



JENKINS CONTAINER DEPLOYMENT ON DOCKER

ABSTRACT

This guide provides basic instructions to deploy Jenkins with various plugins for Continuous Integration (CI) and Continuous Delivery (CD) workflows in a virtualized environment. It leverages the Docker container platform to support deployment in either a cloud or local information system environment.

Table of Contents

Docker Software Installation.....	2
Deploy a Jenkins Container.....	5
Advanced Jenkins Plugin Installation	15
Ansible Plugins	16
Terraform Plugins	16
NodeJS Plugin.....	16
Security Plugins.....	17
SonarQube Plugins.....	17
Aqua Plugins	18
Ansible Plugins	18
GitHub Plugins	19
Amazon Web Services Plugins.....	20
Pipelines Plugins	22
Google and GCP Plugins	23
Kubernetes Plugins	24
Jenkins Advanced Software Installation	25
Modified Jenkins Container Image Creation	26
Jenkins Service Credentials Setup	28
Jenkins AWS IAM Credential.....	28
Add Credentials to Jenkins	29
Create a Jenkins Pipeline.....	30
Setup Jenkins File.....	32
Repository Revision.....	34
Create a Jenkins Pipeline with Terraform Choices	34
Pipeline Creation.....	35
Run Jenkins Pipeline	37
Terraform Destroy Deployed Project.....	43

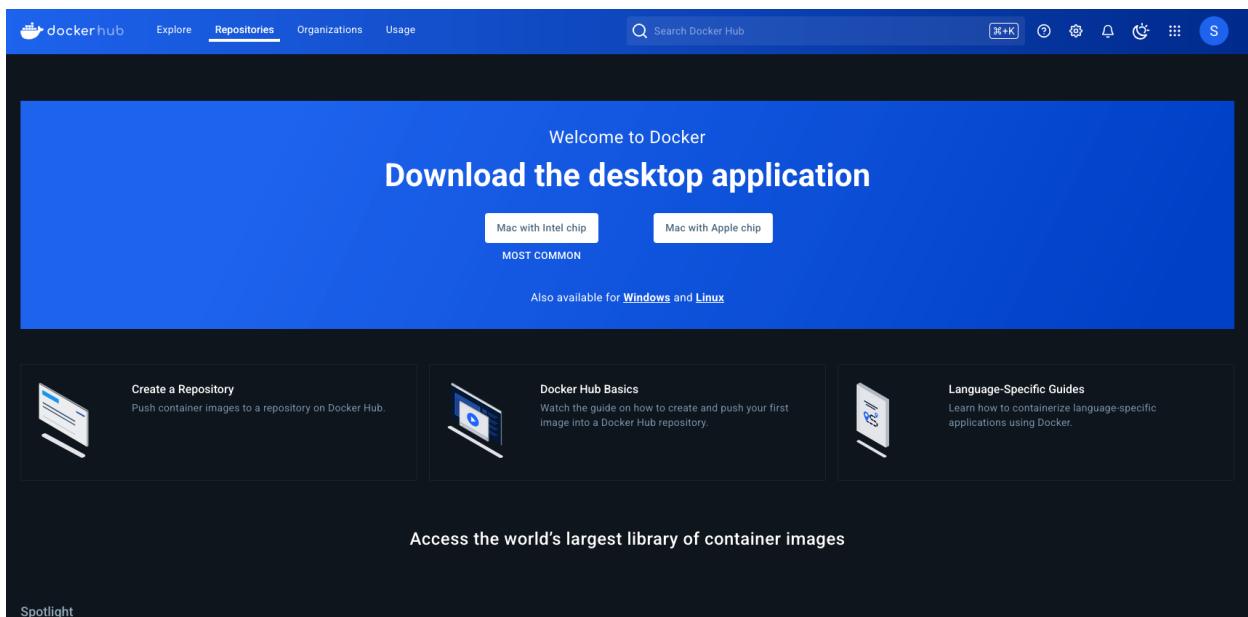
Docker Software Installation

Pre-requisites:

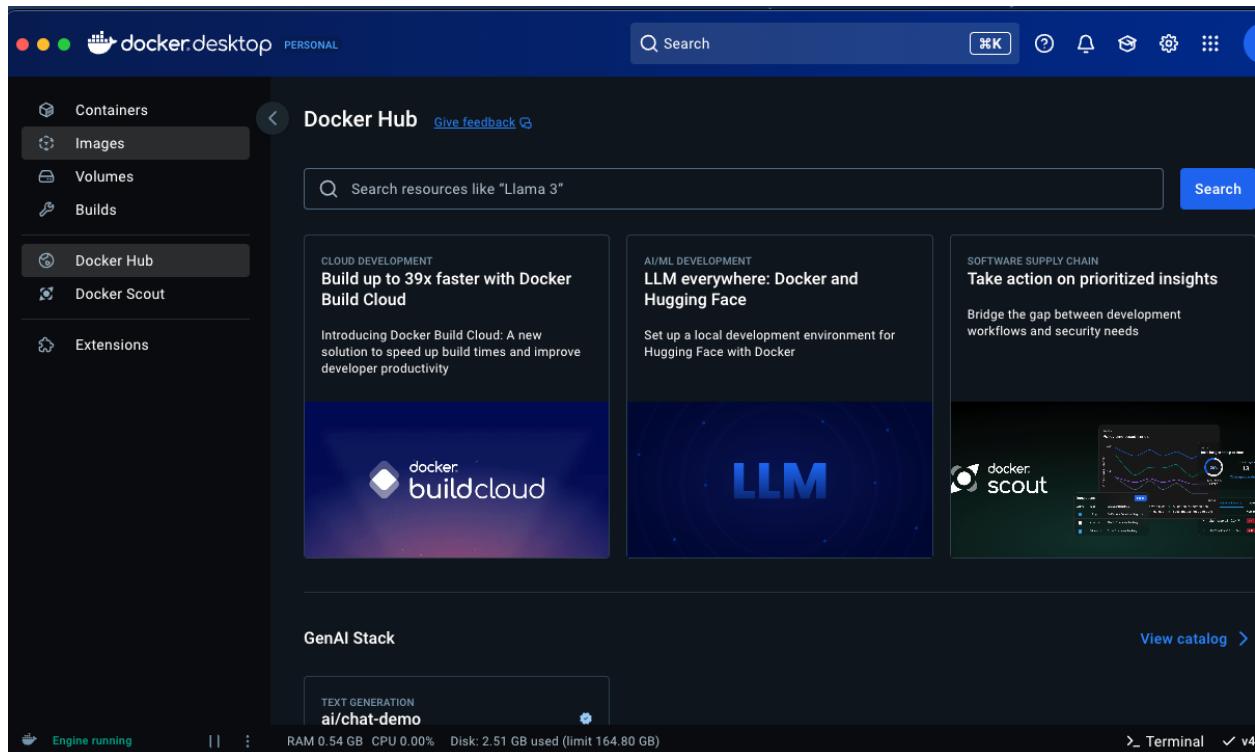
- Docker and Docker-compose installed via command-line using your operating system package installation software (e.g. Homebrew <https://brew.sh/> for MacOS/Linux or Chocolatey <https://chocolatey.org/> for Windows)
- Docker Desktop <https://www.docker.com/products/docker-desktop/> to provide a graphical user interface. This is especially a good application to use if you are new to containers and Docker.
- A docker account to interact with docker services like Docker Hub, <https://app.docker.com/login>.

Note: To get familiar with Docker and running containers on docker Reference: <https://www.docker.com/get-started/>

1. Create an account on Docker Hub. <https://hub.docker.com/>



2. Once you have created an account on Docker Hub, download the docker desktop application for your operating system. Installing the Docker desktop application will include docker-compose as well. You will use Docker Hub to upload and download container images you create over time as you work with docker and containers.
3. Verify docker desktop is installed. After installing the docker desktop application, start the docker desktop application and login to your docker hub account using the docker application.



- Verify docker command line works. **Note:** This is being run on MacOS. You should see similar output on a Linux or Windows based operating system.

```
$ docker version
```

```
Client:  
  Version:          27.5.1  
  API version:     1.47  
  Go version:      go1.22.11  
  Git commit:       9f9e405  
  Built:            Wed Jan 22 13:37:19 2025  
  OS/Arch:          darwin/arm64  
  Context:          desktop-linux  
  
Server: Docker Desktop 4.38.0 (181591)  
Engine:  
  Version:          27.5.1  
  API version:     1.47 (minimum version 1.24)  
  Go version:      go1.22.11  
  Git commit:       4c9b3b0  
  Built:            Wed Jan 22 13:41:25 2025  
  OS/Arch:          linux/arm64  
  Experimental:    false  
containerd:  
  Version:          1.7.25  
  GitCommit:        bcc810d6b9066471b0b6fa75f557a15a1cbf31bb  
runc:  
  Version:          1.1.12  
  GitCommit:        v1.1.12-0-g51d5e946  
docker-init:  
  Version:          0.19.0  
  GitCommit:        de40ad0
```

Deploy a Jenkins Container

1. Search for the official Jenkins container image on Docker Hub using either Docker Desktop application or docker command line.

