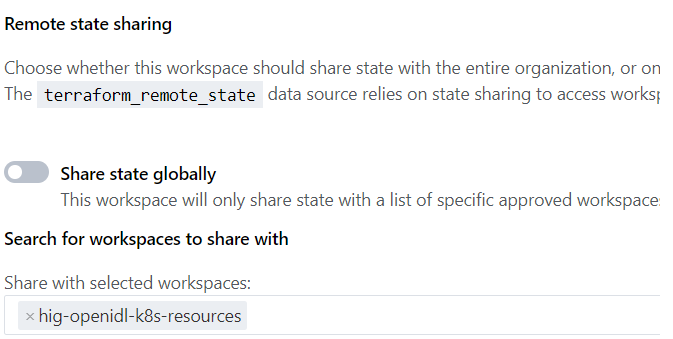
1. A workspace to manage AWS resources is required. Create a new workspace and choose workflow as “API-Drive workflow” and give a meaningful name.
2. Open the workspace go to settings => General and set the execution mode to Remote, Apply method as Manual and Terraform version above 1.0.5
3. And finally, *allow the state file of this workspace is accessible to the workspace used to manage K8s resources.*

***Note:*** *This workspace shares its state file with K8s resources workspace*

****

****

****

****

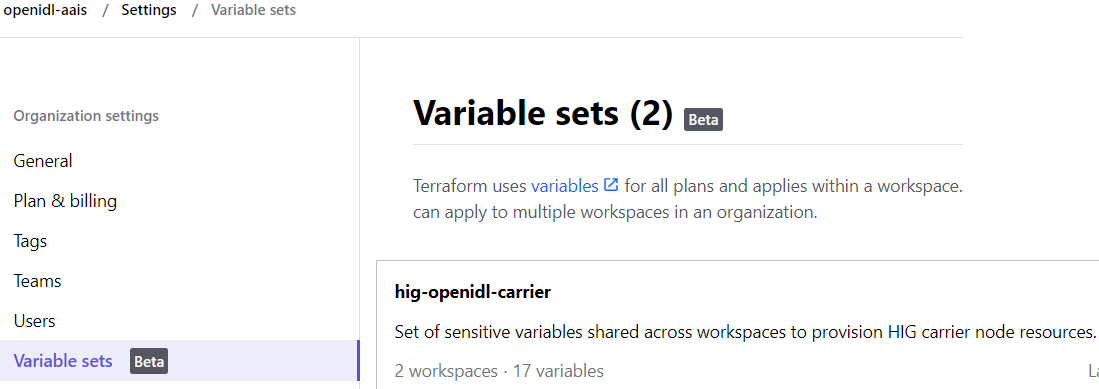
## Variable Set

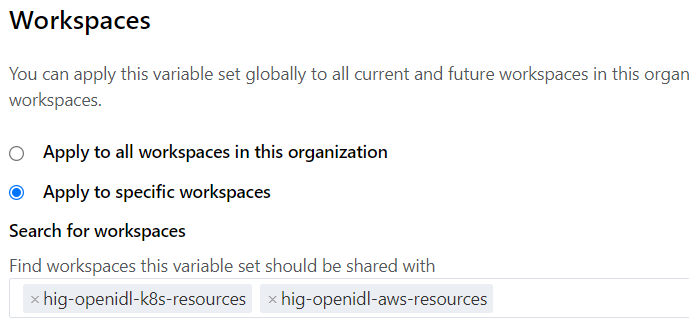
All the terraform variables and their values including non-sensitive and sensitive are added inside variable set. The details of terraform inputs can be referred in the repository under directory “aws/templates”.

The variable set is preferred as it can be shared across workspaces which is the typical use case in our solution. Configure variable set and share them across the workspace’s setup in previous section.

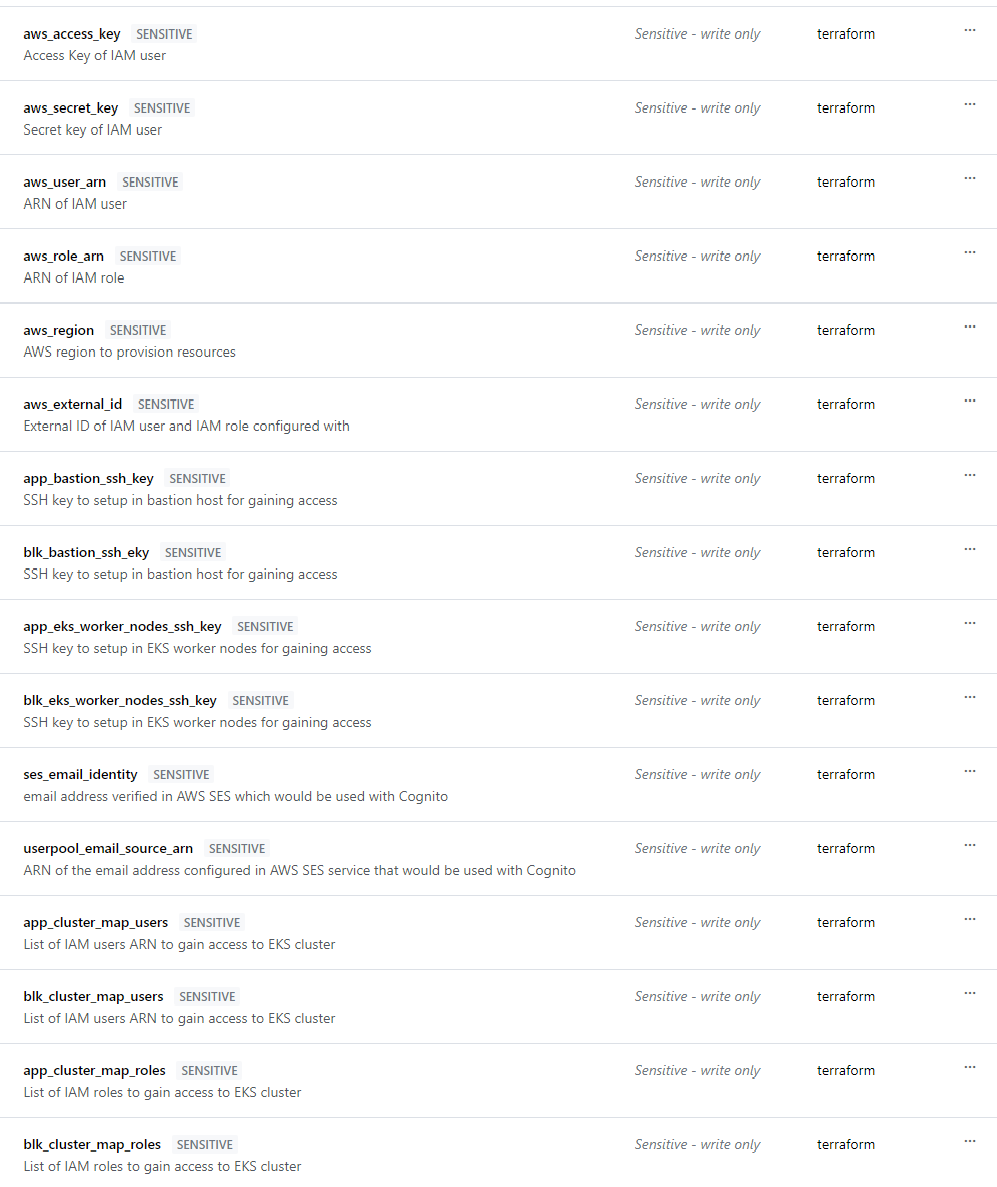
**NOTE:** When you are entering variables, in case of complex data types like maps, lists etc, follow HCL format and ensure the checkbox HCL is checked. Please refer to the below link and section “variable values and format”

<https://www.terraform.io/cloud-docs/workspaces/variables/managing-variables>

****

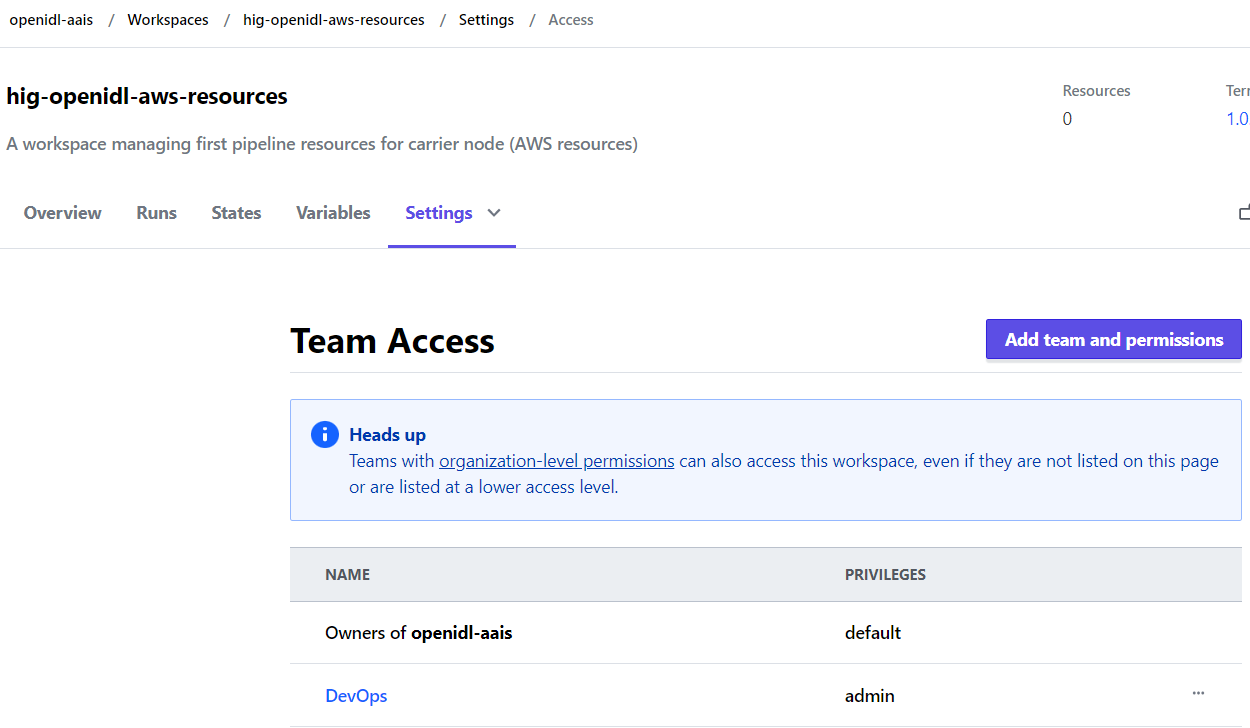
****

The below is an example screenshot of variable set showing subset of variables set.

****

## Team Access

Finally in each workspace configured enable team access in case team token is chosen as preferred method for API access.

****

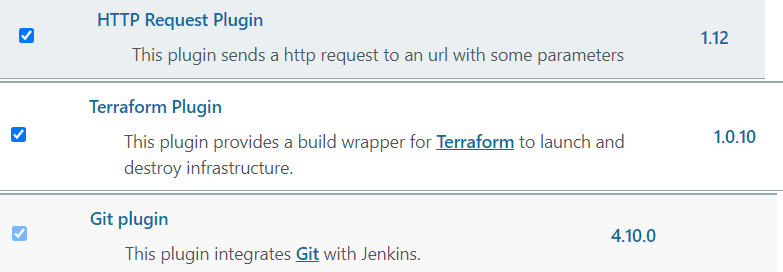
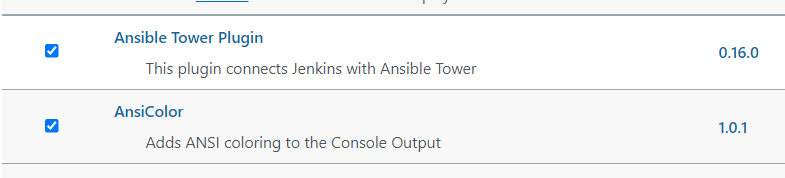
# Jenkin’s Environment

1. Plugins
2. Node labels
3. Global tools configuration
4. Configure System – Ansible Tower/AWX
5. Credentials

## Plugins required

The following are the additional plugins required to enable other than standard plugins which are installed during initial Jenkins’s setup.

1. HTTP Request Plugin
2. Source Code Plugin (GitHub)
3. Terraform Plugin
4. Ansible Tower Plugin
5. AnsiColor



## Node labels

The Jenkins pipeline job code uses a node label “openidl”. Do either of the below.

1. Setup “openidl” as node label to an existing node
2. Setup a new node and label it to “openidl”
3. Update Jenkins pipeline code to fit to a label that refers to a node in your environment.

The steps to labeling a node is skipped as it can be handled by Jenkins’s administrator.

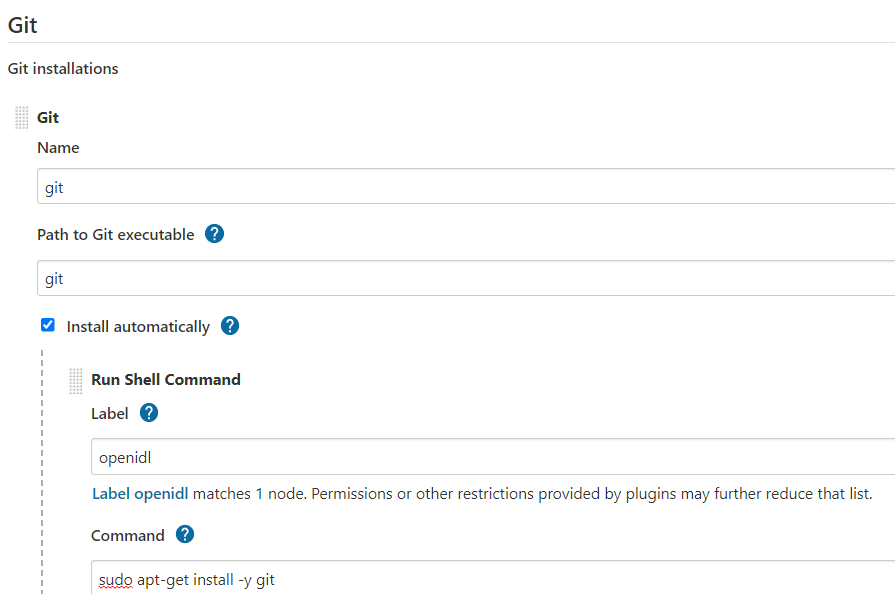
In case chosen to update the pipeline code with relevant node label. Refer to the pipeline code to the following section and replace “openidl” with custom label.

Go to the relevant repository and to the folder Jenkins-jobs/\*. For each job code, update as required.

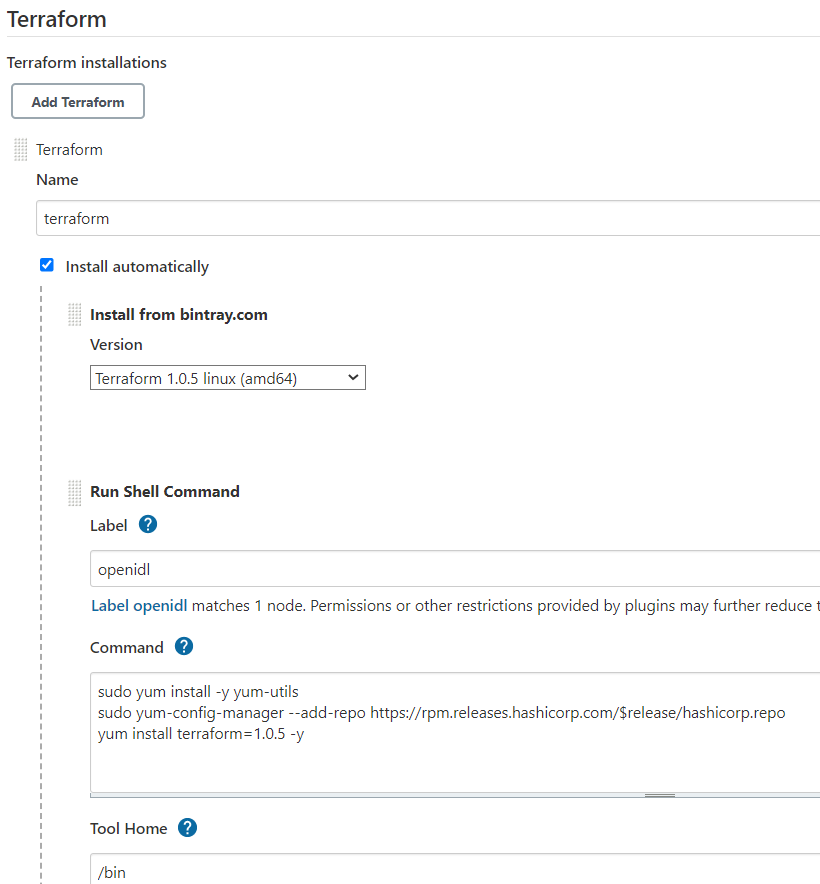
|  |  |
| --- | --- |
| node {  label “openidl”  } | node(‘openidl’) |

## Global tools configuration

1. Go to Jenkins => Mange Jenkins => Global Tool Configuration
2. Ensure Git and Terraform are configured according to your environment
3. Note the command shown here works for Ubuntu node and for Linux it will be different. Hence configure Git according to your nodes operating system.

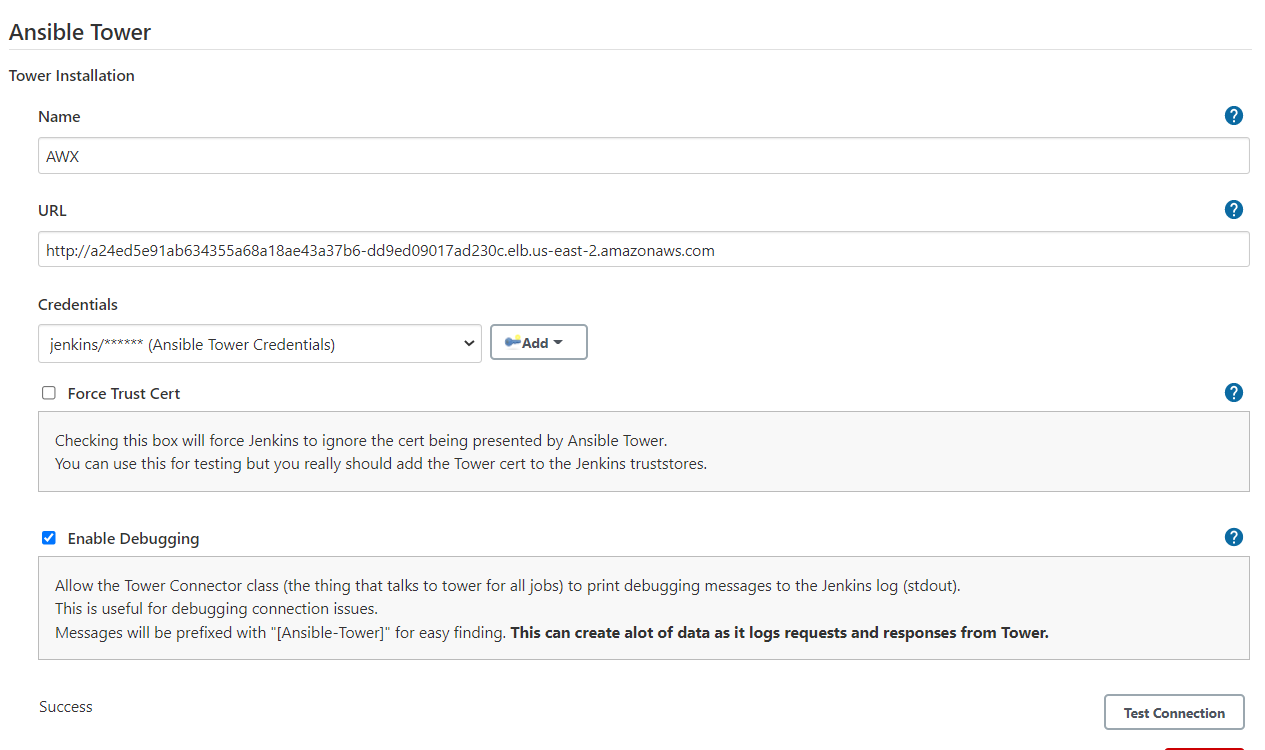


1. The below is just a sample for configuring global tool “terraform”. Ensure the right label of a node, version and installation commands are setup based on node label, operating system etc.