The screenshot shows the Docker Desktop interface. On the left, there's a sidebar with options like Containers, Images, Volumes, Builds, Docker Hub (which is selected), Docker Scout, and Extensions. The main area is titled 'Docker Hub / Search' and has a search bar with 'jenkins'. Below it, it says 'Search results for "jenkins"' and 'Give feedback'. A search bar contains 'jenkins'. It shows 1 - 24 of 2500 results. There are six cards for different Jenkins-related images:

- jenkins/jenkins**: The leading open source automation server. 1B+ stars, 4.1K reviews.
- jenkins**: DEPRECATED; use "jenkins/jenkins:lts" instead. 100M+ stars, 5.7K reviews.
- jenkins/inbound-agent**: This is an image for Jenkins agents using TCP or WebSockets to establish inbound connection to the J. 100M+ stars, 140 reviews.
- jenkins/jnlp-agent-maven**: A JNLP-based agent with Maven 3 built in. 100M+ stars, 4.1K reviews.
- jenkins/jnlp-slave**: a Jenkins agent which can connect to Jenkins using JNLP4 or Websocket protocols. 100M+ stars, 5.7K reviews.
- jenkins/ssh-agent**: Docker image for Jenkins agents connected over SSH. 100M+ stars, 140 reviews.

At the bottom, it shows 'Engine running', resource usage (RAM 1.41 GB, CPU 0.00%, Disk: 3.26 GB used (limit 164.80 GB)), and a 'Terminal' button.

// Docker Command Line Example.

Note: You will see a long list of Jenkins images, but you will use the one named `jenkins/jenkins` or `jenkins/jenkins:lts`. The "lts" is the long-term support version.

```
$ docker search Jenkins
```

NAME	DESCRIPTION	STARS	OFFICIAL
<code>jenkins/jenkins</code>	The leading open source automation server	4085	
<code>jenkins</code>	DEPRECATED; use "jenkins/jenkins:lts" instead	5696	[OK]
<code>jenkins/inbound-agent</code>	This is an image for Jenkins agents using TC...	140	
<code>jenkins/jnlp-agent-maven</code>	A JNLP-based agent with Maven 3 built in	10	
<code>jenkins/jnlp-slave</code>	a Jenkins agent which can connect to Jenkins...	157	
<code>jenkins/ssh-agent</code>	Docker image for Jenkins agents connected ov...	60	
<code>jenkins/slave</code>	base image for a Jenkins Agent, which includ...	50	
<code>jenkins/agent</code>	This is a base image, which provides the Jen...	74	
<code>jenkins/ssh-slave</code>	A Jenkins slave using SSH to establish conne...	41	
<code>jenkins/jnlp-agent-ruby</code>		1	
<code>jenkins/jnlp-agent-docker</code>		11	
<code>jenkins/jnlp-agent-python</code>	A JNLP-based agent with Python built in	4	
<code>jenkins/jnlp-agent-node</code>		1	

2. In docker desktop select pull and the image will be downloaded to your information system. Once downloaded you should see it listed in the Docker Desktop application images inventory.

Note: you can do the same from the command line:

```
$ docker pull jenkins/jenkins
Using default tag: latest
latest: Pulling from jenkins/jenkins
fa8785a0cbfc: Download complete
70ae892edd2c: Download complete
e228c44fff3b: Download complete
e474a4a4cbbf: Download complete
40375a76d092: Download complete
154b66b7ce12: Download complete
eeeada77dafa: Download complete
764eeac48f0c: Download complete
25617dd6d7e8: Download complete
03bb4613691f: Download complete
793748ac1b11: Download complete
8c3f2fecd07d: Download complete
Digest: sha256:119e9decb712ec14aace4343ac4203ef2ba4d6cb7dbcb5387c76b85b29dd15fd
Status: Downloaded newer image for jenkins/jenkins:latest
docker.io/jenkins/jenkins:latest
```

3. Log into the Docker Hub via the command line. You should see a similar result below. You can use either GitBash (on windows) or the terminal (on MacOS or Linux).

// Docker login via command line example

```
$ docker login Authenticating with existing credentials... Login Succeeded
```

- Once you have successfully logged into docker, run the following command:
docker run -p 8080:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home
jenkins/jenkins:lts-jdk11

Note: This is telling docker to perform the following tasks (in no specific order):

- Create a container running on the information system
- Map the computer that docker is running on to ports 8080 and 50000.
- Create a folder named <jenkins_home> and map it to the containers folder </var/jenkins_home>.
- Use the source image named jenkins:lts-jdk11

// Docker run example

```
$ docker run -p 8080:8080 -p 50000:50000 -v jenkins_home:/var/jenkins_home jenkins/jenkins:lts-jdk11
Unable to find image 'jenkins/jenkins:lts-jdk11' locally
lts-jdk11: Pulling from jenkins/jenkins
8bbe33c1a45d: Download complete
f62c370bc989: Download complete
75d89130f4e0: Download complete
4760cf60c6e6: Download complete
6a4c33872faf: Download complete
5e0034fcdf5: Download complete |
39252305a35d: Download complete
ec694b08d3fc: Download complete
6d11c181ebb3: Download complete
4a9739527a50: Download complete
a9af1892fb19: Download complete
1b4ed656dd0a: Download complete
Digest: sha256:6aa6c6bd7da914bf5333305c8102cb26965ea4b227e37f4269315725a2b0cd81
Status: Downloaded newer image for jenkins/jenkins:lts-jdk11
Running from: /usr/share/jenkins/jenkins.war
webroot: /var/jenkins_home/war
```

- Locate the password created during the Jenkins container's initial startup. The password location will be in /var/jenkins_home/secrets/InitialAdminPassword

6. Access Jenkins via a web browser at the following URL: <http://localhost:8080>.

Note: You will need the password you should have seen in the command line to login the Jenkins web portal.

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

Continue

7. Select the recommended Jenkins plugins. This can take some time to finish. You will install additional plugins after the initial setup of Jenkins.

Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.462.3

Getting Started

Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✗ Credentials Binding	** Ionicons API Folders OWASP Markup Formatter ** ASM API ** JSON Path API ** Structs ** Pipeline: Step API ** Token Macro Build Timeout
✗ Timestamper	✗ Workspace Cleanup	✗ Ant	✗ Gradle	
✗ Pipeline	✗ GitHub Branch Source	✗ Pipeline: GitHub Groovy Libraries	✗ Pipeline Graph View	
✗ Git	✗ SSH Build Agents	✗ Matrix Authorization Strategy	✗ PAM Authentication	
✗ LDAP	✗ Email Extension	✗ Mailer	✗ Dark Theme	

** - required dependency

Jenkins 2.462.3

8. Create a new user account. Document the username and password you created during this step.

Getting Started

Create First Admin User

Username

Password

Confirm password

Full name

E-mail address

Jenkins 2.462.3

[Skip and continue as admin](#)

[Save and Continue](#)

9. Set URL that you want jenkins to listen on. In this example that is <http://localhost:8080>.

Getting Started

Instance Configuration

Jenkins URL:

<http://localhost:8080/>

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD_URL environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.462.3

[Not now](#)

[Save and Finish](#)

10. If successful you should see the message "Jenkins is ready!" Select the "start using jenkins" button.

Getting Started

Jenkins is ready!

Your Jenkins setup is complete.

[Start using Jenkins](#)

Jenkins 2.462.3

11. You should see a page stating, "Welcome to Jenkins!".

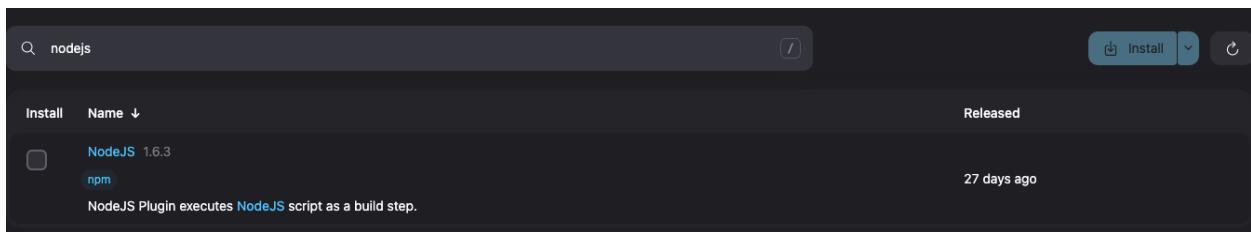
The screenshot shows the Jenkins dashboard. At the top, there is a navigation bar with the Jenkins logo, a search bar containing 'Search (⌘+K)', and user information for 'demo administrator'. Below the navigation bar, the main content area has a title 'Welcome to Jenkins!'. A sub-header below it says 'This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.' There are several buttons and links: '+ New Item', 'Build History', 'Manage Jenkins', 'My Views', 'Create a job' (with a '+' icon), 'Set up a distributed build' (with a computer monitor icon), 'Set up an agent' (with a monitor icon), 'Configure a cloud' (with a cloud icon), and 'Learn more about distributed builds' (with a help icon). On the left, there are two collapsed sections: 'Build Queue' (showing 'No builds in the queue.') and 'Build Executor Status' (showing '1 Idle' and '2 Idle'). At the bottom right, there are links for 'REST API' and 'Jenkins 2.462.3'.

Advanced Jenkins Plugin Installation

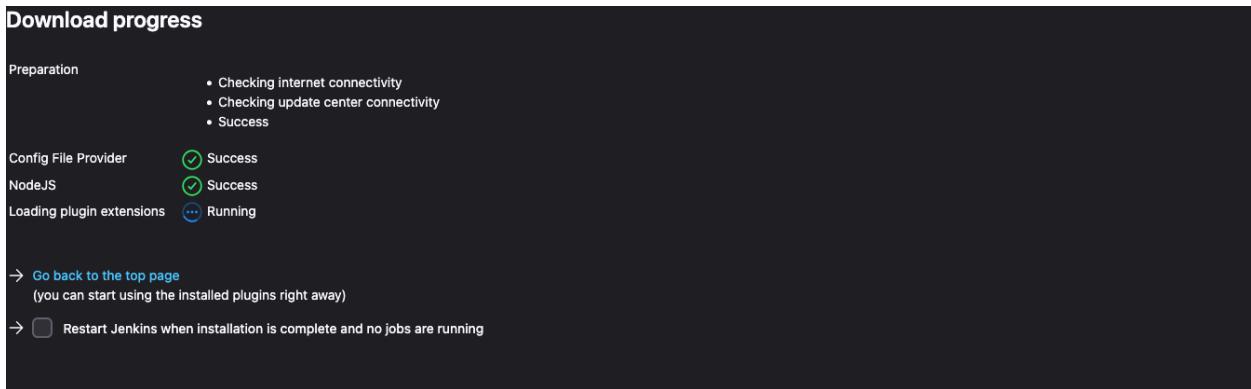
Install additional plugins in the Jenkins software.

1. From the Dashboard, Clouds, install a plugin -- On the left-side of the webpage select available
2. In search window type the following keywords: AWS, GitHub, Google, GCP, SNYK, security, SonarQube, Terraform, Ansible, Docker, and Pipeline.

After you input each search keyword, you should see a list of available plugins, for example input of keyword NodeJS, should show at least one available plugin (if it has not already been installed)



For each plugin you select the “install checkbox” to the left of the plugin you want installed in Jenkins and select the “Install” button on the upper right of the webpage.

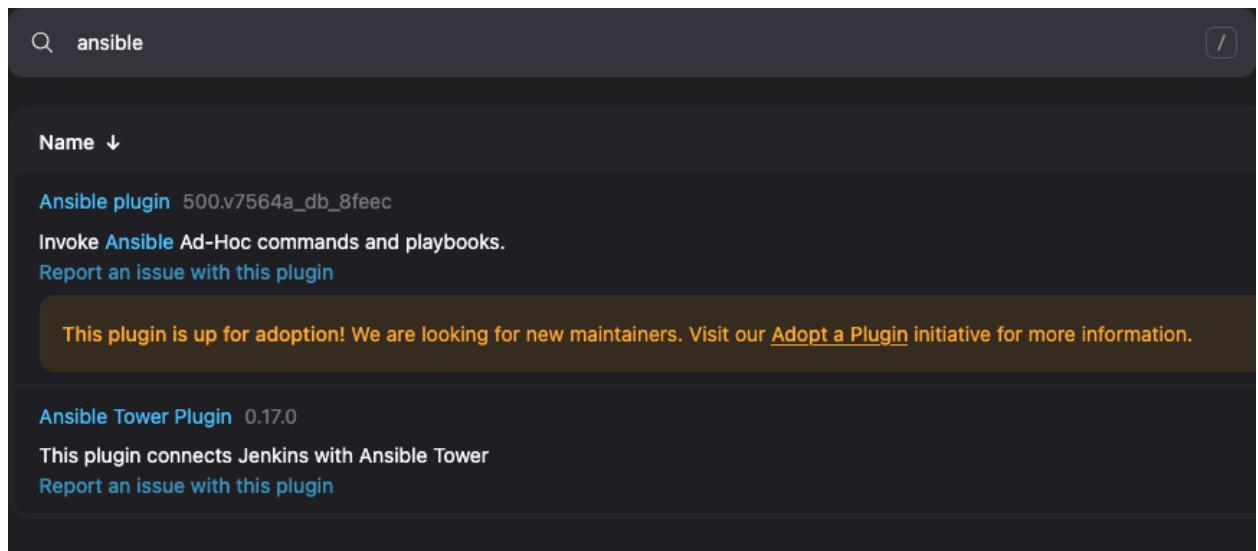


Specific plugins to select for the basic Jenkins will include plugins and associated software to support various CI/CD workflows including security scanners and various opensource and commercial software used in development, security and operational processes.

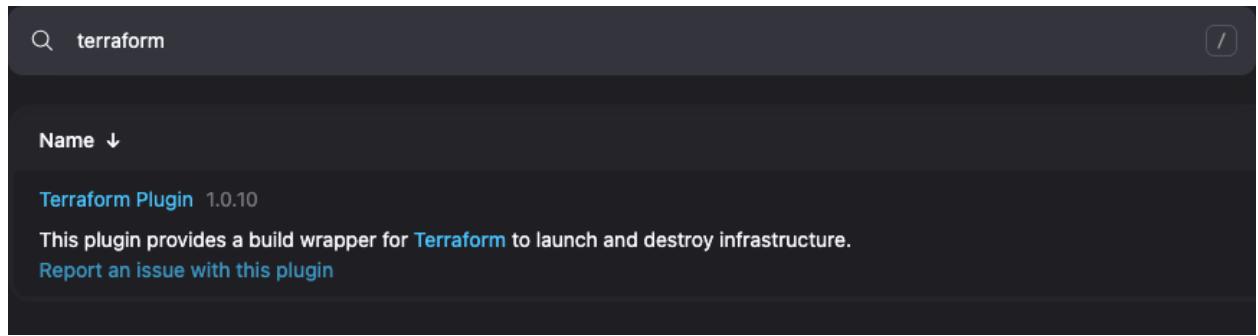
Not all use of each plugin or associated software will be demonstrated in this document as its purpose is to guide you to do a setup of the software to use from basic to advanced projects.

The minimum plugins are presented in screenshot format. Once you have selected and successfully installed these plugins your installation should look like the screenshot's below:

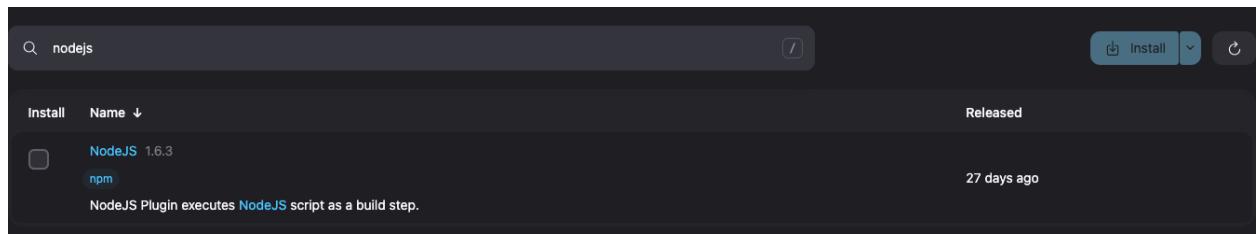
Ansible Plugins



Terraform Plugins



NodeJS Plugin



Security Plugins

The screenshot shows a list of Jenkins security plugins. The search bar at the top contains the text "security". The results are as follows:

- Aqua Security Scanner 3.2.7**
This plugin enables scanning of Docker images using the Aqua API.
[Report an issue with this plugin](#)
- Aqua Security Serverless Scanner 1.0.6**
This plugin enables scanning of serverless functions using the Aqua API.
[Report an issue with this plugin](#)
- Email Extension Plugin 1876.v28d8d38315b_d**
This plugin is a replacement for Jenkins's email publisher. It allows to configure every aspect of email notifications: when an email is sent, who should receive it and what the email says
[Report an issue with this plugin](#)
- Matrix Authorization Strategy Plugin 3.2.4**
Offers matrix-based security authorization strategies (global and per-project).
[Report an issue with this plugin](#)
- Pipeline: Groovy 4018.vf02e01888da_f**
Pipeline execution engine based on continuation passing style transformation of Groovy scripts.
[Report an issue with this plugin](#)
- Pipeline: Nodes and Processes 1405.v1fcd4a_d00096**
Pipeline steps locking agents and workspaces, and running external processes that may survive a Jenkins restart or agent reconnection.
[Report an issue with this plugin](#)
- Script Security Plugin 1373.vb_b_4a_a_c26fa_00**
Allows Jenkins administrators to control what in-process scripts can be run by less-privileged users.
[Report an issue with this plugin](#)
- Snyk Security Plugin 4.1.0**
Add the ability to test your code dependencies for vulnerabilities against Snyk database
[Report an issue with this plugin](#)

SonarQube Plugins

The screenshot shows a list of Jenkins SonarQube plugins. The search bar at the top contains the text "sonar". The results are as follows:

- SonarQube Scanner for Jenkins 2.18**
This plugin allows an easy integration of [SonarQube](#), the open source platform for Continuous Inspection of code quality.
[Report an issue with this plugin](#)

Aqua Plugins

The screenshot shows the Jenkins plugin manager interface with a search bar at the top containing the text "aqua". Below the search bar, there is a sorting header "Name ↓". Three plugin entries are listed:

- Aqua MicroScanner 1.0.8**
Enables scanning of docker build for OS package vulnerabilities.
[Report an issue with this plugin](#)
- Aqua Security Scanner 3.2.7**
This plugin enables scanning of Docker images using the Aqua API.
[Report an issue with this plugin](#)
- Aqua Security Serverless Scanner 1.0.6**
This plugin enables scanning of serverless functions using the Aqua API.
[Report an issue with this plugin](#)