## Configure System (AWX/Ansible Tower)

1. Go to Jenkins => Manage Jenkins => Configure System
2. Go to Ansible Tower
3. Click on Add, Give a name to the instance “AWX”. Please note “AWX” instance name is used in Jenkins’s pipeline code. In case a different name is used, the pipeline code needs to be updated.
4. Update the actual URL of Ansible Tower/AWX instance to make API calls
5. Include the username/password to authenticate Jenkins in AWX/Tower. Hence get the user first created in AWX/Tower and get that credential added in Jenkins as username/password credential type before setting this up. Refer to the section <??> on how to setup a Jenkins username/password credential.
6. During development instance SSL is not used, however in production environment SSL should be enabled which is not documented here, refer to relevant Jenkins’s documentation on enabling SSL.
7. Test the connection between Jenkins and Ansible is successful to proceed further.



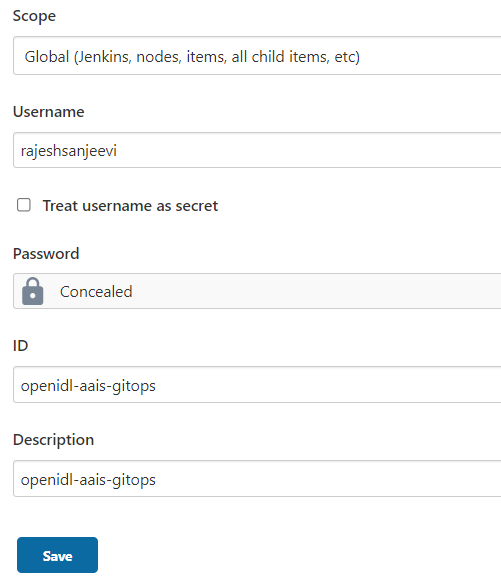
## Credentials

The following are the credential types used. The steps to create and configure detailed further in the document.

|  |  |  |  |
| --- | --- | --- | --- |
| **S No** | **Purpose** | **Credential Type** | **Description** |
| 1 | Jenkins access to AWX | Username and Password | An AWX user account having permissions to run jobs, access required credentials, project, and resources. A username and password are used. |
| 2 | Jenkins access to GitHub (source control) | Username and Password (PAT) | GitHub username and Personal Access Token. This is used by Jenkins to work with source control |
| 3 | Jenkins access to Terraform Cloud/Enterprise | Secret Text | A User/team token created in Terraform Cloud/Enterprise. Get that added as secret text in Jenkins. |

### 5.5.1 Username and Password Type

1. Login to Jenkins go to Manage Jenkins => Manage Credentials => Stores scoped to Jenkins (Jenkins) => Global Credentials (unrestricted) => Add credentials
2. Then choose Kind as “Username Password” and key in username, Password, Description and a unique ID which would be referred in the pipeline code. (An example below)



### 5.5.2 Secret Text Type

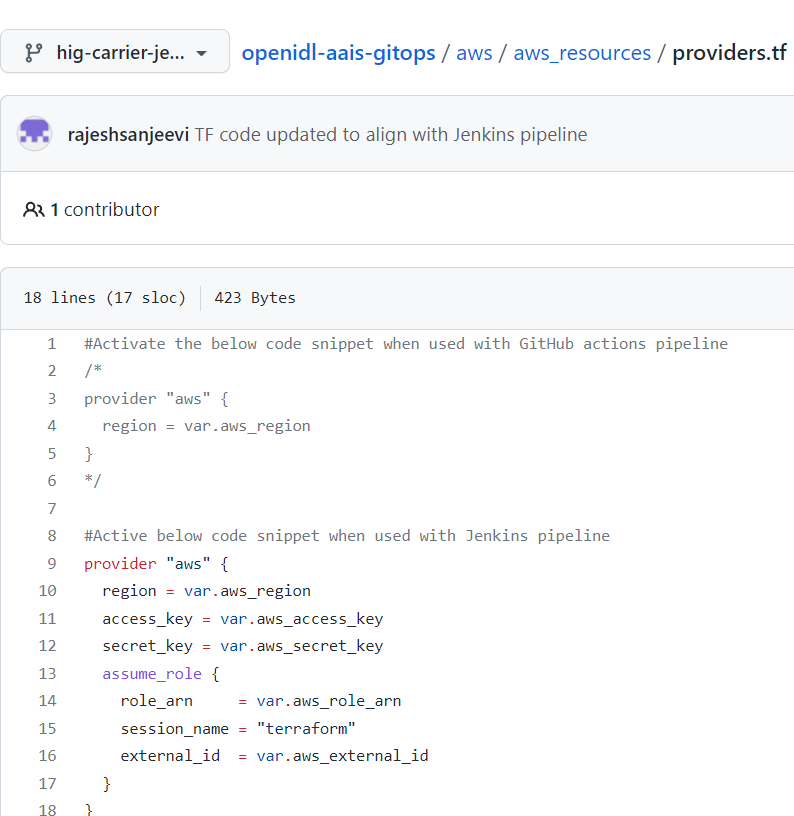
1. Login to Jenkins go to Manage Jenkins => Manage Credentials => Stores scoped to Jenkins (Jenkins) => Global Credentials (unrestricted) => Add credentials
2. Choose Kind as secret text, enter secret text like Token in “secret” field and name the secret ID as unique since it will be used in pipeline code. (An example below)

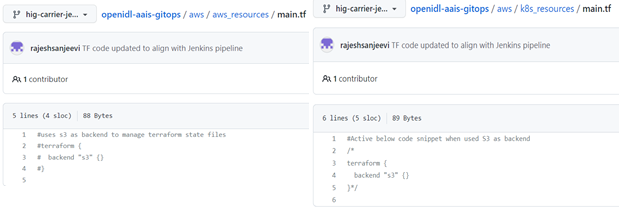


# Terraform code changes to adapt to Terraform Cloud/Enterprise

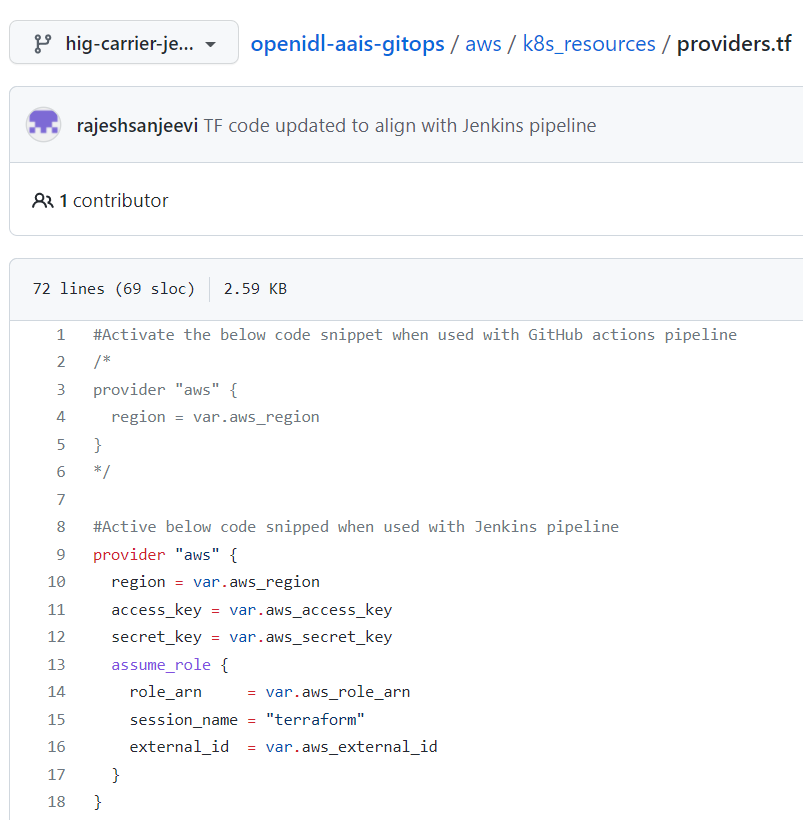
The following the major changes made to the Terraform code to adapt to Terraform Cloud/Enterprise.

Ensure that the code is updated before it is used.

1. Activate the right AWS provider configuration in the code for aws\_resources code set. 
2. Comment the terraform backend section of the code in both aws\_resources and k8s\_resources code set in the file main.tf. Below is an example.



1. Activate the AWS provider configuration as below for k8s\_resources code set and for remaining providers like Kubernetes and helm requires no changes.



1. Finally update/activate the code relevant code snippet as below for data.tf in k8s\_resources code set.



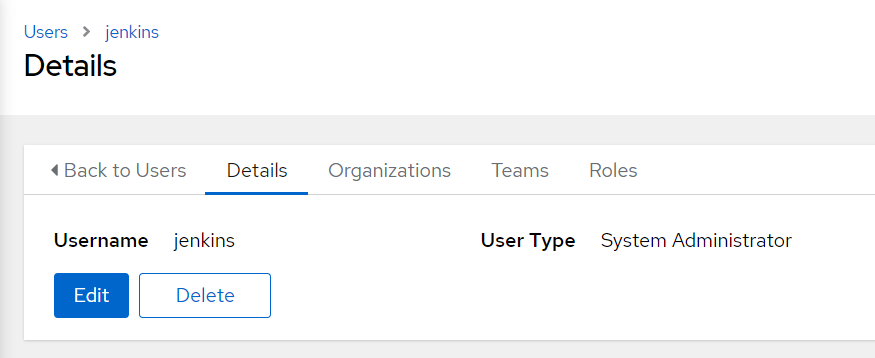
# Ansible Tower/AWX Environment

The following objects/items are required to setup for the pipelines to work.

1. A User Account
2. Credential Types
3. Inventory, Group and Host
4. Credentials
5. Projects
6. Templates

## 6.1 User Account

A user account to allow Jenkins to successfully work with Ansible Tower/AWX API. The user should have necessary permissions to run jobs and its relevant objects. In development used system administrator type, however in production use role-based access control using teams/roles.



## 6.2 Credential Types

For the OpenIDL deployment there are infrastructure and application related pipelines. They require specific credentials and additional variables. Hence custom credential types are used to simply the setup. The following are the credential types and steps to configure them.

1. OpenIDL-IAC => Used in infrastructure provisioning jobs
2. OpenIDL-APP => Used in application deployment jobs
3. OpenIDL-IAC-AWSUser-BAF => AWS IAM user credentials used with Blockchain automation jobs

### 6.2.1 OpenIDL-IAC

1. Login into Ansible Tower/AWX instance, Go to Administration

2. Go to Credential Types

3. Click on Add

4. Name it as OpenIDL-IAC and paste the below configuration in each relevant section.

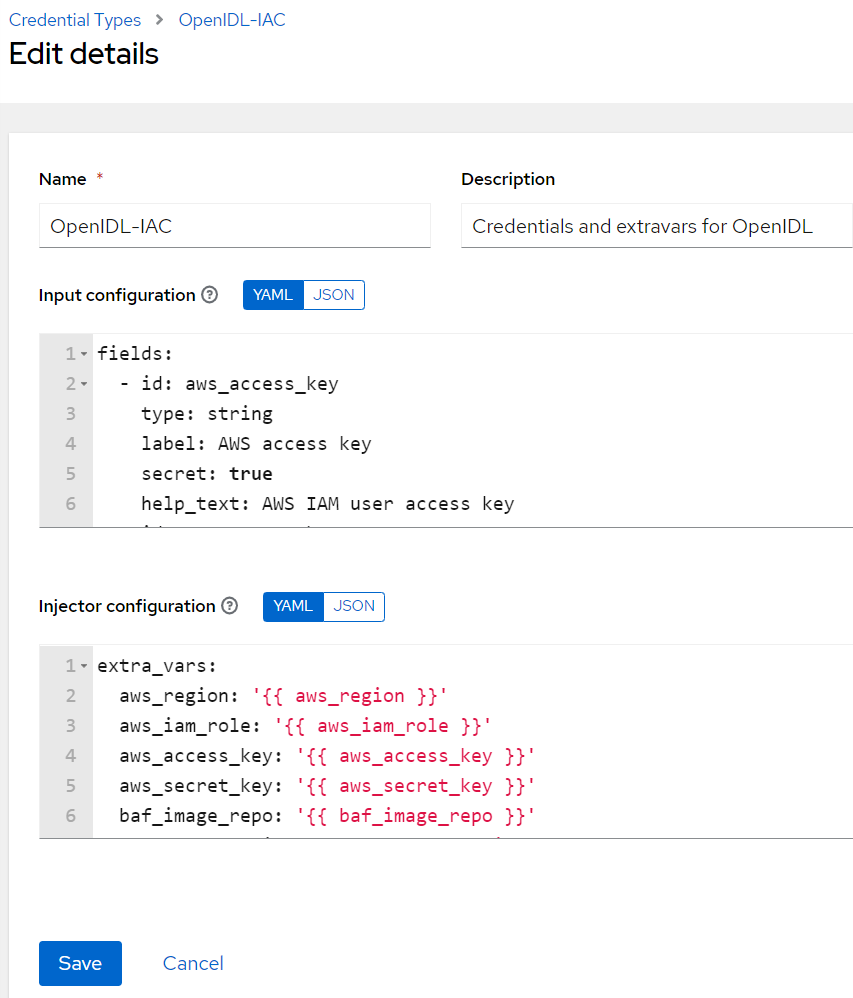
**Input Configuration**

|  |
| --- |
| fields:  - id: aws\_access\_key  type: string  label: AWS access key  secret: true  help\_text: AWS IAM user access key  - id: aws\_secret\_key  type: string  label: AWS secret key  secret: true  help\_text: AWS IAM user secret key  - id: aws\_iam\_role  type: string  label: AWS IAM role  help\_text: AWS IAM role to be assumed  - id: aws\_external\_id  type: string  label: AWS external id  help\_text: Externl ID set during IAM user/role configuration  - id: aws\_region  type: string  label: AWS region  help\_text: AWS Region  - id: aws\_account\_number  type: string  label: AWS account number  secret: true  help\_text: AWS account number  - id: baf\_image\_repo  type: string  label: BAF image repository  help\_text: Blockchain automation framework Docker image repository  - id: blk\_cluster\_name  type: string  label: Blockchain cluster name  help\_text: Blockchain EKS cluster name  - id: app\_cluster\_name  type: string  label: Application cluster name  help\_text: OpenIDL Application EKS cluster name  - id: gitops\_repo\_url  type: string  label: 'Gitops repository URL (without https://)'  help\_text: Github repository URL  - id: gitops\_repo\_branch  type: string  label: Gitops repository branch  help\_text: Branch name in Github repository  - id: gitops\_repo\_user  type: string  label: Gitops repository user  help\_text: GITHUB repository user  - id: gitops\_repo\_user\_token  type: string  label: Gitops repository user PAT  secret: true  help\_text: GITHUB repository user token  - id: gitops\_repo\_user\_email  type: string  label: Gitops repository user email  help\_text: GITHUB repository user email id  required:  - aws\_access\_key  - aws\_secret\_key  - aws\_iam\_role  - aws\_external\_id  - aws\_region  - aws\_account\_number  - blk\_cluster\_name  - baf\_image\_repo  - app\_cluster\_name  - gitops\_repo\_user  - gitops\_repo\_user\_email  - gitops\_repo\_user\_token |

**Injector Configuration**

|  |
| --- |
| extra\_vars:  aws\_region: '{{ aws\_region }}'  aws\_iam\_role: '{{ aws\_iam\_role }}'  aws\_access\_key: '{{ aws\_access\_key }}'  aws\_secret\_key: '{{ aws\_secret\_key }}'  baf\_image\_repo: '{{ baf\_image\_repo }}'  aws\_external\_id: '{{ aws\_external\_id }}'  gitops\_repo\_url: '{{ gitops\_repo\_url }}'  app\_cluster\_name: '{{ app\_cluster\_name }}'  blk\_cluster\_name: '{{ blk\_cluster\_name }}'  gitops\_repo\_user: '{{ gitops\_repo\_user }}'  aws\_account\_number: '{{ aws\_account\_number }}'  gitops\_repo\_branch: '{{ gitops\_repo\_branch }}'  gitops\_repo\_user\_email: '{{ gitops\_repo\_user\_email }}'  gitops\_repo\_user\_token: '{{ gitops\_repo\_user\_token }}' |

1. Save and close, screenshot below.



### 6.2.2 OpenIDL-IAC-AWSUser-BAF

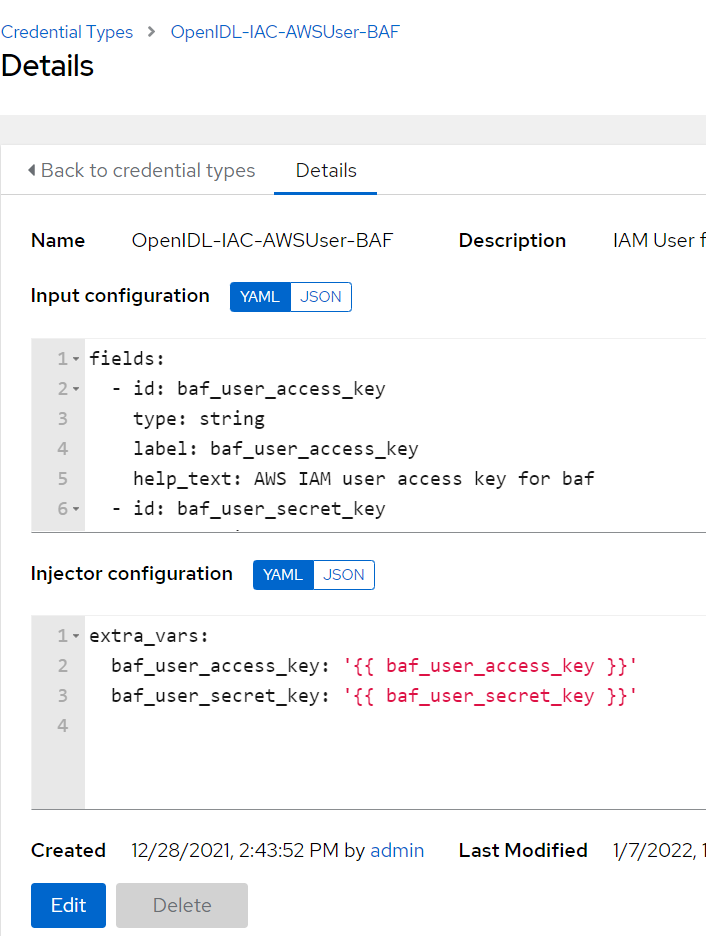
Similarly repeat the above steps to setup this credential type as well.

**Input Configuration**

|  |
| --- |
| fields:  - id: baf\_user\_access\_key  type: string  label: baf\_user\_access\_key  help\_text: AWS IAM user access key for baf  - id: baf\_user\_secret\_key  type: string  label: baf\_user\_secret\_key  secret: true  help\_text: AWS IAM user secret key for baf  required:  - baf\_user\_access\_key  - baf\_user\_secret\_key |

**Injector Configuration**

|  |
| --- |
| extra\_vars:  baf\_user\_access\_key: '{{ baf\_user\_access\_key }}'  baf\_user\_secret\_key: '{{ baf\_user\_secret\_key }}' |



### OpenIDL-APP

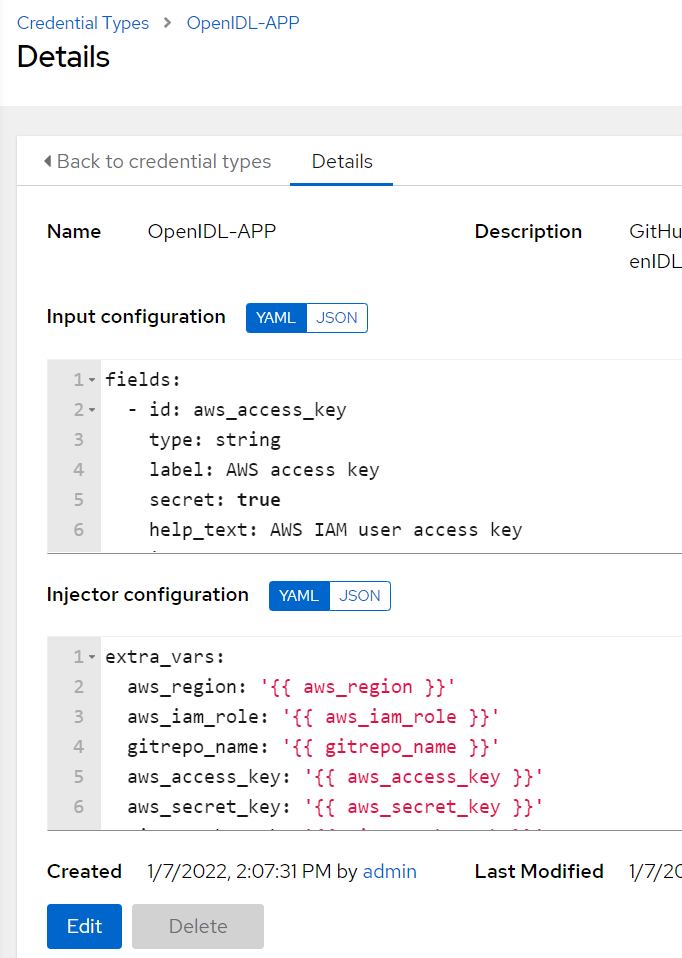
Similarly repeat the above steps to setup this credential type as well.

**Input Configuration**

|  |
| --- |
| fields:  - id: aws\_access\_key  type: string  label: AWS access key  secret: true  help\_text: AWS IAM user access key  - id: aws\_secret\_key  type: string  label: AWS secret key  secret: true  help\_text: AWS IAM user secret key  - id: aws\_iam\_role  type: string  label: AWS IAM role  help\_text: AWS IAM role to be assumed  - id: aws\_external\_id  type: string  label: AWS external id  help\_text: Externl ID set during IAM user/role configuration  - id: aws\_region  type: string  label: AWS region  help\_text: AWS Region  - id: gitrepo\_name  type: string  label: 'Git Repository (without https://)'  help\_text: Git repository URL  - id: gitrepo\_branch  type: string  label: Git branch name  help\_text: Git repository branch name  - id: gitrepo\_username  type: string  label: Gitrepo username  help\_text: Git repository login username  - id: gitrepo\_pat  type: string  label: Gitrepo PAT  secret: true  help\_text: Git repository personl access token  - id: app\_cluster\_name  type: string  label: Application cluster name  help\_text: OpenIDL Application EKS cluster name  - id: vault\_secret\_name  type: string  label: vault secret name  help\_text: Vault secret name provisioned in AWS secrets manager  required:  - aws\_access\_key  - aws\_secret\_key  - aws\_iam\_role  - aws\_external\_id  - aws\_region  - gitrepo\_username  - gitrepo\_password  - gitrepo\_name  - gitrepo\_branch  - app\_cluster\_name  - vault\_secret\_name |

**Injector Configuration**

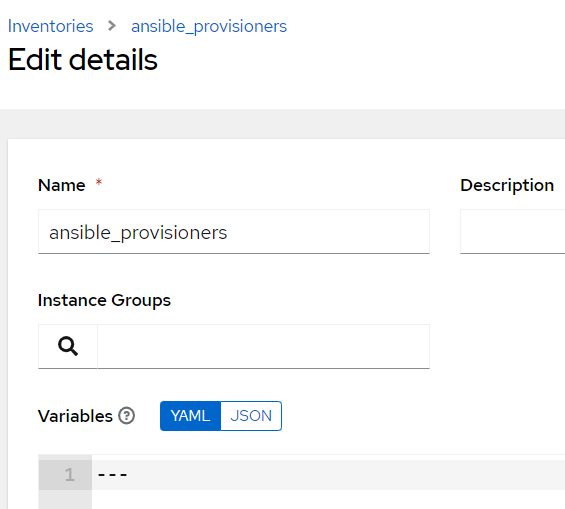
|  |
| --- |
| extra\_vars:  aws\_region: '{{ aws\_region }}'  aws\_iam\_role: '{{ aws\_iam\_role }}'  gitrepo\_name: '{{ gitrepo\_name }}'  aws\_access\_key: '{{ aws\_access\_key }}'  aws\_secret\_key: '{{ aws\_secret\_key }}'  gitrepo\_branch: '{{ gitrepo\_branch }}'  aws\_external\_id: '{{ aws\_external\_id }}'  app\_cluster\_name: '{{ app\_cluster\_name }}'  gitrepo\_password: '{{ gitrepo\_pat }}'  gitrepo\_username: '{{ gitrepo\_username }}'  vault\_secret\_name: '{{ vault\_secret\_name }}' |



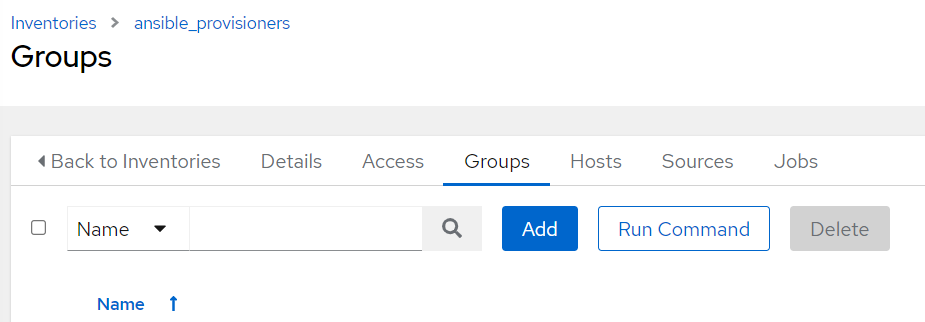
## Inventory, Group and Host

The OpenIDL ansible playbooks use the inventory group “ansible\_provisioners” and a localhost. Hence setup the relevant inventory, its group and host details in Ansible Tower/AWX.

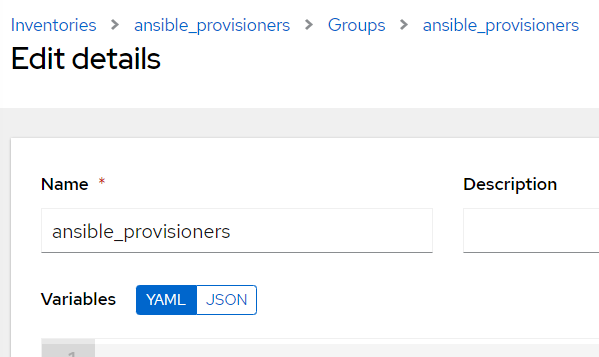
1. Login to the instance, go to Resources => Inventories => Add
2. Name it as “ansible\_provisioners and save.



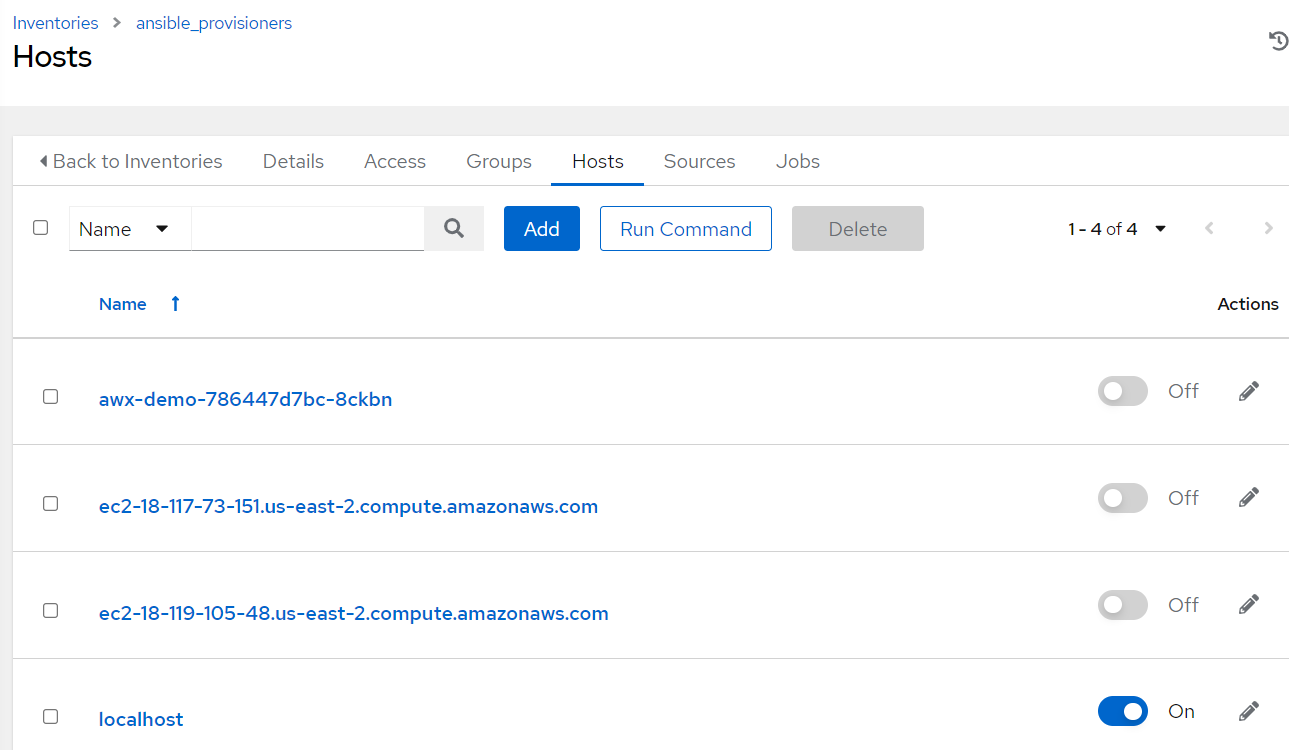
1. Now open the created inventory and go to Groups and click on Add



1. Name the group as “ansible\_provisioners”.



1. Now go to inventory ansible\_provisioners and then go to Hosts and click on Add to include localhost part of the group.



1. Note localhost is by default added to the inventory file part of playbook configuration in the repository. In case chosen to use alternate node then ensure that the relevant node is added to the ansible\_provisioners group/inventory and further the same host information is added to the inventory file located in the path “awx-automation/inventory/ansible\_provisioners” file in the github repositories as well. For example, below.
2. This entry should be added to both the repositories. (app and infra).



## Credentials

The following are the credentials to be configured in Ansible Tower/AWX.

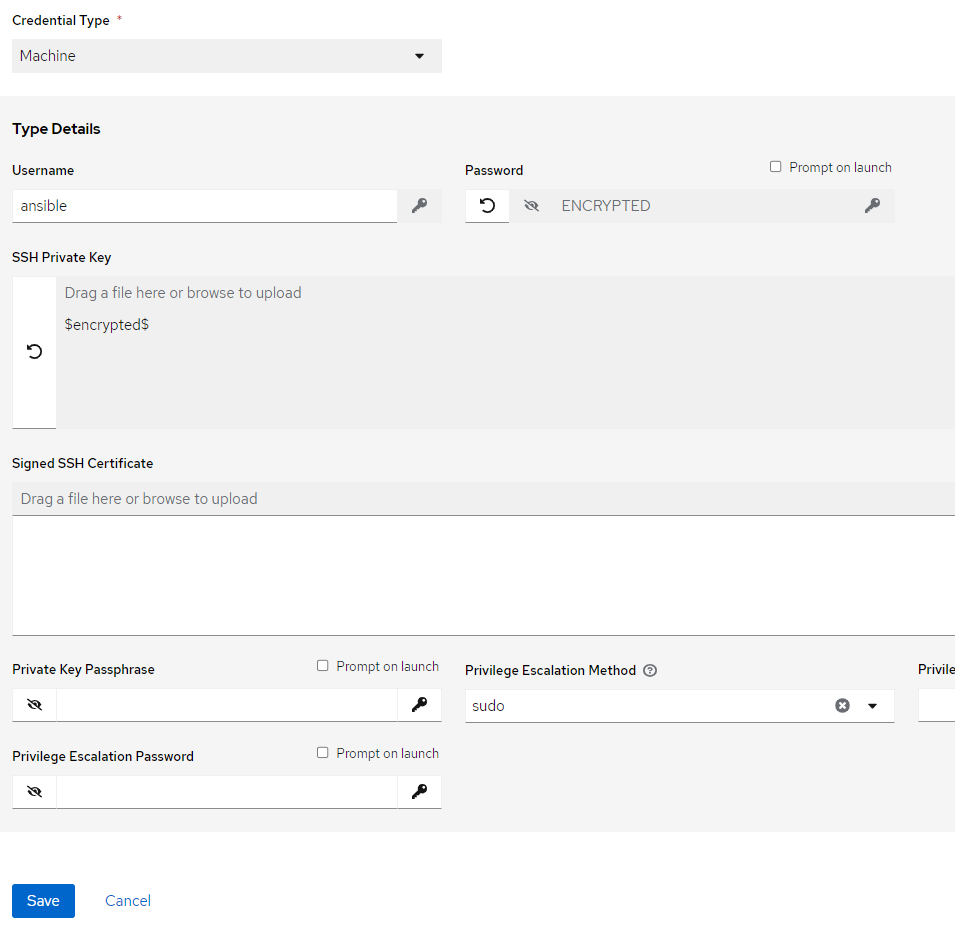
1. Machine credential
2. Source Control credential
3. OpenIDL-IAC
4. OpenIDL-APP
5. OpenIDL-IAC-AWSUser-BAF

### Machine Credential

This credential is used by the playbook to authenticate a host to run the playbooks. It may be either a localhost or remote node. This is a system SSH credential configured to allow Ansible instance to access a host to run the playbook.

First ensure the host (localhost/remotehost) is configured such access and further the credential is added here.

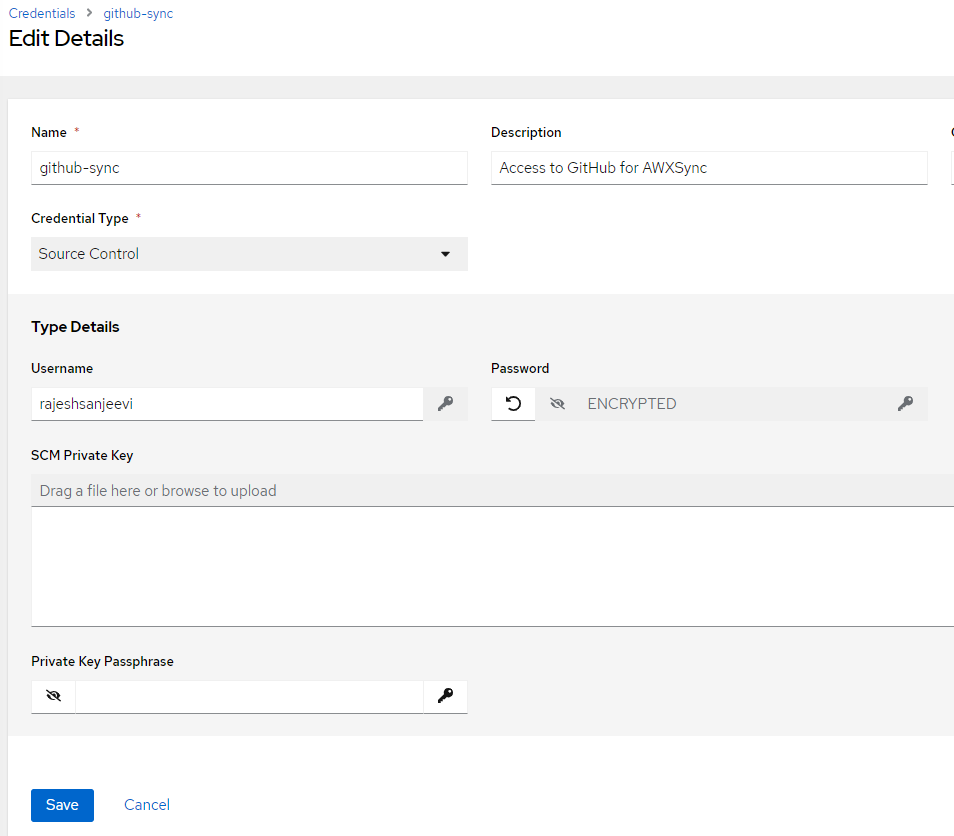
1. Go to Ansible instance => Resource => Credentials and add.
2. Ensure the credential type chosen is Machine
3. Enter a name, Input username of the account and add private key of the SSH key pair which is configured in the system and also enable privilege escalation method as sudo.
4. Note that in development an account “ansible” is used with sudo permissions. The account can be adjusted with permissions carefully reviewing the playbook actions and relevant permissions only to allow sudo commands.