Ansible Plugins

The screenshot shows the Jenkins plugin manager interface with a search bar at the top containing the text "ansible". Below the search bar, there is a sorting header "Name ↓". Two plugin entries are listed:

- Ansible plugin 500.v7564a_db_8feec**
Invoke Ansible Ad-Hoc commands and playbooks.
[Report an issue with this plugin](#)

This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information.
- Ansible Tower Plugin 0.17.0**
This plugin connects Jenkins with Ansible Tower
[Report an issue with this plugin](#)

GitHub Plugins

The screenshot shows a Jenkins search interface with the query "github" entered. Below the search bar, there is a dropdown menu labeled "Name" with an arrow pointing down. The main content area displays a list of Jenkins plugins related to GitHub:

- Blue Ocean** 1.27.16
BlueOcean Aggregator
[Report an issue with this plugin](#)
- GitHub API Plugin** 1.321-478.vc9ce627ce001
This plugin provides GitHub API for other plugins.
[Report an issue with this plugin](#)
- GitHub Authentication plugin** 621.v33b_4394dda_4d
Authentication plugin using GitHub OAuth to provide authentication and authorization capabilities for GitHub and GitHub Enterprise.
[Report an issue with this plugin](#)
- GitHub Branch Source Plugin** 1810.v913311241fa_9
Multibranch projects and organization folders from GitHub. Maintained by CloudBees, Inc.
[Report an issue with this plugin](#)
- GitHub Integration Plugin** 0.7.2
GitHub Integration Plugin for Jenkins
- GitHub Pipeline for Blue Ocean** 1.27.16
BlueOcean GitHub organization pipeline creator
[Report an issue with this plugin](#)
- GitHub plugin** 1.41.0
This plugin integrates GitHub to Jenkins.
[Report an issue with this plugin](#)
- Pipeline GitHub Notify Step Plugin** 49.vf37bf92d2bc8
Plugin that provides a GitHub status notification step
[Report an issue with this plugin](#)
- Pipeline implementation for Blue Ocean** 1.27.16
This plugin is a part of BlueOcean Plugin
[Report an issue with this plugin](#)
- Pipeline: GitHub** 2.8-159.09e4403bc62f
Allows programmatic access to GitHub via new global variables in pipeline builds.
[Report an issue with this plugin](#)
- Pipeline: GitHub Groovy Libraries** 61.v629f2cc41d83
Allows Pipeline Groovy libraries to be loaded on the fly from GitHub.
[Report an issue with this plugin](#)

Amazon Web Services Plugins

For AWS there will be a significant number of plugins that will end up being installed either by the initial plugin steps and as you install other plugins. The list of plugins is provided, but a more detailed screenshot is also provided to help ensure you have all the plugins you may need for your projects.

- AWS Plugins:
 - AWS Credentials Pipeline
 - AWS Steps
 - Amazon EC2
 - Amazon Elastic Container Service (ECS)/Fargate
 - AWS CodeDeploy
 - AWS Lambda
 - Amazon S3 Bucket Credentials
 - AWS Codebuild
 - AWS CodePipeline
 - AWS Secrets Manager Secret
 - Source Configuration as Code AWS SSM secrets
 - CloudFormation
 - AWS SAM

Q|amAWS

[Amazon EC2 plugin](#) 1856.vf40220e7a_75f
This plugin integrates Jenkins with [Amazon EC2](#) or anything implementing the EC2 API's such as an Ubuntu.
[Report an issue with this plugin](#)

[Amazon ECR plugin](#) 1.151.vb_ca_71ddd0b_cf
This plugin generates Docker authentication token from Amazon Credentials to access Amazon ECR.
[Report an issue with this plugin](#)

This plugin is up for adoption! We are looking for new maintainers. Visit our [Adopt a Plugin](#) initiative for more information.

[Amazon Elastic Container Service \(ECS\) / Fargate plugin](#) 1.49
Use [Amazon EC2 Container Service](#) to provide elastic agents.
[Report an issue with this plugin](#)

[Amazon S3 Bucket Credentials Plugin](#) 1.0.0
[Report an issue with this plugin](#)

[Amazon Web Services SDK 2 :: CloudFormation](#) 2.30.10-22.v1339d66b_e0f4
CloudFormation module for the [AWS SDK for Java](#).
[Report an issue with this plugin](#)

[Amazon Web Services SDK 2 :: CodeDeploy](#) 2.30.10-22.v1339d66b_e0f4
CodeDeploy module for the [AWS SDK for Java](#).
[Report an issue with this plugin](#)

[Amazon Web Services SDK 2 :: Core](#) 2.30.10-22.v1339d66b_e0f4
Core modules for the [AWS SDK for Java](#).
[Report an issue with this plugin](#)

[Amazon Web Services SDK 2 :: EC2](#) 2.30.10-22.v1339d66b_e0f4
EC2 module for the [AWS SDK for Java](#).
[Report an issue with this plugin](#)

[AWS CodeDeploy Plugin for Jenkins](#) 1.23
This plugin provides a "post-build" step for AWS CodeDeploy.
[Report an issue with this plugin](#)

[AWS CodePipeline Plugin](#) 0.49
[AWS CodePipeline Integration](#)
[Report an issue with this plugin](#)

[AWS Credentials Plugin](#) 243.v41c19a_fb_5dcf
Allows storing Amazon IAM credentials within the Jenkins Credentials API. Store Amazon IAM access keys (`AWSSecretKey` and `AWSSecretKey`) within the Jenkins Credentials API. Also support IAM Roles and IAM MFA Token.
[Report an issue with this plugin](#)

[AWS Global Configuration Plugin](#) 144.vfb_4447b_7c93e
A Jenkins plugin to configure AWS related settings
[Report an issue with this plugin](#)

[AWS Lambda Cloud plugin](#) 0.4
AWS Lambda Cloud Plugin: Dynamic Agents Running on AWS Lambda
[Report an issue with this plugin](#)

[AWS Lambda Plugin](#) 0.5.10
This Plugin allows you to upload a zip file or folder to AWS Lambda
[Report an issue with this plugin](#)

[AWS SAM](#) 1.2.13
Simplify automated deployments of serverless applications by using AWS SAM Templates.
[Report an issue with this plugin](#)

[AWS Secrets Manager Credentials Provider](#) 1.214.va_0a_d8268d068
Source Jenkins Credentials from AWS Secrets Manager.
[Report an issue with this plugin](#)

[AWS Secrets Manager SecretSource](#) 1.72.v61781b_35c542
AWS Secrets Manager backend for the Jenkins SecretSource API
[Report an issue with this plugin](#)

[CloudFormation plugin](#) 231.v3b_84a_7a_074ec
Adds a build wrapper that can spawn an AWS Cloud Formation recipe at the start of the build and take it down at the end.
[Report an issue with this plugin](#)

[Configuration as Code AWS SSM secrets](#) 1.0.1
This plugin resolves secrets from Amazon SSM for CASC plugin
[Report an issue with this plugin](#)

[Pipeline: AWS Steps](#) 1.45
This plugin adds Jenkins pipeline steps to interact with the AWS API.
[Report an issue with this plugin](#)

Pipelines Plugins

Note: Various plugins with the keyword pipeline will end up in your list of available and installed plugins as you work through selecting plugins for your jenkins installation. You should have the following plugins selected for installation when using the pipeline keyword.

- Kubernetes plugin
- Pipeline
- Pipeline GitHub Notify Step Plugin
- Pipeline Utility Steps
- Pipeline: AWS Steps
- Pipeline: Build Step
- Pipeline: GitHub
- Pipeline: Groovy
- Pipeline: MultiSteps

Kubernetes plugin 4306.vc91e951ea_eb_d
This plugin integrates Jenkins with Kubernetes
[Report an issue with this plugin](#)

Lockable Resources plugin 1349.v8b_ccb_c5487f7
This plugin allows to define external resources (such as printers, phones, computers) that can be locked by builds. If a build requires an external resource which is already locked, it will wait for the resource to be free.
[Report an issue with this plugin](#)

Pipeline 600.vb_57cd26fdd7
A suite of plugins that lets you orchestrate automation, simple or complex. See [Pipeline as Code](#) with Jenkins for more details.
[Report an issue with this plugin](#)

Pipeline GitHub Notify Step Plugin 49.vf37bf92d2bc8
Plugin that provides a GitHub status notification step
[Report an issue with this plugin](#)

Pipeline Graph Analysis Plugin 216.vfd8b_ece330ca_...
Provides a REST API to access pipeline and pipeline run data.
[Report an issue with this plugin](#)

Pipeline Graph View Plugin 409.v98f212e980b_4
Provides "Pipeline Graph" visualization of a pipeline job run.
[Report an issue with this plugin](#)

Pipeline: GCP Steps 22.vcc10eff7c13f
This plugin adds Jenkins pipeline steps to interact with the GCP API.
[Report an issue with this plugin](#)

Pipeline: GitHub 2.8-159.09e4403bc62f
Allows programmatic access to GitHub via new global variables in pipeline builds.
[Report an issue with this plugin](#)

Pipeline: GitHub Groovy Libraries 61.v629f2cc41d83
Allows Pipeline Groovy libraries to be loaded on the fly from GitHub.
[Report an issue with this plugin](#)

Pipeline Utility Steps 2.18.0
Utility steps for pipeline jobs.
Report an issue with this plugin
Pipeline: API 1363.v03f731255494
Plugin that defines Pipeline API.
Report an issue with this plugin
Pipeline: AWS Steps 1.45
This plugin adds Jenkins pipeline steps to interact with the AWS API.
Report an issue with this plugin
Pipeline: Basic Steps 1079.vce64b_a_929c5a_
Commonly used steps for Pipelines.
Report an issue with this plugin
Pipeline: Build Step 555.v589d5c24a_3d6
Adds the Pipeline step <code>build</code> to trigger builds of other jobs.
Report an issue with this plugin
Pipeline: Declarative 2.2221.vc657003fb_d93
An opinionated, declarative Pipeline.
Report an issue with this plugin
Pipeline: Declarative Extension Points API 2.2221.vc657003fb_d93
APIs for extension points used in Declarative Pipelines.
Report an issue with this plugin
Pipeline: GCP Steps 22.vcc10eff7c13f
This plugin adds Jenkins pipeline steps to interact with the GCP API.
Report an issue with this plugin
Pipeline: GitHub 2.8-159.09e4403bc62f
Allows programmatic access to GitHub via new global variables in pipeline builds.
Report an issue with this plugin

Google and GCP Plugins

For the Google Cloud Platform we will use two keywords, “Google” and “GCP” to obtain our plugin list.