### Source Control Credential

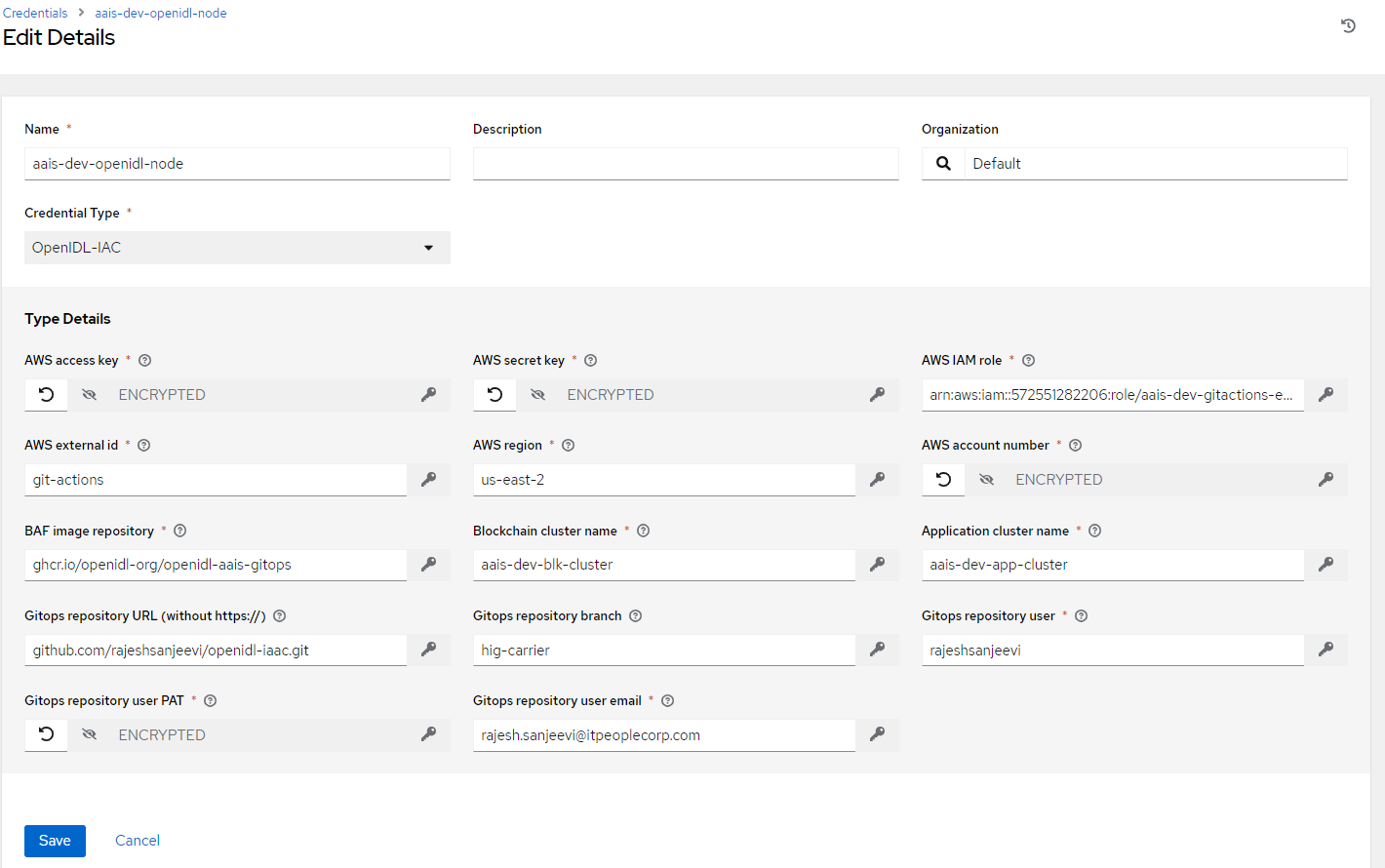
Similarly create a credential of type source control to enter GitHub user credential (username and PAT) to allow Ansible to successfully work with repositories. The screenshot for reference.

1. Name the credential
2. Select Type as Source Control
3. Enter the GitHub username and Personal Access Token (alternate is to use SSH method)



### OpenIDL-IAC

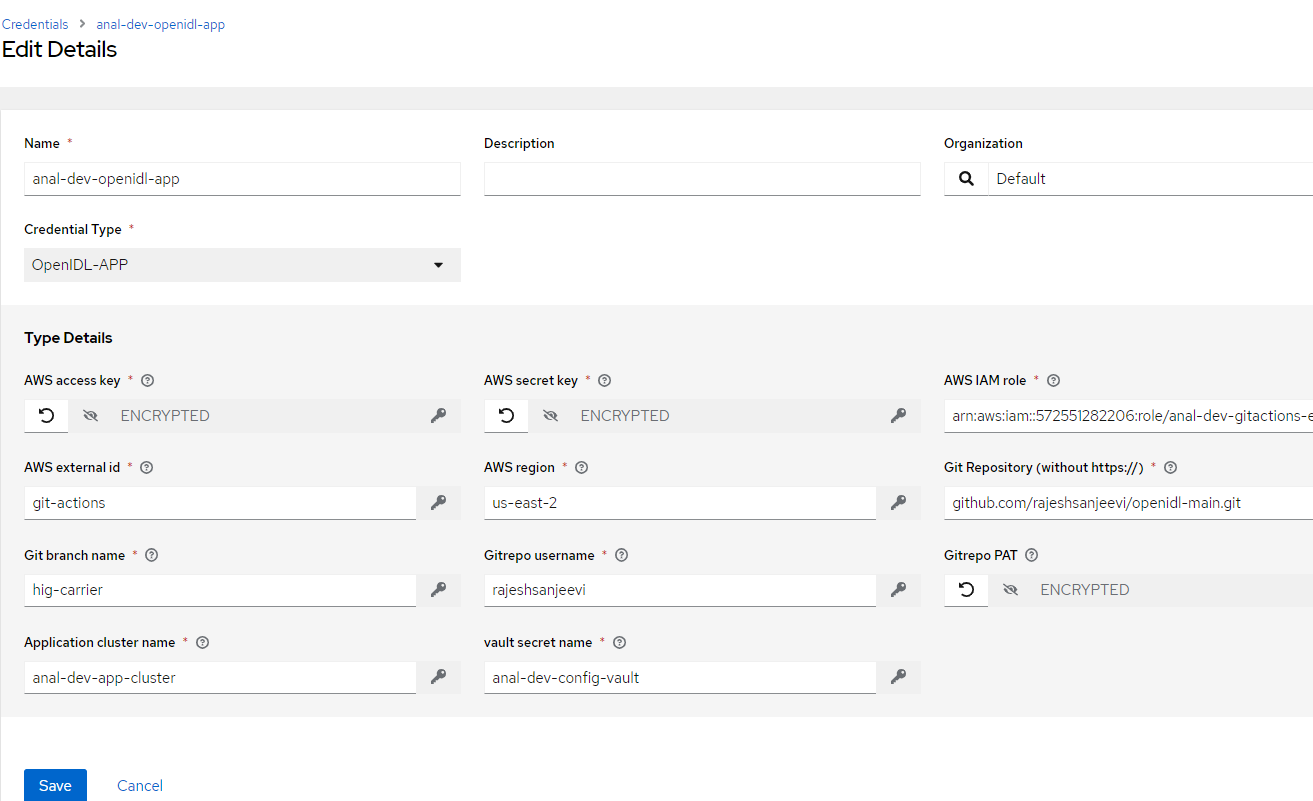
The next step is to use credential of type OpenIDL-IAC. This will be used by infrastructure jobs. A reference screenshot and significance of each field is detailed in below table.



|  |  |  |
| --- | --- | --- |
| **S No** | **Key** | **Description** |
| 1 | Credential Type | Select type OpenIDL-IAC |
| 2 | AWS access key | AWS access key of GitHub actions IAM user provisioned |
| 3 | AWS secret key | AWS secret key of GitHub actions IAM user provisioned |
| 4 | AWS IAM role | AWS IAM role provisioned to be assumed by GitHub actions IAM user |
| 5 | AWS external id | “git-actions” by default |
| 6 | AWS region | AWS region in which resources are provisioned |
| 7 | AWS account number | AWS account number |
| 8 | BAF image repository | The repository in which Docker image for Blockchain Automation Framework is located. Presently this is public repository. |
| 9 | Blockchain cluster name | Name of the cluster provisioned for blockchain resources |
| 10 | Application cluster name | Name of the cluster provisioned for application resources |
| 11 | Gitops repository URL | GitHub repository URL in which infrastructure code is located |
| 12 | Gitops repository branch | GitHub repository branch to be used |
| 13 | GitOps repository user | Username has access to the repository |
| 14 | Gitops repository user PAT | Personal access token of the user to authenticate with GitHub to use with ansible playbooks |
| 15 | Gitops repository user email | User email ID of the GitHub repository user used |

### OpenIDL-APP

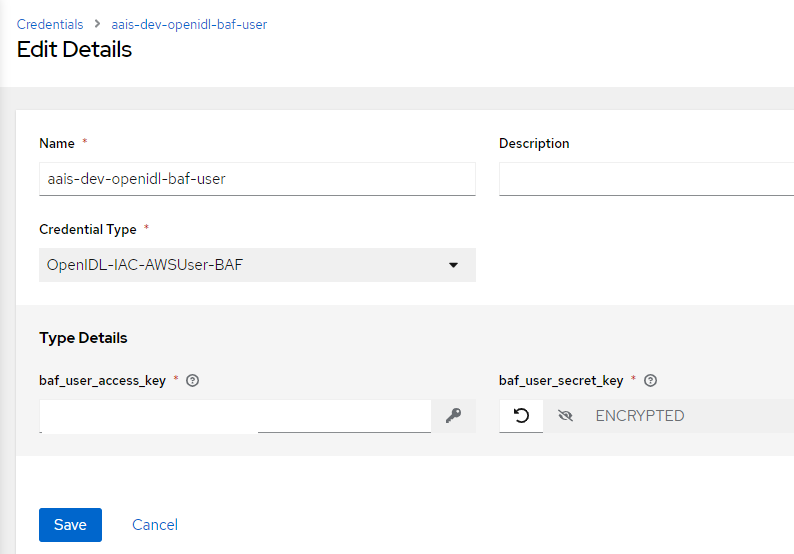
Create the credential of type OpenIDL-APP as described below which will be used by jobs related to OpenIDL application.



|  |  |  |
| --- | --- | --- |
| **S No** | **Key** | **Description** |
| 1 | Credential Type | OpenIDL-APP |
| 2 | AWS access key | AWS access key of GitHub actions IAM user provisioned |
| 3 | AWS secret key | AWS secret key of GitHub actions IAM user provisioned |
| 4 | AWS IAM role | AWS IAM role provisioned to be assumed by GitHub actions IAM user |
| 5 | AWS external id | “git-actions” by default |
| 6 | AWS region | AWS region in which resources are provisioned |
| 7 | Git Repository | GitHub repository related to applications |
| 8 | Git branch name | Name of the GitHub branch |
| 9 | Gitrepo username | Email id of the GitHub user used |
| 10 | Gitrepo PAT | Personal access token created previously to use with ansible playbooks |
| 11 | Application cluster name | Application cluster name |
| 12 | Vault Secret name | Secret created in AWS secret manage which holds credentials of vault. The standard format is <orgname>-<env>-config-vault |

### OpenIDL-IAC-AWSUser-BAF

Finally, provision credential of type OpenIDL-IAC-AWSUser-BAF. Choose the relevant credential type, key in AWS access key and secret key of AWS IAM user provisioned related to BAF.



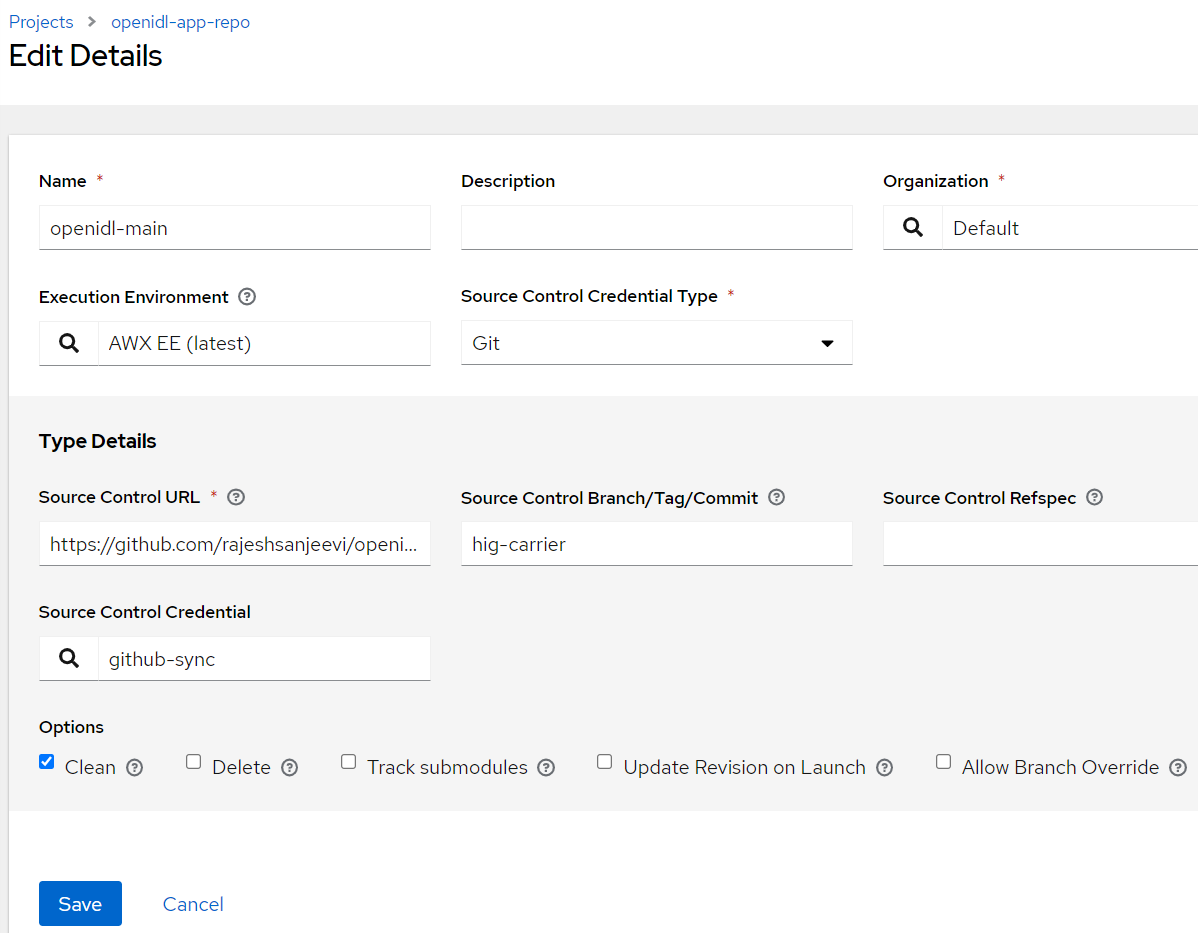
## 6.5 Projects

The next step is to configure projects which is used to pull the ansible playbook contents from GitHub to ansible tower/AWX.

1. openidl-main
2. openidl-aais-gitops

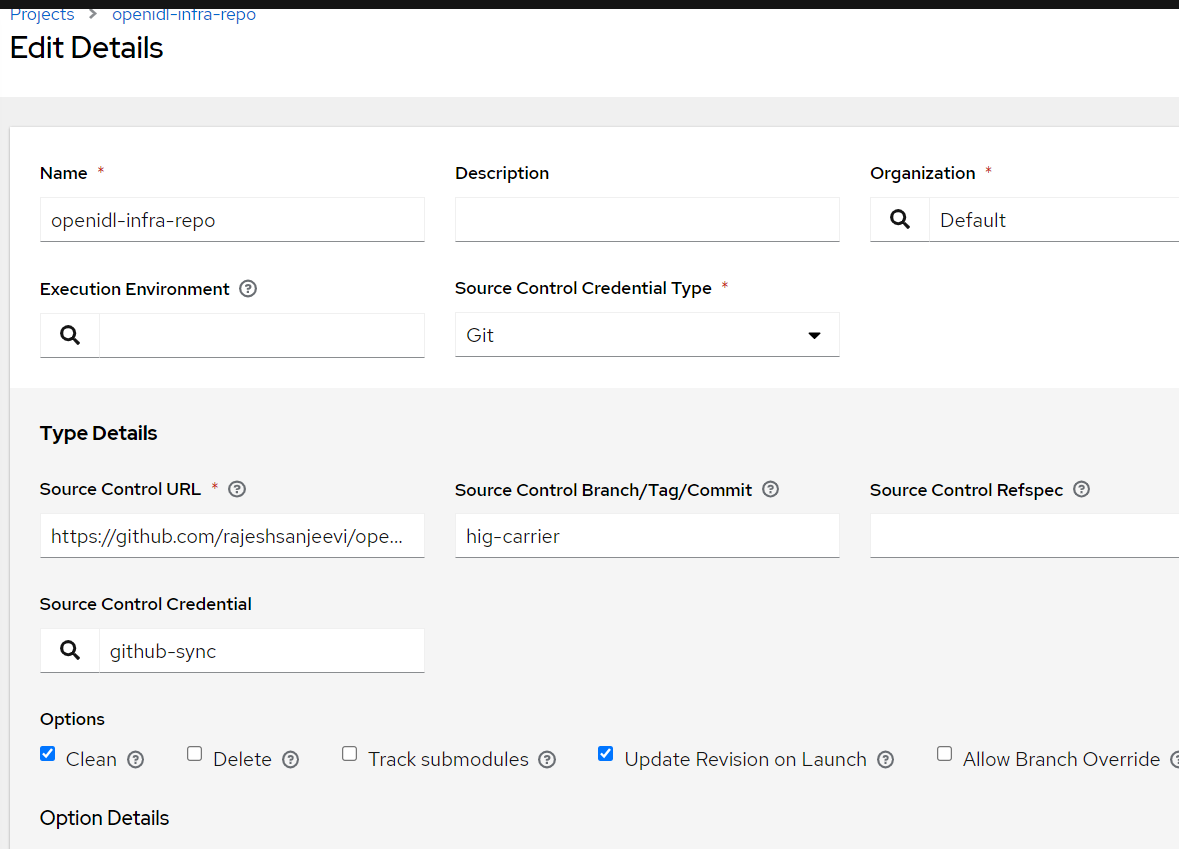
### openidl-main

1. Go to Ansible => Resources => Projects => Add and get the details added referring to the screenshot below.
2. Ensure the relevant organization, Execution environment are chosen
3. Select source control credential type as Git for GitHub.
4. Key in the Source Control URL, Branch and relevant code check in options
5. Finally choose the source control credential created previously to allow Ansible Tower/AWX to authenticate for syncing the code from repository to Ansible.



### openidl-aais-gitops

Repeat the same above steps to configure project for infrastructure code.



## 6.6 Templates

It is time to configure ansible job templates in Ansible Tower/AWX. The following are the list of job templates required to configure.