<input type="text"/> google	<input type="button"/>
Name ↓	
Google Cloud Platform SDK :: Auth 26.23.0-28.vd30f921a_22a_8	
This plugin provides all Google Cloud Auth SDK for Java modules not packaged as standalone plugins.	
Report an issue with this plugin	
Google Cloud Platform SDK :: Storage 26.23.0-28.vd30f921a_22a_8	
This plugin provides all Google Cloud Storage SDK for Java modules not packaged as standalone plugins.	
Report an issue with this plugin	
Google Cloud Storage plugin 1.360.v6ca_38618b_41f	
This plugin provides the “Google Cloud Storage Uploader” post-build step for publishing build artifacts to Google Cloud Storage.	
Report an issue with this plugin	
Google Metadata plugin 0.5	
Provides a basic framework for steps in a build’s lifecycle to attach JSON-serializable metadata to a build (as an invisible action).	
Report an issue with this plugin	
Google OAuth Credentials plugin 1.330.vf5e86021cb_ec	
This plugin implements the OAuth Credentials interfaces to surface Google Service Account credentials to Jenkins.	
Report an issue with this plugin	

GCP

Name ↓

GCP Secrets Manager Credentials Provider 0.3.1
Source Jenkins Credentials from GCP Secrets Manager.
[Report an issue with this plugin](#)

Google Cloud Platform SDK :: Auth 26.23.0~28.vd30f921a_22a_8
This plugin provides all [Google Cloud Auth SDK for Java](#) modules not packaged as standalone plugins.
[Report an issue with this plugin](#)

Google Cloud Platform SDK :: Storage 26.23.0~28.vd30f921a_22a_8
This plugin provides all [Google Cloud Storage SDK for Java](#) modules not packaged as standalone plugins.
[Report an issue with this plugin](#)

Pipeline: GCP Steps 22.vcc10eff7c13f
This plugin adds Jenkins pipeline steps to interact with the GCP API.
[Report an issue with this plugin](#)

Kubernetes Plugins

kubernetes|

Name ↓

Kubernetes Client API Plugin 6.10.0~240.v57880ce8b_0b_2
Kubernetes Client API plugin for use by other Jenkins plugins.
[Report an issue with this plugin](#)

Kubernetes Credentials Plugin 190.v03c305394deb_
Common classes for Kubernetes credentials
[Report an issue with this plugin](#)

Kubernetes plugin 4306.vc91e951ea_eb_d
This plugin integrates Jenkins with [Kubernetes](#)
[Report an issue with this plugin](#)

Jenkins Advanced Software Installation

This section covers additional software that will need to be installed either via command-line or via Jenkins Web Console. The software installation we will perform at this point will be to support Amazon Web Services Command Line Interface (AWS CLI) and HashiCorp's Terraform Infrastructure as Code software.

In this guide I present commands as they would be run inside of a container. If you are using this guide in a virtual machine or physical Linux system than you will preface each command with “sudo”.

** We login as root because this is a docker container and the jenkins user does not have a password setup. A more secure method would be to login the container as root, create a password for user jenkins, then disconnect from the container and reconnect as user jenkins.

1. Start Docker desktop or docker command line. In this example we will use the Docker Desktop application. Pressing the start button in the Docker Desktop application.
2. Connect to the jenkins container as user root.

```
$ docker exec -it --user root bash
```

3. Install AWS CLI and patch the Jenkins container.

```
$ sudo apt update && sudo apt install -y awscli
```

4. Verify AWS CLI is installed.

```
$ aws --version
```

```
[root@c7f749e1cc85:/# su - jenkins
[jenkins@c7f749e1cc85:~$ id
uid=1000(jenkins) gid=1000(jenkins) groups=1000(jenkins)
[jenkins@c7f749e1cc85:~$ aws --version
aws-cli/2.24.0 Python/3.12.6 Linux/6.12.5-linuxkit exe/aarch64.debian.12
jenkins@c7f749e1cc85:~$ ]
```

5. Install additional software packages

```
$ sudo apt-get install -y wget curl git unzip gnupg software-properties-common
```

6. Install Terraform
 - a. \$ sudo wget -O- <https://apt.releases.hashicorp.com/gpg> | sudo gpg --dearmor | sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null
 - b. \$ sudo gpg --no-default-keyring --keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg --fingerprint
 - c. \$ sudo echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] <https://apt.releases.hashicorp.com> \$(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
 - d. sudo apt update
 - e. sudo apt-get install terraform
7. Verify terraform is installed.

```
$ terraform -version
```

```
jenkins@c7f749e1cc85:~$ terraform -version
Terraform v1.10.5
on linux_arm64
jenkins@c7f749e1cc85:~$ █
```

Modified Jenkins Container Image Creation

Here we will create an image of this container as we have significantly modified it to support our Jenkins deployment.

1. Stop the jenkins container.
2. Run the following command on your local system where you have Docker installed to commit a new docker container image of the jenkins container.

```
$ docker commit jenkins-terraform
```

Note: You should see a new container image in Docker Desktop with the name you provided. It will be much larger than the original Jenkins container due to the software packages that were installed.

3. Start the new container using the following commands:

```
docker run --name=jenkins-terraform-v1 \
-p 8080:8080 \
```

```
-p 50000:50000 \
-v jenkins_home:/var/jenkins_home \
jenkins-terraform
```

Note: In this example I named the container "jenkins-terraform-v1". You can name it according to what you use in your DevOps environment.

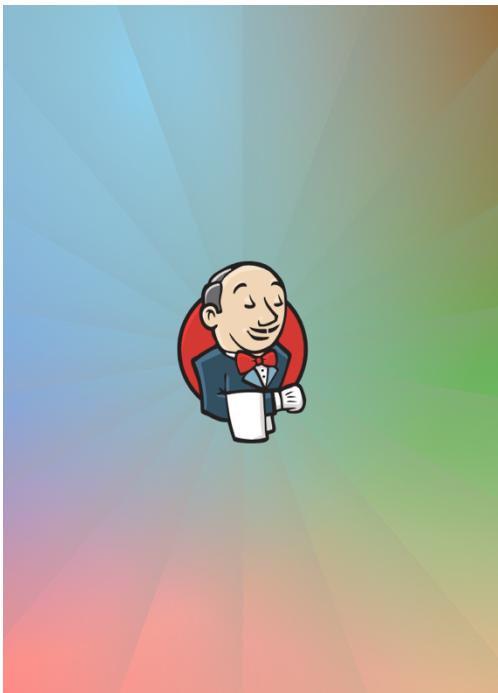
4. Connect to the new container via either Docker Desktop or via command-line. Here we will use the Docker Desktop application. The objective is to verify it is up and running and listening on the ports you specified in your startup command.

The screenshot shows the Docker Desktop interface with the 'Containers' tab selected. The sidebar on the left includes options for Containers, Images, Volumes, Builds, Docker Hub, Docker Scout, and Extensions. The main area displays container statistics: Container CPU usage (0.64% / 600%) and Container memory usage (772.6MB / 7.47GB). A search bar and a filter for 'Only show running containers' are present. A table lists four containers:

Name	Container ID	Image	Port(s)	CPU (%)	Actions
objective_germain	1ff822546d0c	jenkins/jenl	50000:50000 Show all ports (2)	0%	⋮ 🗑
mongo-express	44975b856fcf	mongo-exp	8081:8081	0%	⋮ 🗑
demo-mongodb	12068f916903	mongo	27017:27017	0%	⋮ 🗑
jenkins-terraform-v1	64c995d49ed9	jenkins-terr	50000:50000 Show all ports (2)	0.2%	⋮ 🗑

At the bottom, there are sections for 'Walkthroughs', 'Multi-container applications', 'Containerize your application', and a terminal window showing 'Engine running'. Resource usage stats (RAM 1.44 GB, CPU 0.33%, Disk: 6.56 GB used / limit 164.80 GB) are also displayed.

5. Access the jenkins container via a web browser application and input the username and password from your previous Jenkins container as the account information is stored in your jenkins_home directory.