1. Vault install
2. MongoDB install
3. Blockchain
4. Register Users (BAF preregister users)
5. OpenIDL application secrets install
6. OpenIDL application install

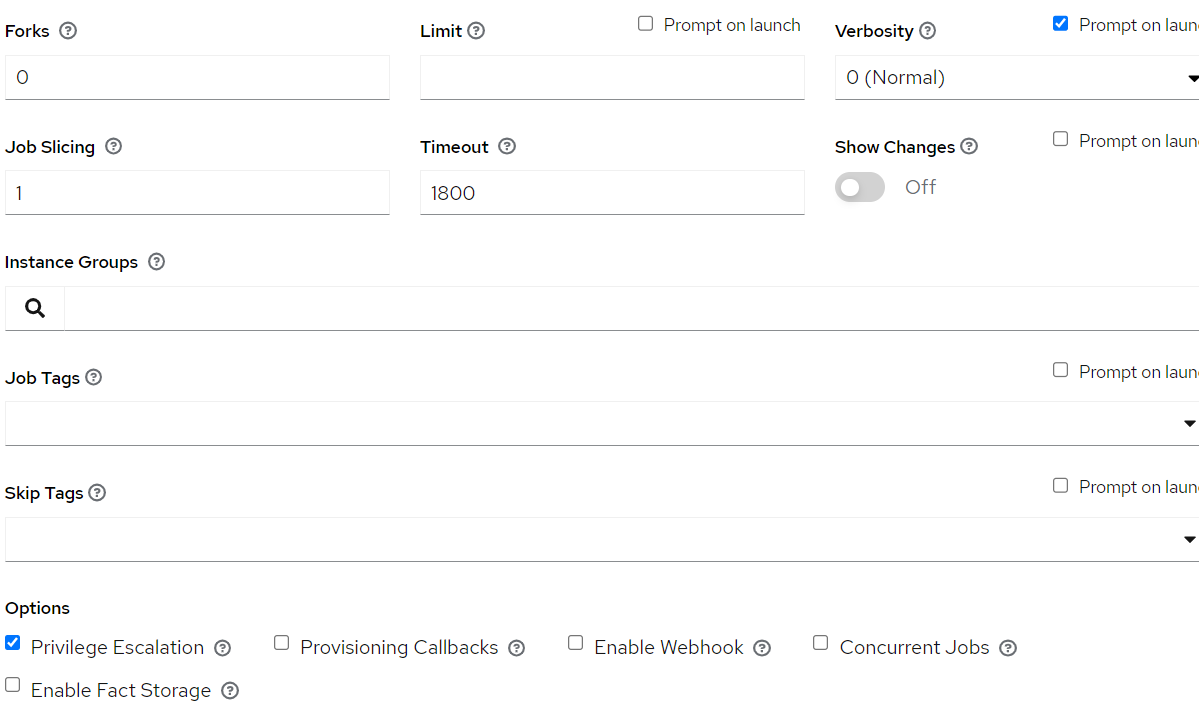
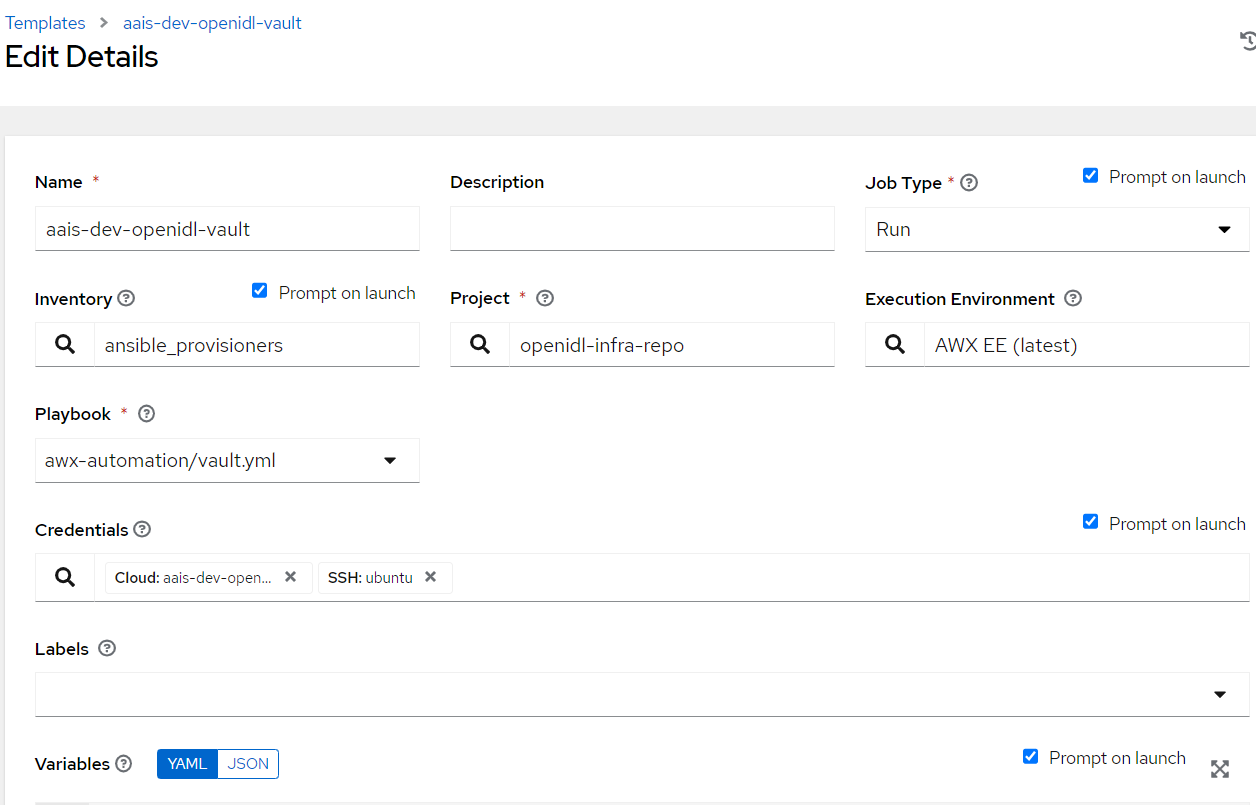
### Vault Install

1. Login to Ansible Tower/AWX, Go to Resources => Templates => Add
2. Key in Job name. The format is <org\_name>-<env>-openidl-vault.

**Org\_name:** First 4 characters of org\_name. Note Jenkins pipeline code refers to the job template name and hence it is vital.

**Env:** dev | test | prod

1. Select Job type as Run and check on Prompt on Launch
2. Choose inventory as ansible\_provisioners which was configured in previous step.
3. Choose the project that holds the IaC code. (openidl-aais-gitops) configured in previous step
4. Choose the relevant execution environment
5. Choose the playbook “awx-automation/vault.yml”.
6. Choose the following credentials.
   1. Machine credential configured in previous step
   2. OpenIDL-IAC credential configured in previous step
7. Choose prompt on Launch for variables (mandatory)
8. Set relevant verbosity level, Timeout at minimum 1800 seconds.
9. Set the Option “Privilege Escalation”.



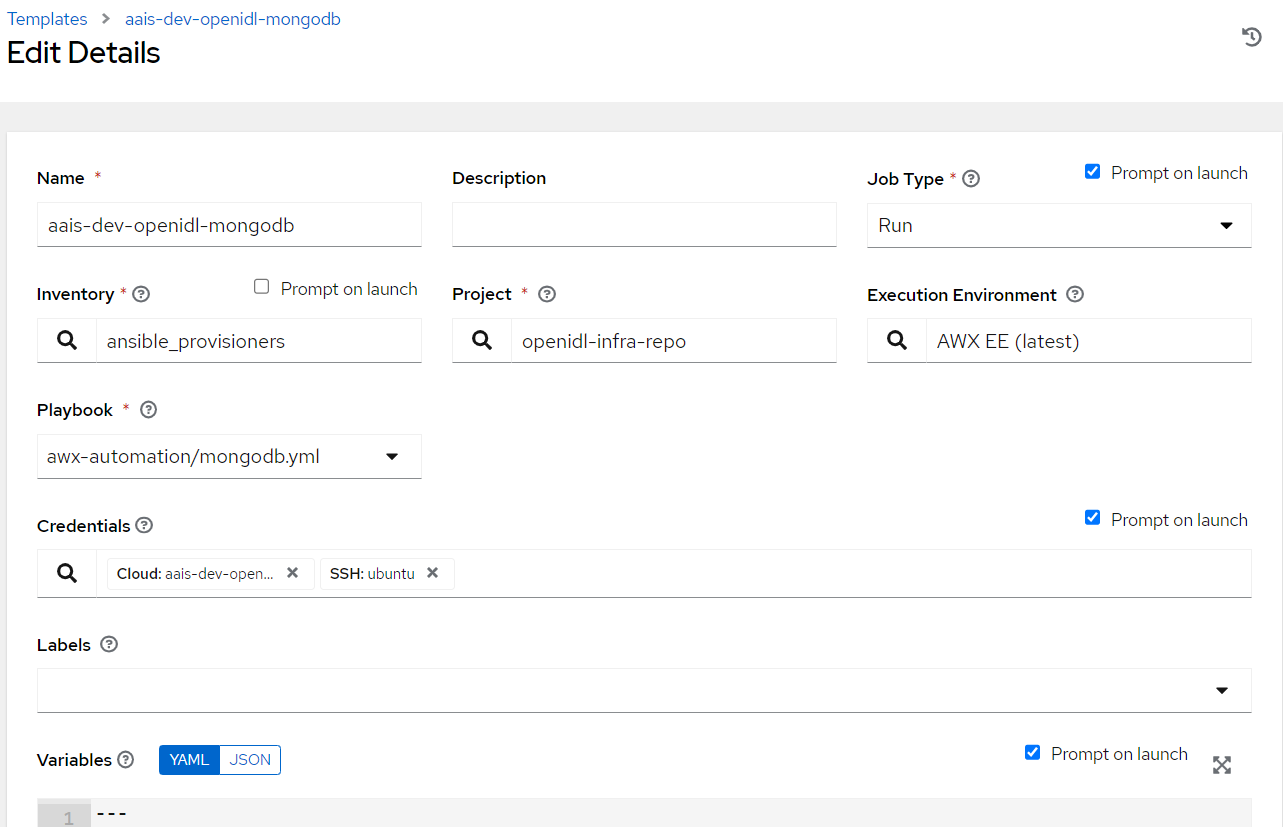
### MongoDB Install

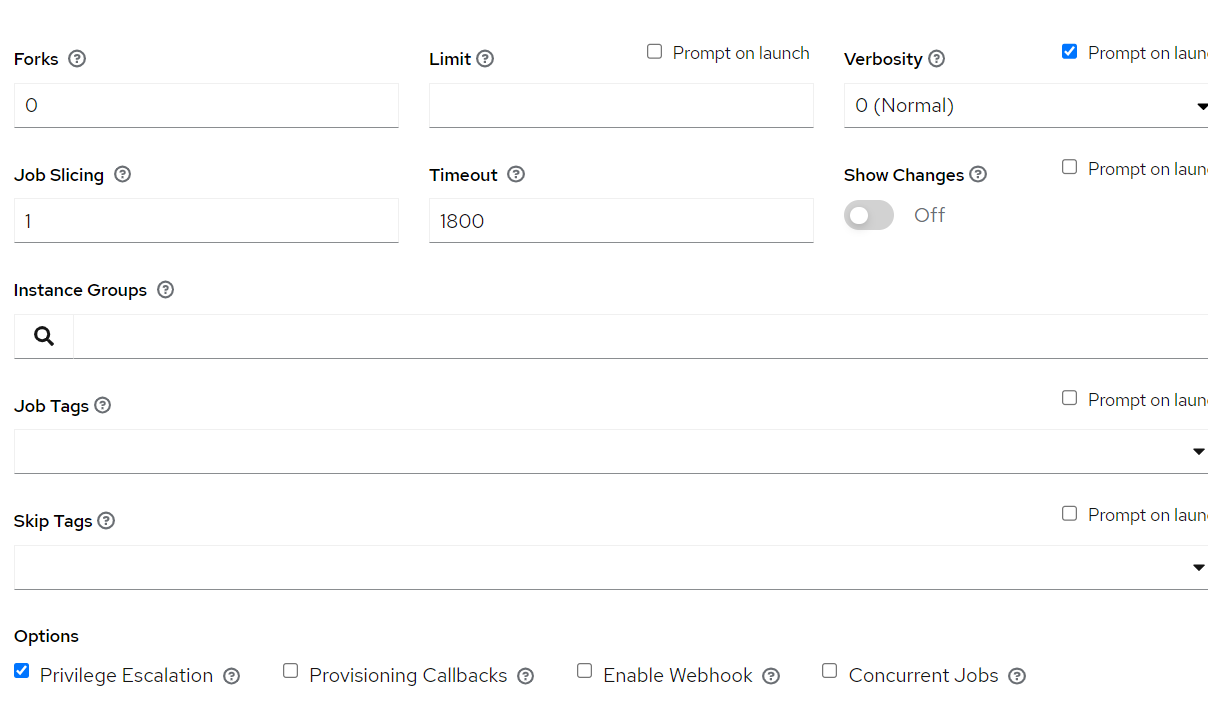
1. Login to Ansible Tower/AWX, Go to Resources => Templates => Add
2. Key in Job name. The format is <org\_name>-<env>-openidl-mongodb.

**Org\_name:** First 4 characters of org\_name. Note Jenkins pipeline code refers to the job template name and hence it is vital.

**Env:** dev | test | prod

1. Select Job type as Run and check on Prompt on Launch
2. Choose inventory as ansible\_provisioners which was configured in previous step.
3. Choose the project that holds the IaC code. (openidl-aais-gitops) configured in previous step
4. Choose the relevant execution environment
5. Choose the playbook “awx-automation/mongodb.yml”.
6. Choose the following credentials.
   1. Machine credential configured in previous step
   2. OpenIDL-IAC credential configured in previous step
7. Choose prompt on Launch for variables (mandatory)
8. Set relevant verbosity level, Timeout at minimum 1800 seconds.
9. Set the Option “Privilege Escalation”.





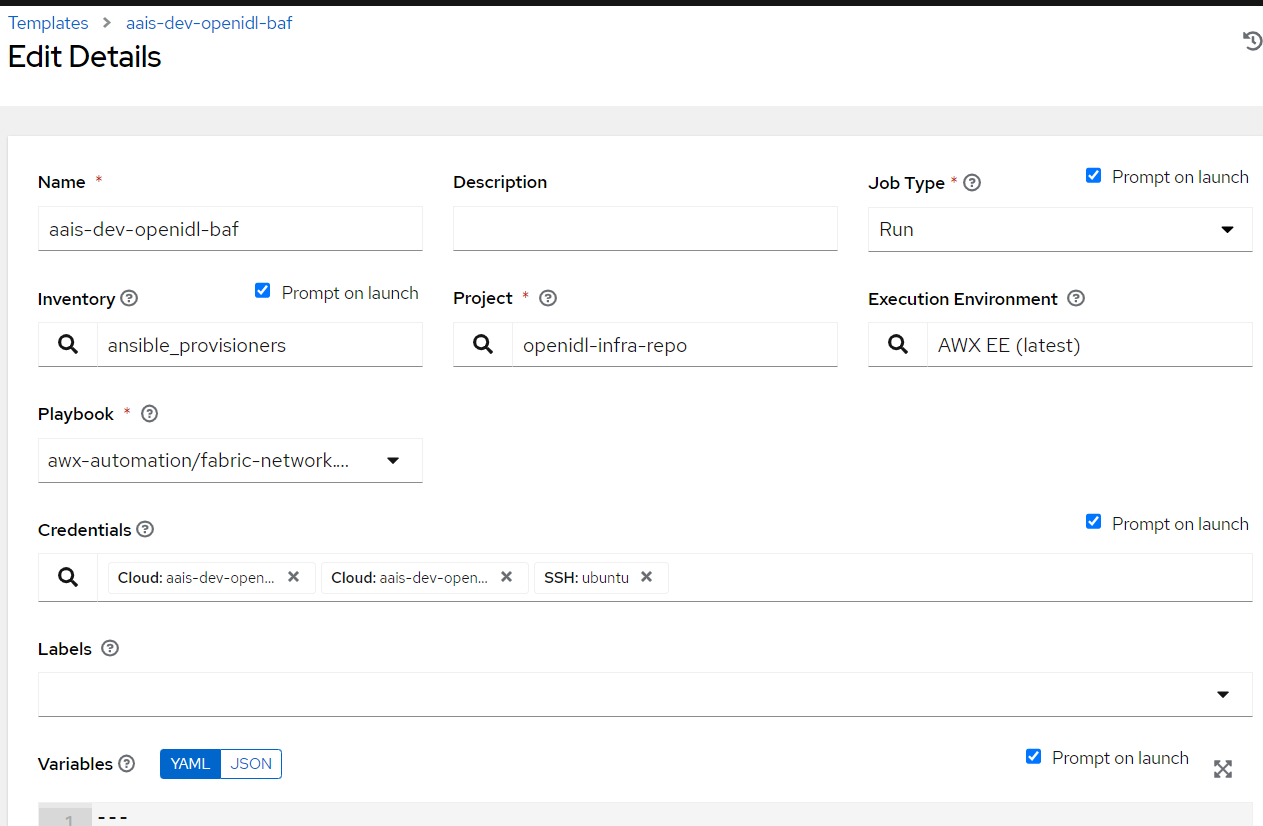
### BlockChain

1. Login to Ansible Tower/AWX, Go to Resources => Templates => Add
2. Key in Job name. The format is <org\_name>-<env>-openidl-baf.

**Org\_name:** First 4 characters of org\_name. Note Jenkins pipeline code refers to the job template name and hence it is vital.

**Env:** dev | test | prod

1. Select Job type as Run and check on Prompt on Launch
2. Choose inventory as ansible\_provisioners which was configured in previous step.
3. Choose the project that holds the IaC code. (openidl-aais-gitops) configured in previous step
4. Choose the relevant execution environment
5. Choose the playbook “awx-automation/fabric-network.yml”.
6. Choose the following credentials.
   1. Machine credential configured in previous step
   2. OpenIDL-IAC credential configured in previous step
   3. OpenIDL-IAC-AWSUser-BAF configured in previous step
7. Choose prompt on Launch for variables (mandatory)
8. Set relevant verbosity level, Timeout at minimum 0 seconds.
9. Set the Option “Privilege Escalation”.





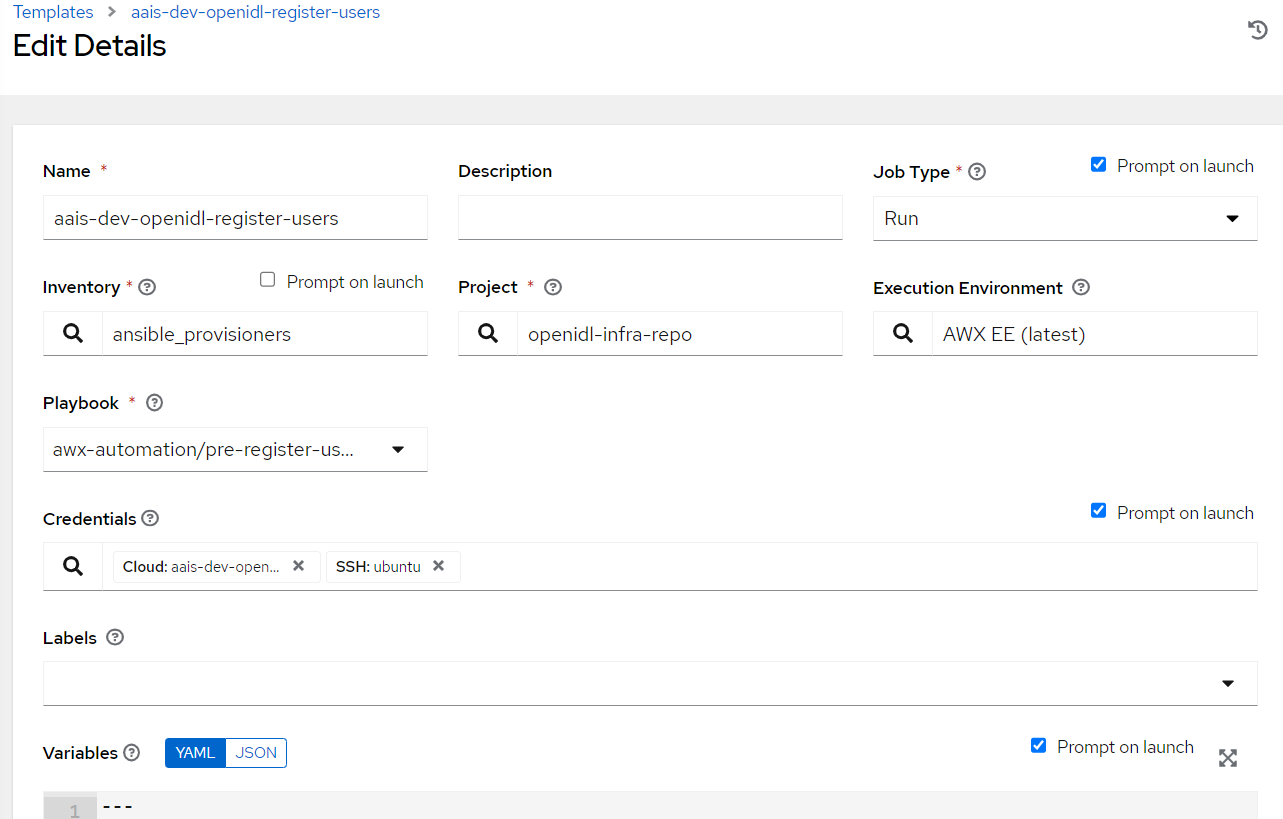
### Register Users

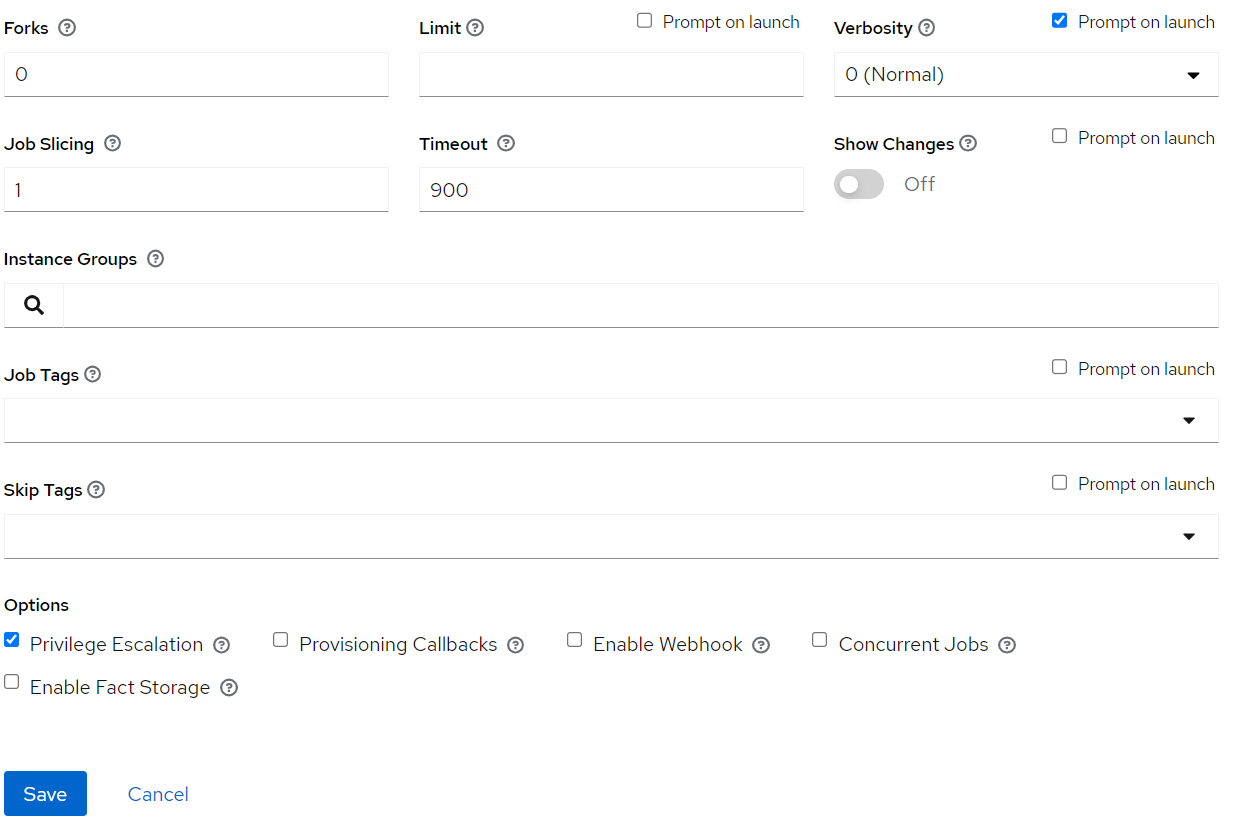
1. Login to Ansible Tower/AWX, Go to Resources => Templates => Add
2. Key in Job name. The format is <org\_name>-<env>-openidl-register-users

**Org\_name:** First 4 characters of org\_name. Note Jenkins pipeline code refers to the job template name and hence it is vital.

**Env:** dev | test | prod

1. Select Job type as Run and check on Prompt on Launch
2. Choose inventory as ansible\_provisioners which was configured in previous step.
3. Choose the project that holds the IaC code. (openidl-aais-gitops) configured in previous step
4. Choose the relevant execution environment
5. Choose the playbook “awx-automation/pre-register-users.yml”.
6. Choose the following credentials.
   1. Machine credential configured in previous step
   2. OpenIDL-IAC credential configured in previous step
7. Choose prompt on Launch for variables (mandatory)
8. Set relevant verbosity level, Timeout at minimum 900 seconds.
9. Set the Option “Privilege Escalation”.





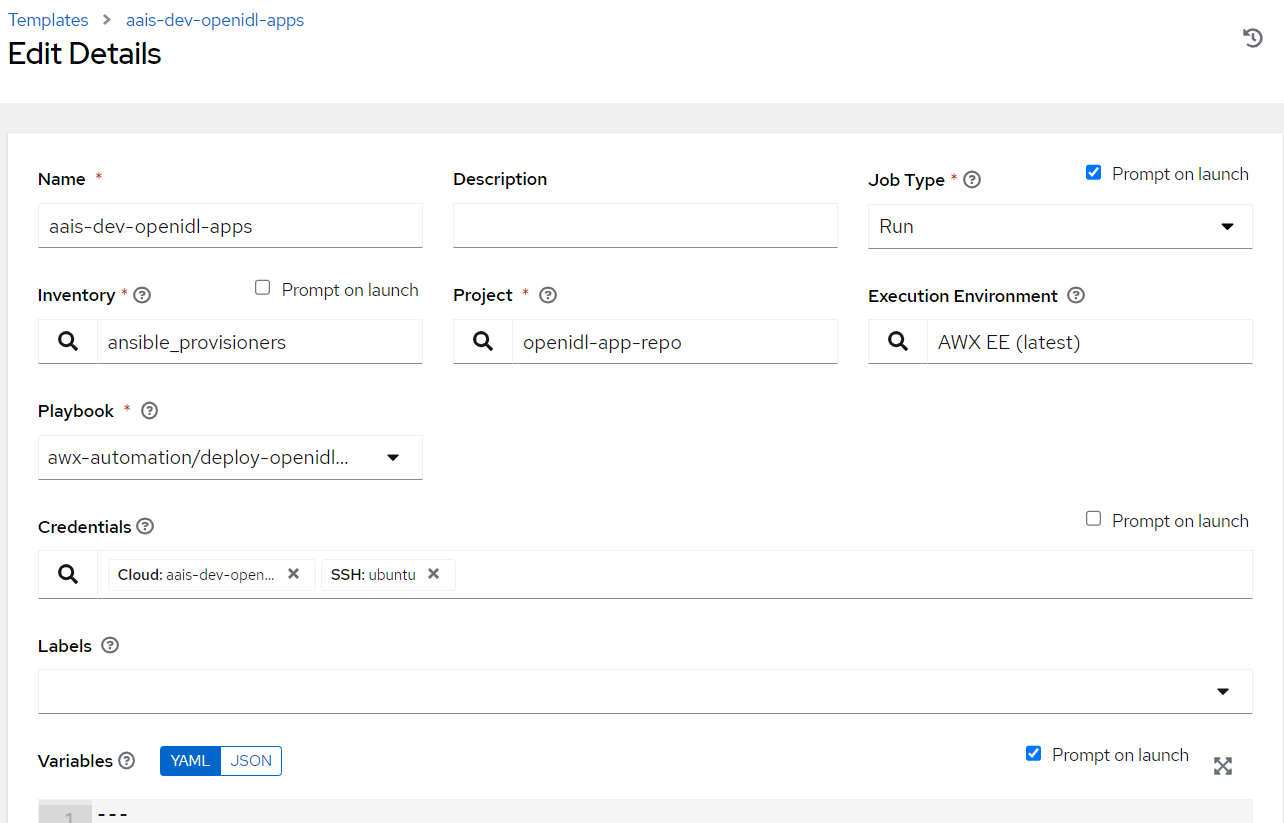
### OpenIDL Application Install

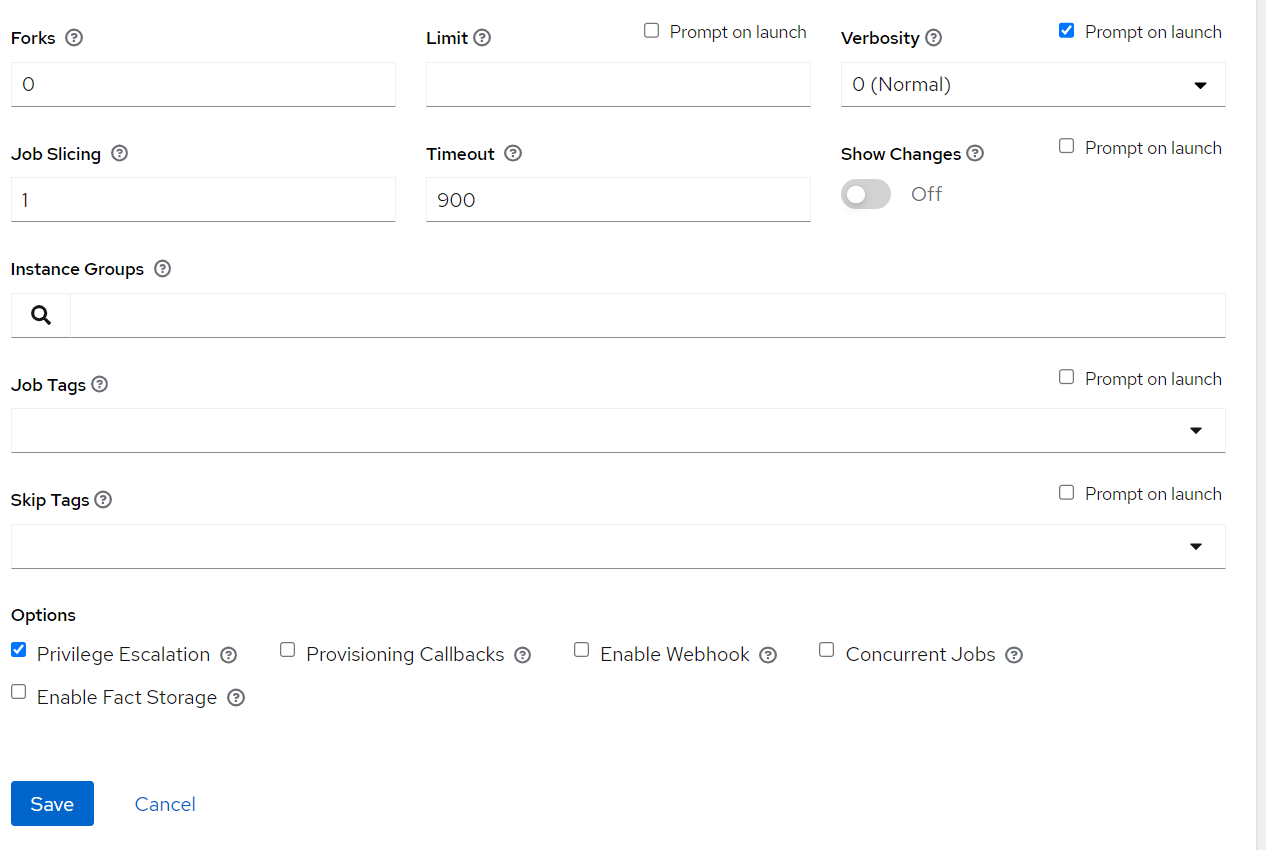
1. Login to Ansible Tower/AWX, Go to Resources => Templates => Add
2. Key in Job name. The format is <org\_name>-<env>-openidl-apps

**Org\_name:** First 4 characters of org\_name. Note Jenkins pipeline code refers to the job template name and hence it is vital.

**Env:** dev | test | prod

1. Select Job type as Run and check on Prompt on Launch
2. Choose inventory as ansible\_provisioners which was configured in previous step.
3. Choose the project that holds the application code. (openidl-main) configured in previous step
4. Choose the relevant execution environment
5. Choose the playbook “awx-automation/deploy-openidl-apps.yaml”.
6. Choose the following credentials.
   1. Machine credential configured in previous step
   2. OpenIDL-APP credential configured in previous step
7. Choose prompt on Launch for variables (mandatory)
8. Set relevant verbosity level, Timeout at minimum 900 seconds.
9. Set the Option “Privilege Escalation”.





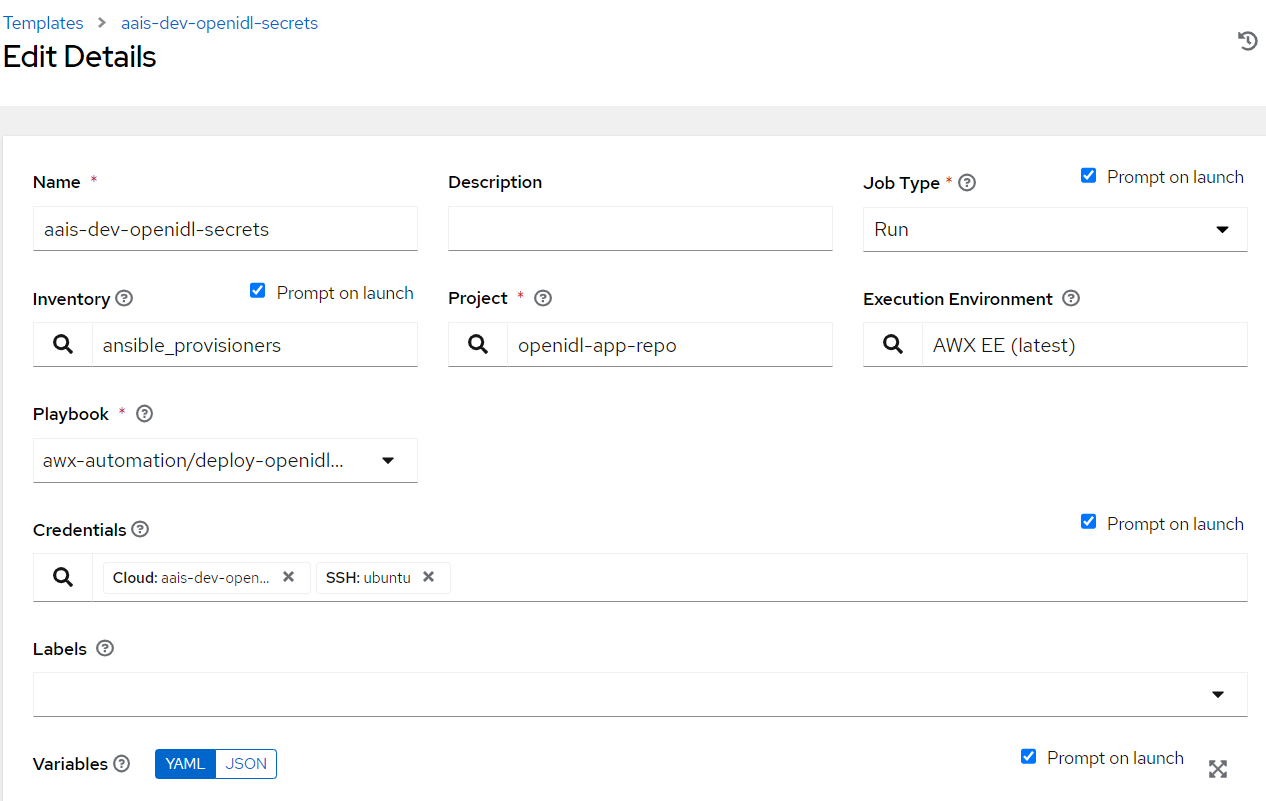
### OpenIDL Application Secrets Install

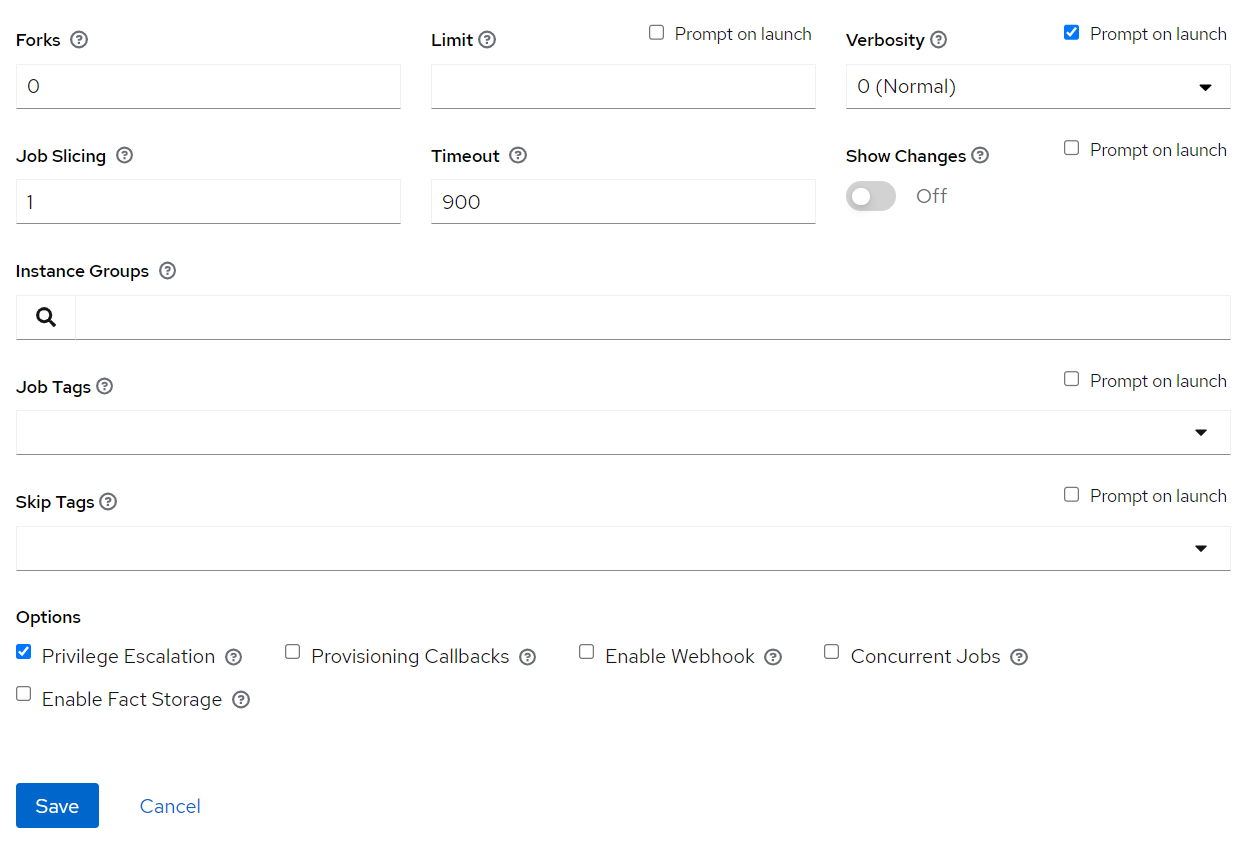
1. Login to Ansible Tower/AWX, Go to Resources => Templates => Add
2. Key in Job name. The format is <org\_name>-<env>-openidl-secrets

**Org\_name:** First 4 characters of org\_name. Note Jenkins pipeline code refers to the job template name and hence it is vital.