Sign in to Jenkins

Username

Password

Keep me signed in

A screenshot of a Jenkins sign-in page. It features a "Sign in to Jenkins" header, two input fields for "Username" and "Password", a "Keep me signed in" checkbox, and a prominent blue "Sign in" button.

Jenkins Service Credentials Setup

We need to provide Jenkins to be able to access a given service platform to run commands that we task Jenkins to perform. In this example we will provide Jenkins with account information for AWS. You will create these credentials in the AWS IAM dashboard.

Jenkins AWS IAM Credential

1. Logon to AWS web console and go to the IAM dashboard.
2. You will create an IAM user with programmatic access and ***NOT*** AWS management console access.

It is ***HIGHLY*** recommended to ***NOT*** select the provide user access to AWS Management Console option for these service accounts.
3. Select Users.
4. Select "Create User"
5. Input a Username.
6. Select Next.
7. Select Attach Policies directly.
8. Search and select policies that Jenkins will need to perform pipeline functions in AWS. You need to know the specific policies that Jenkins will need to perform actions in your AWS environment. You
9. Select next.
10. At the "Review and create step" you need to verify you have set the required permissions and you can add Tags.

11. Select Create User, then select create access key. Your access key and secret key will be displayed. Copy and securely store the Access Key and Secret Access Key.

Add Credentials to Jenkins

1. Go to the Jenkins dashboard and select "Manage Jenkins"
2. Select Credentials > System > Global credentials (unrestricted).
3. Select New credentials.
4. In "Kind" click on the down arrow and select "AWS Credentials". **Note:** Do not change the default "Scope" setting.
 - In the ID field, add the name AWS_SECRET_ACCESS_KEY.
 - In the Description field, add the name AWS_SECRET_ACCESS_KEY.
 - In the Access Key ID field, input the access key id from your AWS IAM for the jenkins service.
 - In the Secret Access Key field, input the AWS secret key. Once finished you should have a new credentials record in Jenkins.

The screenshot shows the Jenkins 'New credentials' configuration page for AWS Credentials. The 'Kind' dropdown is set to 'AWS Credentials'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'ID' field contains 'AWS_SECRET_ACCESS_KEY'. The 'Description' field also contains 'AWS_SECRET_ACCESS_KEY'. The 'Access Key ID' field is empty, indicated by a blue border. The 'Secret Access Key' field is empty. The 'IAM Role Support' section is collapsed. At the bottom, there is an 'Advanced' dropdown and a 'Create' button.

The screenshot shows the Jenkins 'Credentials' page. At the top, there's a navigation bar with the Jenkins logo and the word 'Jenkins'. Below it, a secondary navigation bar shows 'Dashboard > Manage Jenkins > Credentials'. The main content area is titled 'Credentials' and contains a table with one row. The table has columns for 'T' (Type), 'P' (Path), 'Store ↓', 'Domain', and 'ID'. The row shows 'System' as the type, 'System' as the path, '(global)' as the domain, and 'AWS_SECRET_ACCESS_KEY' as the ID. Below this table, a section titled 'Stores scoped to Jenkins' is shown, featuring another table with columns for 'P', 'Store ↓', and 'Domains'. It shows 'System' as the store and '(global)' as the domain.

Credentials

T	P	Store ↓	Domain	ID
System	System	(global)		AWS_SECRET_ACCESS_KEY

P	Store ↓	Domains
System	System	(global)

Create a Jenkins Pipeline

We will create a Jenkins Pipeline that will perform tasks in AWS based on what is in a GitHub repository. You should use a repository that you know works before assigning it to Jenkins to avoid any troubleshooting issues if the deployment does not run as expected.

Perform the following steps to create the pipeline in Jenkins:

- Select Dashboard > All > New Item
- Enter an item name, for example "test_pipeline".
- Select Pipeline, then okay.
- Select the pipeline name you entered and then select Configuration.
- Select Advanced Project Options.
- Under "Definition" select Pipeline script from SCM.
- Under SCM click the dropdown and select "Git".
- Under Repositories input the URL to the GitHub repository you intend to use with Jenkins.
- Select branch and change to the name in the GitHub repository. In this example that is "main"
- Select "Save".
- Go back to the Jenkins Dashboard.
- In the GitHub repository you will need to add a Jenkinsfile to the repo you selected. This will ensure that Jenkins uses the desired workflow in the pipeline.

Jenkins

Dashboard > All > New Item

New Item

Enter an item name

 ([X](#))

Setup Jenkins File

At this stage we will be setting up a Jenkins file that will be used to deploy a basic AWS Virtual Private Cloud, with a EC2 instance. Software installation of Docker will be made on the EC2 instance, and a Nginx container will be installed and setup on port 8080 and be accessible via the internet. This will help get an understanding of how a Jenkins file works to support CI/CD and other projects regardless of the complexity of the architecture.

For this guide the Jenkins file is located at

https://github.com/statuc30721/jenkins_terraform_pipeline/blob/main/Jenkinsfile.

The Jenkins file will need to be modified to be used in your environment. You will need to edit the file in a code editor (e.g. Microsoft Visual Studio Code) or an editor you are familiar with. I have provided screenshots of the pipeline to help visualize what should occur as Jenkins processes the instructions to perform the deployment leveraging terraform.

```
pipeline {
    agent any
    environment {
        AWS_REGION = 'us-east-1'
        ACTION = "${params.ACTION}"
    }
    parameters {
        choice (name: 'ACTION',
            choices: [ 'plan', 'apply', 'destroy'],
            description: 'Run terraform plan / apply / destroy')
    }
    stages {
        stage('Set AWS Credentials') {
            steps {
                withCredentials([
                    $class: 'AmazonWebServicesCredentialsBinding',
                    credentialsId: 'AWS_SECRET_ACCESS_KEY'
                ]) {
                    sh ''
                    echo "AWS_ACCESS_KEY_ID: $AWS_ACCESS_KEY_ID"
                    aws sts get-caller-identity
                    ''
                }
            }
        }
        stage('Checkout Code') {
            steps {
                git branch: 'main', url: 'https://github.com/statuc30721/jenkins_terraform_pipeline'
            }
        }
        stage('Initialize Terraform') {
            steps {
                sh ''
                terraform init
                ''
            }
        }
    }
}
```

```

stage('Plan Terraform') {
    when { anyOf
        {
            environment name: 'ACTION', value: 'plan';
            environment name: 'ACTION', value: 'apply'
        }
    }
    steps {
        withCredentials([
            $class: 'AmazonWebServicesCredentialsBinding',
            credentialsId: 'AWS_SECRET_ACCESS_KEY'
        ]) {
            sh ''
            export AWS_ACCESS_KEY_ID=$AWS_ACCESS_KEY_ID
            export AWS_SECRET_ACCESS_KEY=$AWS_SECRET_ACCESS_KEY
            terraform plan -out=tfplan
            '''
        }
    }
}
stage('Apply Terraform') {
    when { anyOf
        {
            environment name: 'ACTION', value: 'apply'
        }
    }
    steps {
        input message: "Approve Terraform Apply?", ok: "Deploy"
        withCredentials([
            $class: 'AmazonWebServicesCredentialsBinding',
            credentialsId: 'AWS_SECRET_ACCESS_KEY'
        ]) {
            sh ''
            export AWS_ACCESS_KEY_ID=$AWS_ACCESS_KEY_ID
            export AWS_SECRET_ACCESS_KEY=$AWS_SECRET_ACCESS_KEY
            terraform apply -auto-approve tfplan
            '''
        }
    }
}
stage('Destroy Terraform') {
    when { anyOf
        {
            environment name: 'ACTION', value: 'destroy'
        }
    }
    steps {
        script {
            def IS_APPROVED = input(
                message: "Destroy Deployed Project ?!",
                ok: "Yes",
                parameters: [
                    string(name: 'IS_APPROVED', defaultValue: 'No', description: 'Think again!!!')
                ]
            )
            if (IS_APPROVED != 'Yes') {
                currentBuild.result = "ABORTED"
                error "User destruction cancelled"
            }

            withCredentials([
                $class: 'AmazonWebServicesCredentialsBinding',
                credentialsId: 'AWS_SECRET_ACCESS_KEY'
            ]) {
                sh ''
                export AWS_ACCESS_KEY_ID=$AWS_ACCESS_KEY_ID
                export AWS_SECRET_ACCESS_KEY=$AWS_SECRET_ACCESS_KEY
                terraform destroy -auto-approve
                '''
            }
        }
    }
}

```

This guide requires a working repository so to reduce the amount of troubleshooting it is recommended to select a working repository that leverages terraform. You then will only need to upload your modified Jenkins file to the repository and run the pipeline builds from within Jenkins.

Repository Revision

1. Clone the repository to your local system using git.

```
$ git clone <repository_url>
```

2. Open the Jenkins file in an editor and make the following revisions:

Line 4: Change the AWS region to the region you want to deploy your project (e.g. eu-west-1).

Lines 20, 53, 76 and 113: Change the credential ID with what you created in the AWS IAM portal and set in the Jenkins credential step.

Line 31: Update the URL with what the actual repository you will be using for your Jenkins pipeline project.

Save the file and push the updated Jenkins file to your repo.

```
git add .
git commit -m "<input your commit message>"
git push
```

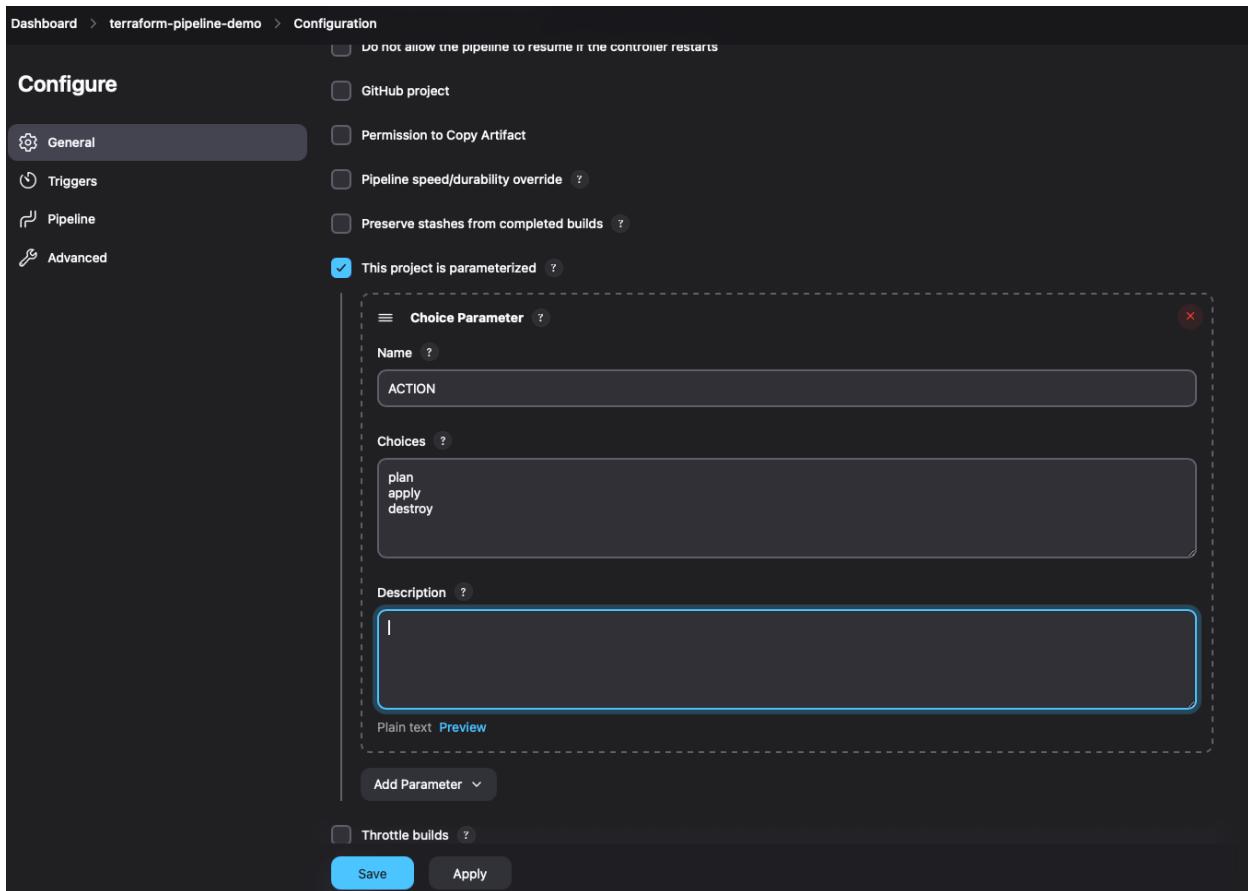
Note: At this point you should have a working Jenkins file located with your repository.

Create a Jenkins Pipeline with Terraform Choices

In this setup we will create a Jenkins pipeline leveraging the Jenkins file you cloned and revised. The file provides three choices for input to Terraform; Plan, Apply and Destroy. When selecting the destroy option you will be required to perform additional steps to get this to work.

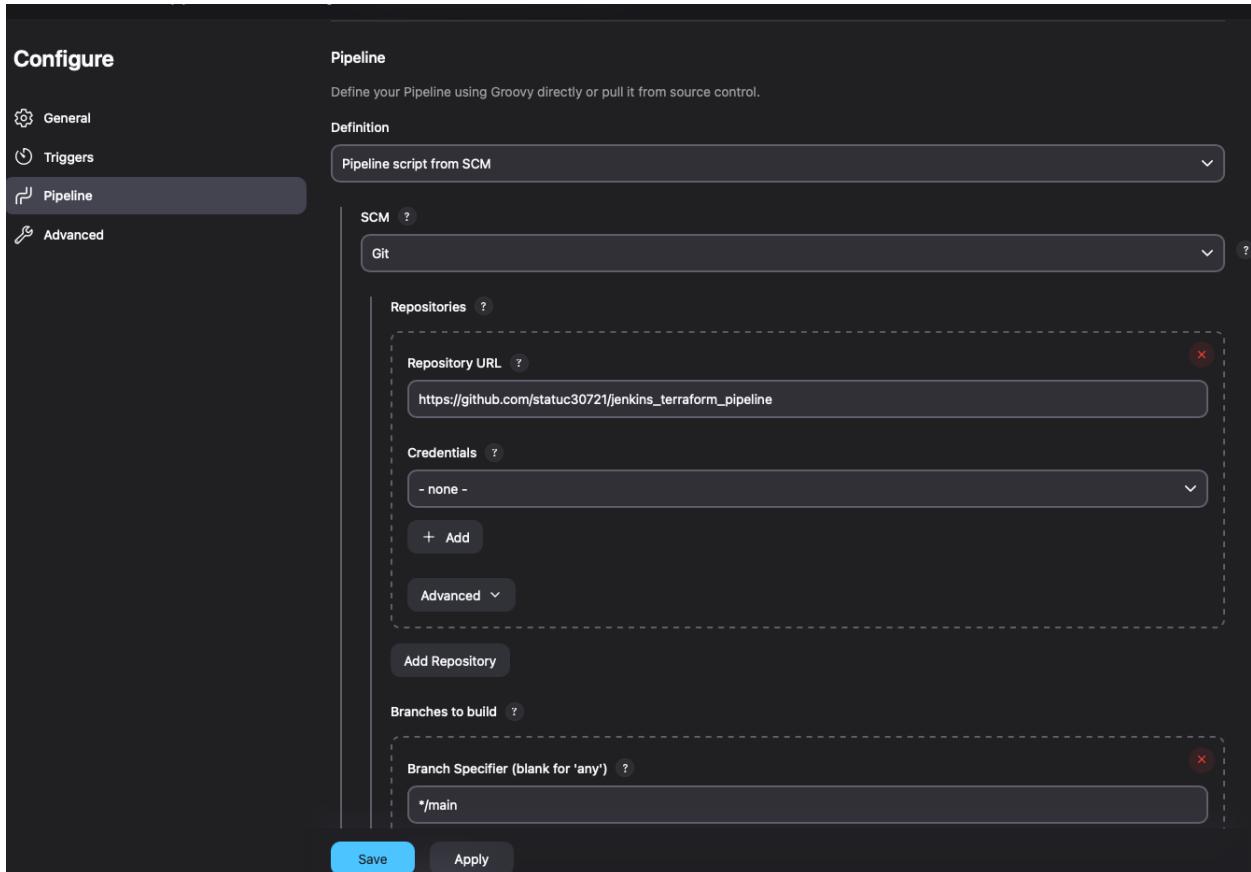
Pipeline Creation

1. Return to the Jenkins Dashboard.
2. Click New Item.
3. Enter a name in the item name field.
4. Select Pipeline and click OK.
5. At the pipeline configuration screen select configuration [Dashboard > pipeline name > Configuration]
6. Input a description of your pipeline (optional).
7. Scroll down the page, and select the box “This project is parameterized”
 - a. Select “Add Parameter” and select “Choice”.
 - b. In the “Name” field input ACTION
 - c. In the “Choices” field input one row at a time
 - plan
 - apply
 - destroy



8. Scroll down to the Pipeline section.
9. Under Definition, select the dropdown “Pipeline script from SCM”.
10. Under SCM select “Git”.

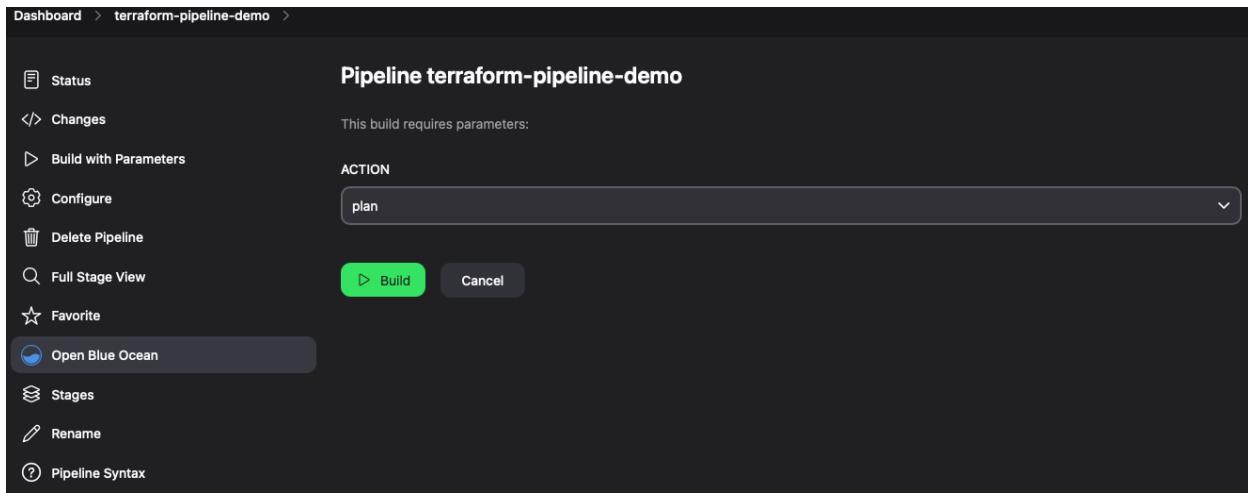
11. Under Repository URL input the URL of the GitHub repository that you will be using that has your Jenkins file.
12. Change the branch from master to the branch you intend to use in your pipeline.
13. Select “Save”.