**Env:** dev | test | prod

1. Select Job type as Run and check on Prompt on Launch
2. Choose inventory as ansible\_provisioners which was configured in previous step.
3. Choose the project that holds the application code. (openidl-main) configured in previous step
4. Choose the relevant execution environment
5. Choose the playbook “awx-automation/deploy-openidl-secrets.yaml”.
6. Choose the following credentials.
   1. Machine credential configured in previous step
   2. OpenIDL-APP credential configured in previous step
7. Choose prompt on Launch for variables (mandatory)
8. Set relevant verbosity level, Timeout at minimum 900 seconds.
9. Set the Option “Privilege Escalation”.





# Jenkins Job Configuration

### Credentials

1. Before configuring Jenkins’s job ensure that the required credentials relevant to the jobs are already configured in Jenkins.

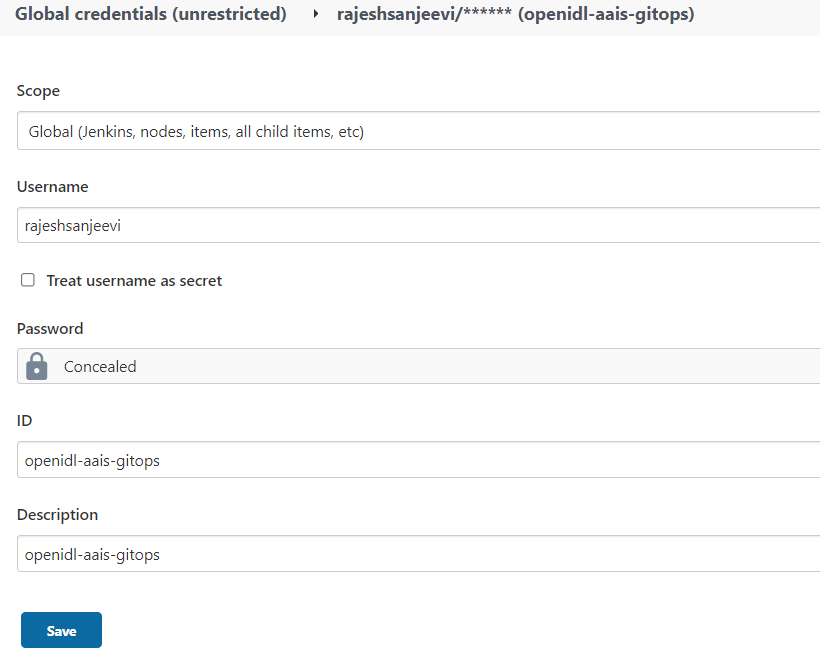
Terraform credentials

AWX (Ansible Tower/AWX User credentials)

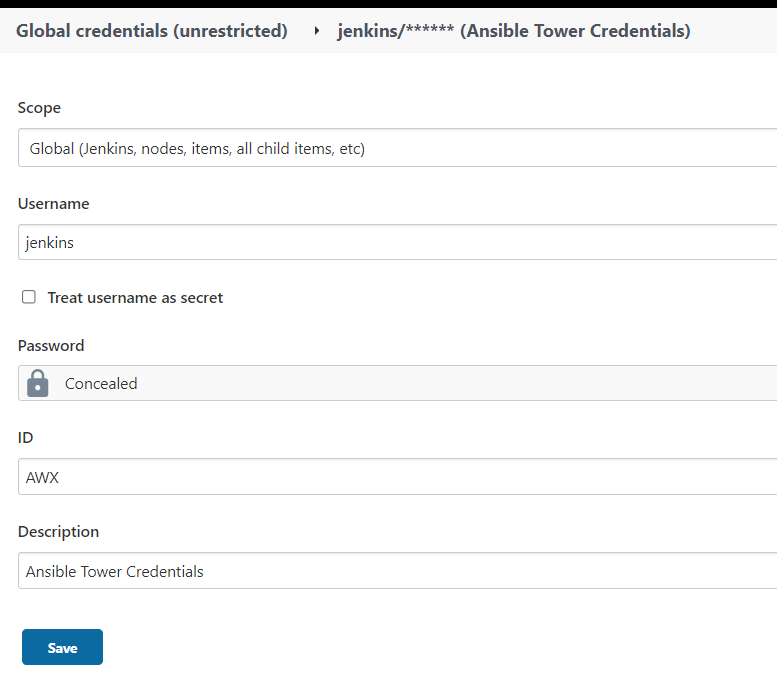
GitHub User credentials

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S No** | **Credential Type** | **ID** | **Username** | **Password/Secret** | **Description** |
| 1 | Username with password | openidl-aais-gitops | GitHub account username | Personal access token created | GitHub credentials |
| 2 | Username with password | AWX | Ansible tower username | Ansible tower user password | Ansible Tower/AWX credentials |
| 3 | Secret text | TF\_BEARER\_TOKEN | NA | Terraform user/team API token | Terraform Cloud/Enterprise access token |

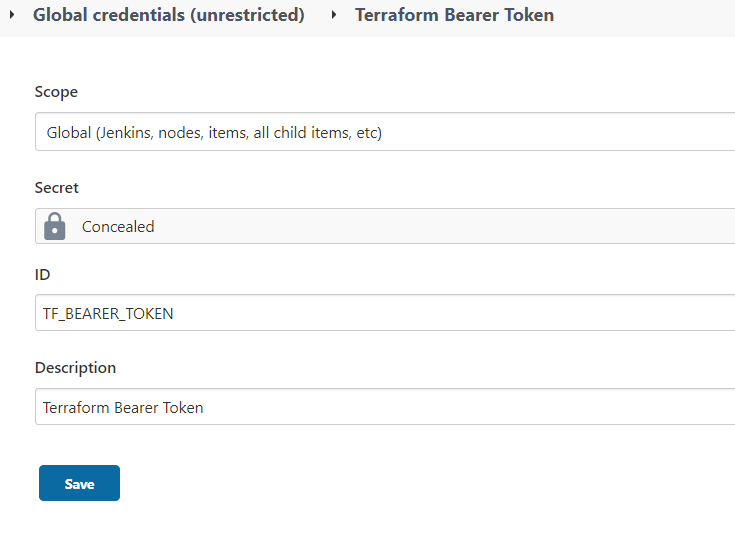
**References: GitHub credential**



**References: AWX credential**



**References: Terraform credential**



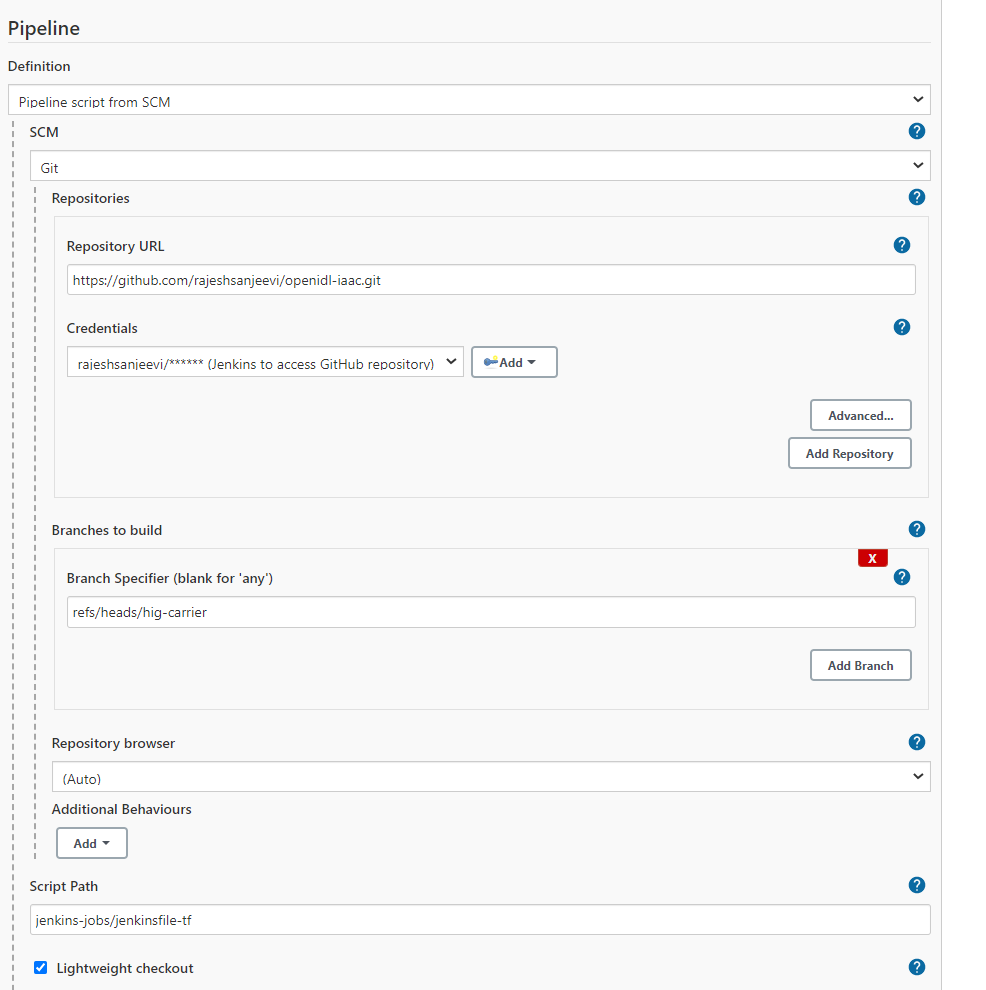
### Job Configurations

The list of jobs to be configured are

1. Job to provision AWS resources and K8s resources using Terraform Cloud/Enterprise
2. Job to provision Vault using Ansible Tower/AWX
3. Job to provision Blockchain Network using Ansible Tower/AWX
4. Job to provision MongoDB using Ansible Tower/AWX
5. Job to provision OpenIDL application secrets and application using Ansible Tower/AWX

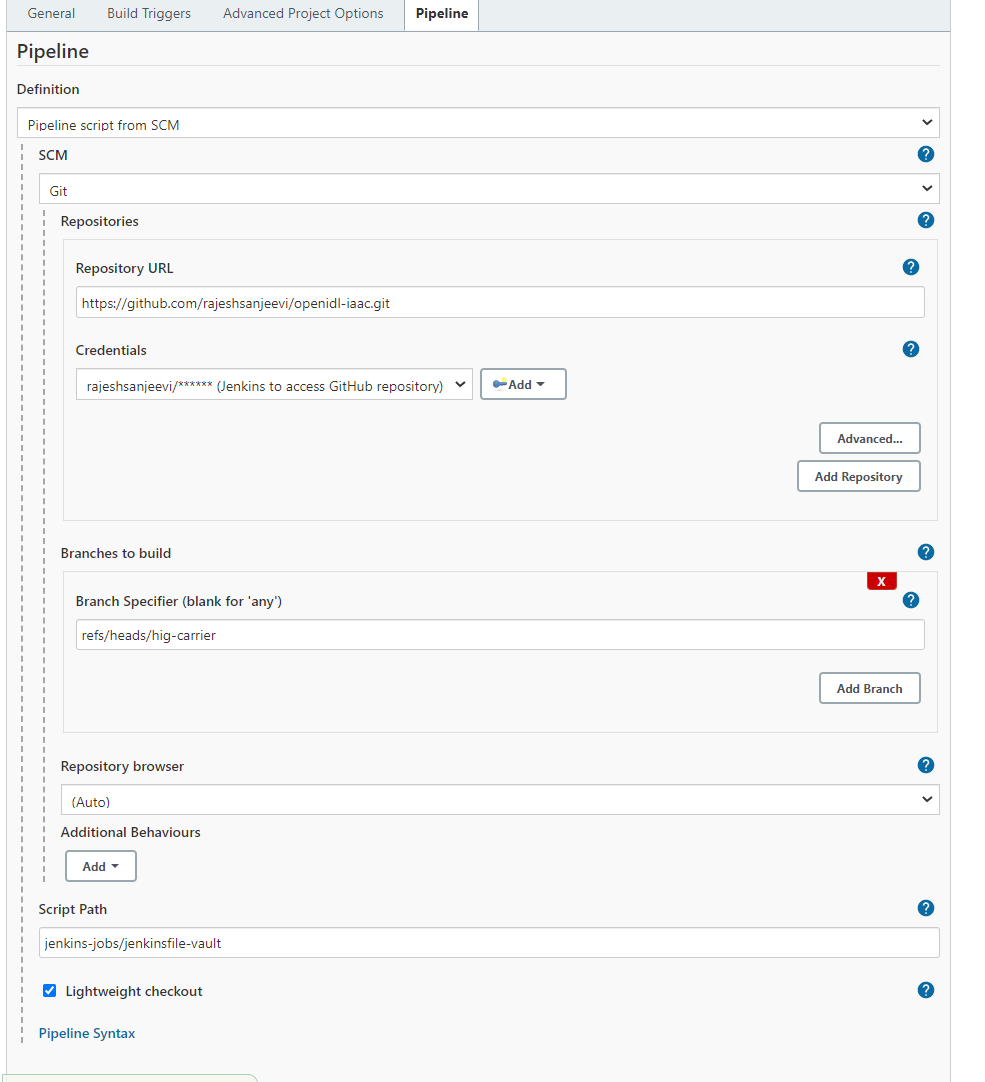
### Terraform Job

1. Go to Jenkins => New Item => Give a meaningful name
2. Select Job type as PIPELINE and proceed next
3. Give a description to the job and move to pipeline section
4. Select Definition as Pipeline Script from SCM
5. Select SCM as Git
6. Key in the Infrastructure code repository (openidl-aais-gitops) url.
7. Select the GitHub credentials
8. Specify the relevant branch “refs/heads/<branch-name>”.
9. Set script path to “Jenkins-jobs/jenkinsfile-tf”.



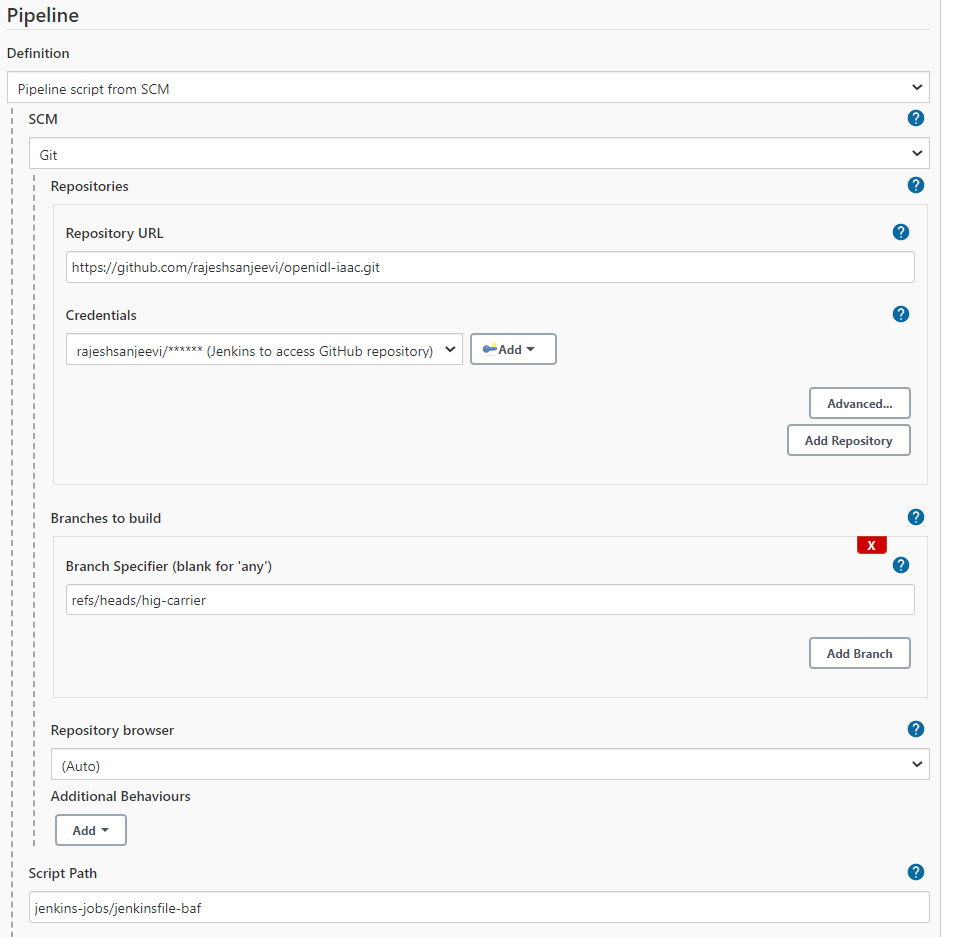
### Vault Job

1. Go to Jenkins => New Item => Give a meaningful name
2. Select Job type as PIPELINE and proceed next
3. Give a description to the job and move to pipeline section
4. Select Definition as Pipeline Script from SCM
5. Select SCM as Git
6. Key in the Infrastructure code repository (openidl-aais-gitops) url.
7. Select the GitHub credentials
8. Specify the relevant branch “refs/heads/<branch-name>”.
9. Set script path to “Jenkins-jobs/jenkinsfile-vault”.



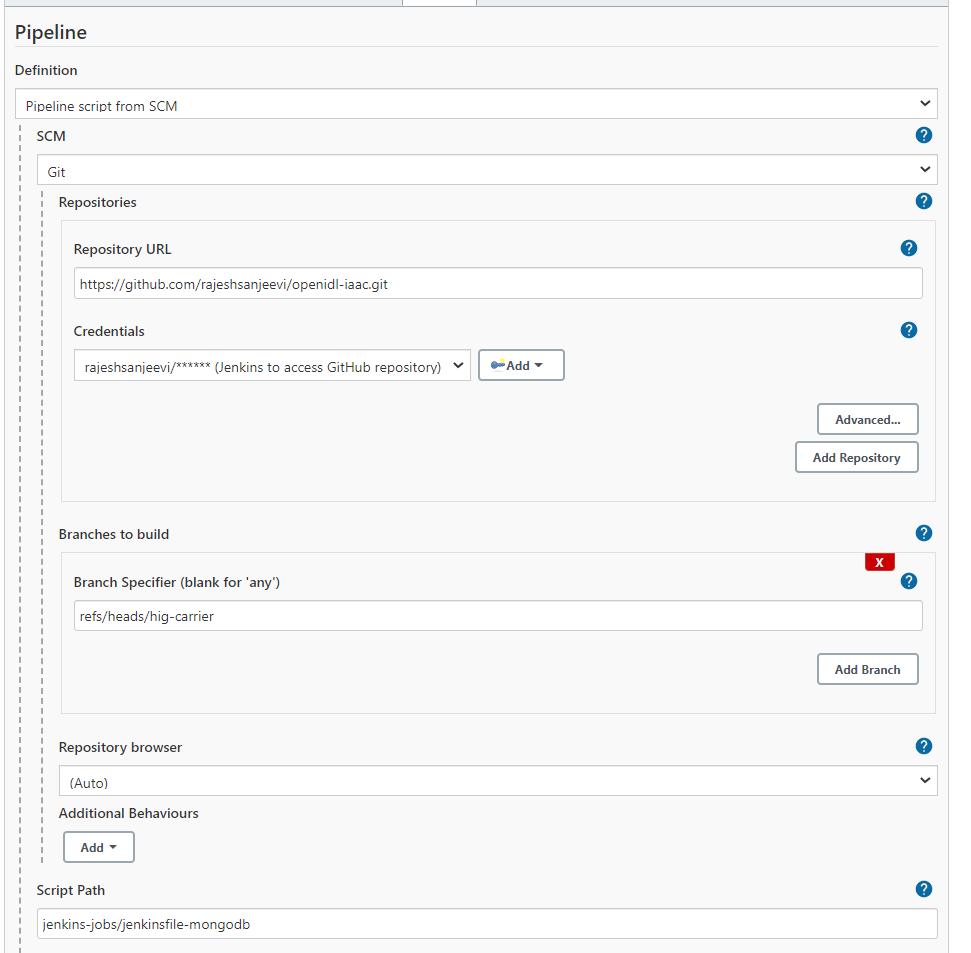
### Blockchain Network Job

1. Go to Jenkins => New Item => Give a meaningful name
2. Select Job type as PIPELINE and proceed next
3. Give a description to the job and move to pipeline section
4. Select Definition as Pipeline Script from SCM
5. Select SCM as Git
6. Key in the Infrastructure code repository (openidl-aais-gitops) url.
7. Select the GitHub credentials
8. Specify the relevant branch “refs/heads/<branch-name>”.
9. Set script path to “Jenkins-jobs/jenkinsfile-baf”.



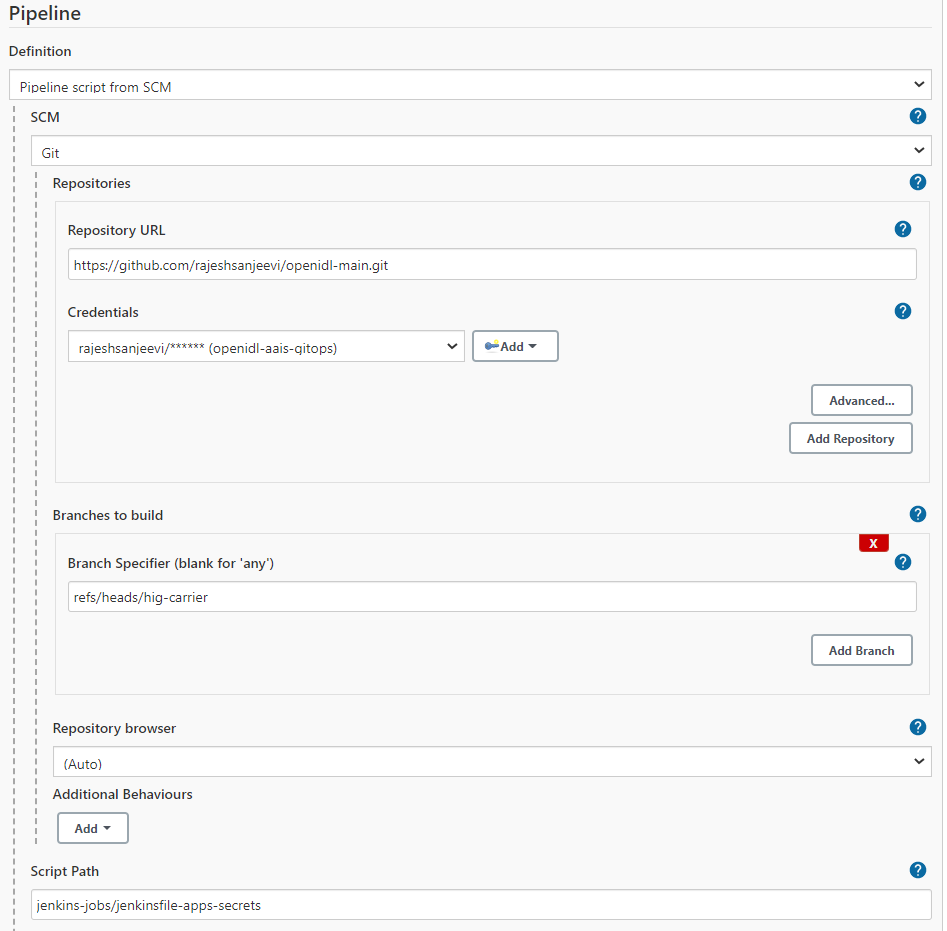
### MongoDB Job

1. Go to Jenkins => New Item => Give a meaningful name
2. Select Job type as PIPELINE and proceed next
3. Give a description to the job and move to pipeline section
4. Select Definition as Pipeline Script from SCM
5. Select SCM as Git
6. Key in the Infrastructure code repository (openidl-aais-gitops) url.
7. Select the GitHub credentials
8. Specify the relevant branch “refs/heads/<branch-name>”.
9. Set script path to “Jenkins-jobs/jenkinsfile-mongodb”.



### OpenIDL Application Job

1. Go to Jenkins => New Item => Give a meaningful name
2. Select Job type as PIPELINE and proceed next
3. Give a description to the job and move to pipeline section
4. Select Definition as Pipeline Script from SCM
5. Select SCM as Git
6. Key in the Infrastructure code repository (openidl-main) url.
7. Select the GitHub credentials
8. Specify the relevant branch “refs/heads/<branch-name>”.
9. Set script path to “Jenkins-jobs/jenkinsfile-apps-secrets”.



# Executing Jenkins Pipeline

### Terraform Job

This is the job previously setup. It is used to provision AWS resources and K8s resources. Before trigging the pipeline ensure the following are setup.

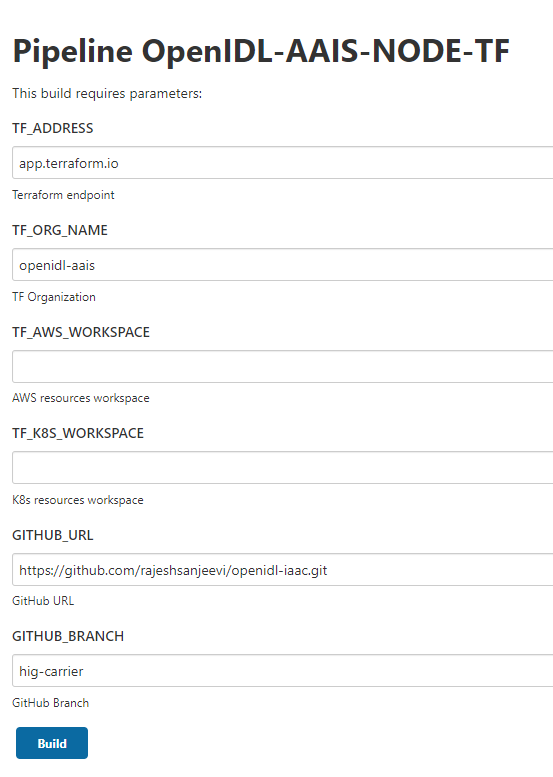
1. Terraform Cloud/Enterprise (Workspaces, VariableSet, API Token)
2. Jenkins (Credentials, Job configuration)
3. Terraform code changes and pushed to repository

**Note**: First run after configuring the job is dummy run as the option shows as “Build Now”. This will fail and will update your job with relevant parameters required for the job to run. Further runs will show an option Build with Parameters which will be right run.

1. To trigger the job, go to Jenkins => relevant job => Build with Parameters.
2. Enter the values to the inputs as listed below.

|  |  |
| --- | --- |
| **Field** | **Description** |
| TF\_ADDRESS | Terraform Cloud/Enterprise endpoint |
| TF\_ORG\_NAME | Organization name setup in Terraform |
| TF\_AWS\_WORKSPACE | Terraform workspace name setup specifically for AWS resources |
| TF\_K8S\_WORKSPACE | Terraform workspace name setup specifically for K8s resources |
| GITHUB\_URL | GitHub repository to check out the code |
| GITHUB\_BRANCH | GitHub branch specifically to check out the code |

1. The job runs terraform plan and asks manual confirmation before running terraform apply. This job will run first to provision AWS resources and further run for K8s resources. Hence twice it asks input to confirm before performing terraform apply.



**NOTE:**

It is noticed that sometimes the request to upload configuration data (git repository content) to Terraform fails with below HTTP error 422. In case when you see the pipeline failed with this error, rerun the pipeline which will help.

|  |  |  |
| --- | --- | --- |
| **Status** | **Response** | **Reason** |
| 422 | JSON API error object | Malformed request. |

### Preparing Config file for Infra Job

For the following pipelines the configuration file should be prepared and uploaded to the specific directory in the repository before triggering the pipeline.

1. Vault
2. MongoDB
3. Blockchain Network

The template and example configuration files are in the repository under “awx-automation/config-references”. Using these templates, the actual config file can be created and placed in the path “awx-automation/config”. The file name should follow the naming standard as below.

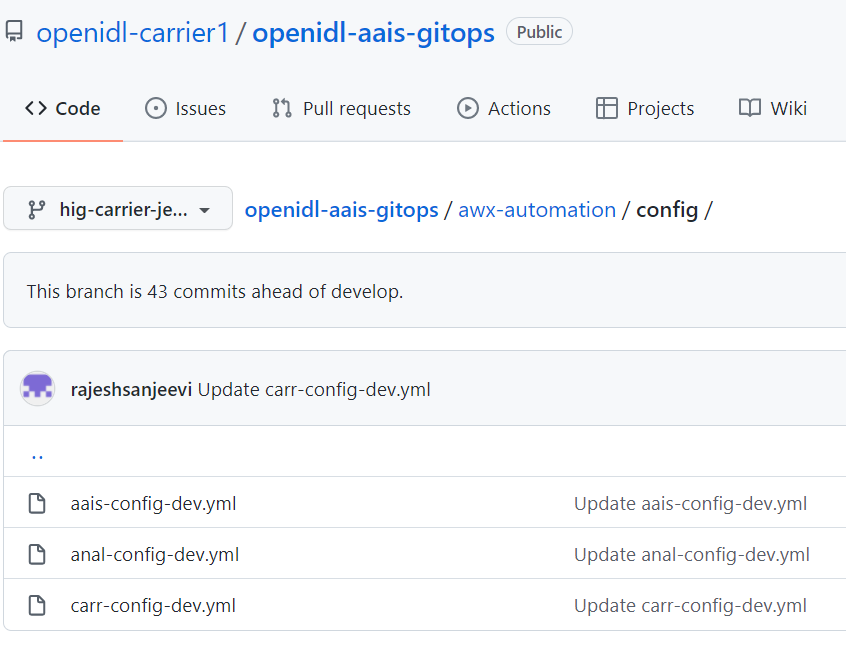
Name: <org-name>-config-<env>.yml

Org-name: First 4 characters of the org name

Env: dev | test | prod

The configuration file should be placed in the path “awx-automation/config/<org-name>-config-<env>.yml.

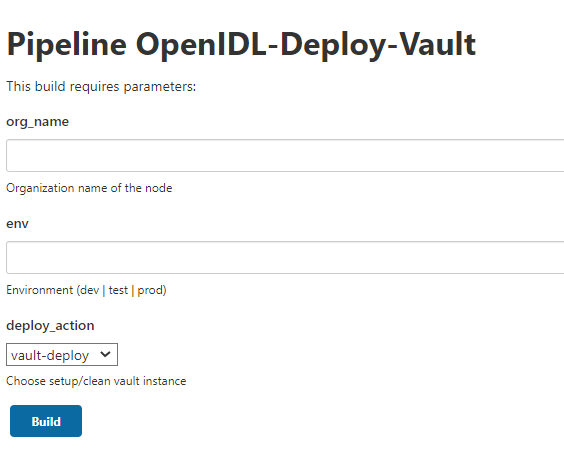
**NOTE:** The details in preparing the config file are to refer from base document.



### Vault Job

To run a vault job, go to specific Jenkins Job and click on Build with Parameters and key in organization name and environment type (dev | test |prod) and choose deploy\_action whether vault-deploy/vault-clean up based on the typical action to take.

**Note**: First run after configuring the job is dummy run as the option shows as “Build Now”. This will fail and will update your job with relevant parameters required for the job to run. Further runs will show an option Build with Parameters which will be right run.



### Blockchain Network Job

To run blockchain relevant tasks, go to the Job created for Blockchain Network and trigger relevant actions following the base document.

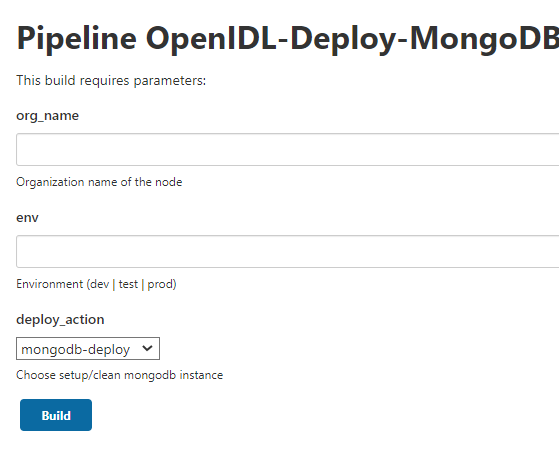
**Note**: First run after configuring the job is dummy run as the option shows as “Build Now”. This will fail and will update your job with relevant parameters required for the job to run. Further runs will show an option Build with Parameters which will be right run.



### MongoDB Job

To run a mongoDB job, go to specific Jenkins Job and click on Build with Parameters and key in organization name and environment type (dev | test |prod) and choose deploy\_action whether mongoDB-deploy/mongoDB-clean up based on the typical action to take.

**Note**: First run after configuring the job is dummy run as the option shows as “Build Now”. This will fail and will update your job with relevant parameters required for the job to run. Further runs will show an option Build with Parameters which will be right run.



### Preparing Config files for OpenIDL application job

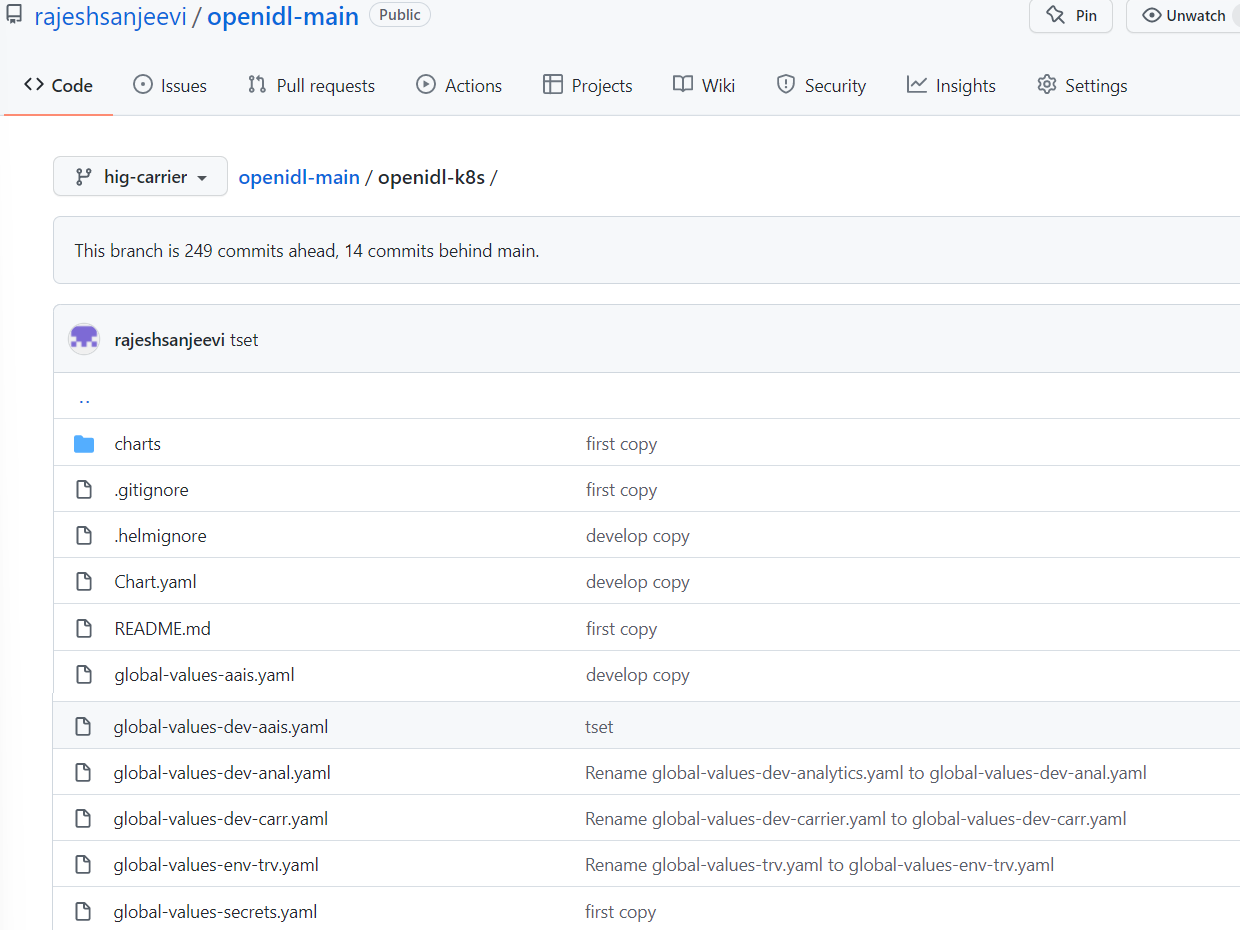
Before running application, specific jobs ensure that the following actions are completed referring to base document.

1. Credentials in AWX specific to application and secrets job templates
2. Referring to the base document getting the application related config json files to vault
3. Referring to the base document getting the application relate global-vaules-<org-name>.yml files under “openidl-main/openidl-k8s” directory in the repository.

**NOTE:** The global-values-\*\*\*.yml files should follow the naming standard as below.

**NAME:** global-vaules-<org-name>.yaml. The org-name should be 4-character representation only.

1. The details about how to prepare the file to be referred from base document.

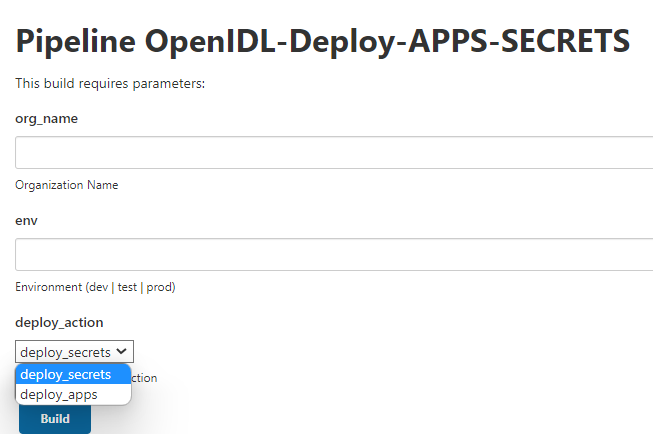


### OpenIDL Application Job

To deploy application secrets and OpenIDL application, run the job configured for OpenIDL applications. Go to Jenkins and select the relevant job and use Build with Parameters.

**Note**: First run after configuring the job is dummy run as the option shows as “Build Now”. This will fail and will update your job with relevant parameters required for the job to run. Further runs will show an option Build with Parameters which will be right run.

This job has two step process. First perform deploy-secrets and then deploy-apps action. The first action deploys relevant configuration as Kubernetes secrets and the next action deploys OpenIDL application containers.



|  |  |
| --- | --- |
|  |  |