Run Jenkins Pipeline

Return to the Jenkins dashboard and select the pipeline you want to run.

- Select "Build Now" on the left side menu.
- In this pipeline we have parameters so you will be presented with options you input when setting up the pipeline parameters.
- Our first test run we will run a "Plan" build.



Here we see a console output for terraform plan. Nothing else will occur since the Jenkins file states that we will only run a "terraform plan" and display an output. I added the option to allow a more "pleasant" view of terraforms output as by default in Jenkins it looks like the screenshot below:

```
  }[32m+ 0m 0m "Name" = "demo-vpc"
}

[[1mPlan: 6 to add, 0 to change, 0 to destroy.
0m

Changes to Outputs:
  [32m+ 0m 0m aws-ami_id    = "ami-053a45ffff0a704a47"
  [32m+ 0m 0m ec2-public_ip = (known after apply)
90m

Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:
  terraform apply "tfplan"
```

What we should be able to do is review our plan before we make any changes to our build. So, I added the option to read the resulting output file as pictured below:

```
# aws_vpc.myapp-vpc will be created
+ resource "aws_vpc" "myapp-vpc" {
    + arn                                = (known after apply)
    + cidr_block                         = "10.0.0.0/16"
    + default_network_acl_id            = (known after apply)
    + default_route_table_id           = (known after apply)
    + default_security_group_id        = (known after apply)
    + dhcp_options_id                  = (known after apply)
    + enable_dns_hostnames             = (known after apply)
    + enable_dns_support                = true
    + enable_network_address_usage_metrics = (known after apply)
    + id                                = (known after apply)
    + instance_tenancy                 = "default"
    + ipv6_association_id              = (known after apply)
    + ipv6_cidr_block                  = (known after apply)
    + ipv6_cidr_block_network_border_group = (known after apply)
    + main_route_table_id              = (known after apply)
    + owner_id                           = (known after apply)
    + tags
        + "Name" = "demo-vpc"
    }
    + tags_all                          = {
        + "Name" = "demo-vpc"
    }
}

Plan: 6 to add, 0 to change, 0 to destroy.
```

If everything looks correct, then we can proceed to deploy the project by selecting the option of “Apply”. In this parameter the Jenkins file instructs Jenkins to perform both a “terraform plan” and “terraform apply”. So, if you already know you want to apply a revision you have made in your repository you can select the “apply” option.

Dashboard > **terraform-pipeline-demo** >

Status **Pipeline terraform-pipeline-demo**

</> **Changes** This build requires parameters:

▷ **Build with Parameters** ACTION

Configure Run terraform plan / apply / destroy

Delete Pipeline **apply**

Full Stage View

Favorite **> Build** **Cancel**

Open Blue Ocean

Following selecting apply you will see your changes and if they meet your requirements than you click “Deploy”. If something is wrong or you don’t want to apply changes, then select “Abort”.

```
Plan: 6 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ aws-ami_id      = "ami-053a45fff0a704a47"
+ ec2-public_ip = (known after apply)
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Apply Terraform)
[Pipeline] input
Approve Terraform Apply?
Deploy or Abort
```

Select “Deploy” and then either monitor the logs from within Jenkins as the steps are completed or go to your AWS console and verify your deployment worked. In Jenkins if the deployment was successful, you should see a display like the screenshot from a terraform pipeline.

Here we have selected the “Deploy” option and per the console output Jenkins has successfully deployed the repository project leveraging terraform.

```
Apply complete! Resources: 6 added, 0 changed, 0 destroyed.
```

```
±[0m±[0m±[1m±[32m
```

```
Outputs:
```

```
±[0maws-ami_id = "ami-053a45fff0a704a47"
ec2-public_ip = "3.230.135.241"
[Pipeline] }
[Pipeline] // withCredentials
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Destroy Terraform)
Stage "Destroy Terraform" skipped due to when conditional
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

We visit our AWS console and verify that our project did deploy as expected. In this repository we should see the following:

- One Virtual Private Cloud (VPC)
- One EC2 Virtual Machine Instance
- Services: Docker deployed and running on the EC2 instance. Nginx container deployed on the EC2 instance running under docker.
- Nginx container accessible via the internet at the provided public ip address presented in the Jenkins console output.

The screenshot shows the AWS VPC Details page for 'demo-vpc'. The 'Details' section includes fields like VPC ID (vpc-0a3c5aa2cd566cd2e), State (Available), Main network ACL (acl-0b3236c6f5c3dd8b2), and IPv6 CIDR (Network border group). The 'Resource map' section shows a diagram with four components: 'VPC' (Your AWS virtual network, demo-vpc), 'Subnets (1)' (Subnets within this VPC, us-east-1f, demo-subnet-1), 'Route tables (1)' (Route network traffic to resources, demo-main-rtb), and 'Network connections (1)' (Connections to other networks, demo-igw).

The screenshot shows the AWS Instances page with one instance listed: 'demo-server' (Instance ID: i-0516a0e6e6a699d7c, State: Running, Type: t2.medium, Status check: 2/2 checks passed). The page also includes filters, connectivity options, and instance state dropdowns.

```
Apply complete! Resources: 6 added, 0 changed, 0 destroyed.
Outputs:

awsami_id = "ami-053a45fff0a704a47"
ec2-public_ip = "3.230.135.241"
```

```
aws | Search [Option+S]

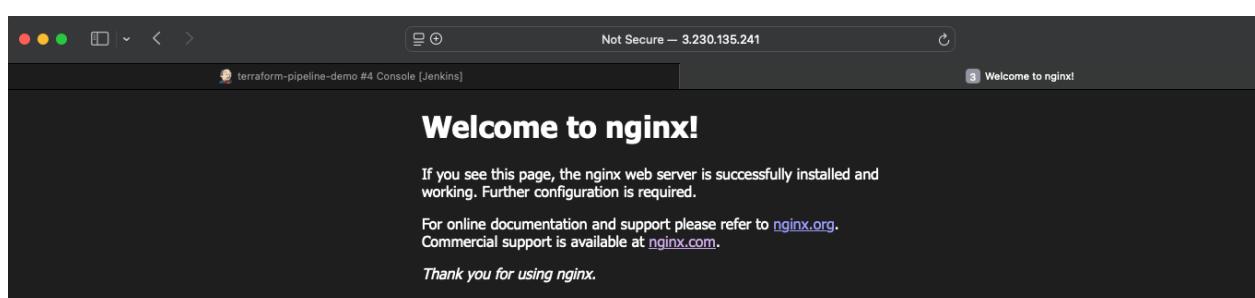
VPC

'`#_#
--\_\#\#\#_ Amazon Linux 2023
-- \#\#\#
-- \#\#|
-- \#/ https://aws.amazon.com/linux/amazon-linux-2023
-- V-,->
--- /
~~-. / / /
~/m

[ec2-user@ip-10-0-34-241 ~]$ ping -c4 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=58 time=1.53 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=58 time=1.60 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=58 time=1.65 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=58 time=1.43 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.430/1.550/1.649/0.082 ms
[ec2-user@ip-10-0-34-241 ~]$ systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: disabled)
     Active: active (running) since Sun 2025-02-16 12:23:50 UTC; 4min 20s ago
TriggeredBy: ● docker.socket
    Docs: https://docs.docker.com
   Process: 4380 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
   Process: 4416 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
 Main PID: 4418 (dockerd)
   Tasks: 28
  Memory: 239.2M
    CPU: 4.684s
   CGroup: /system.slice/docker.service
           └─4418 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536
             ├─13699 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8080 -container-ip 172.17.0.2 -container-port 80
             └─13704 /usr/bin/docker-proxy -proto tcp -host-ip :: -host-port 8080 -container-ip 172.17.0.2 -container-port 80

Feb 16 12:23:49 ip-10-0-34-241.ec2.internal systemd[1]: Starting docker.service - Docker Application Container Engine...
Feb 16 12:23:49 ip-10-0-34-241.ec2.internal dockerd[4418]: time="2025-02-16T12:23:49.815686298Z" level=info msg="Starting up"
Feb 16 12:23:49 ip-10-0-34-241.ec2.internal dockerd[4418]: time="2025-02-16T12:23:49.867894505Z" level=info msg="Loading containers: start"
Feb 16 12:23:50 ip-10-0-34-241.ec2.internal dockerd[4418]: time="2025-02-16T12:23:50.277567500Z" level=info msg="Loading containers: done"
Feb 16 12:23:50 ip-10-0-34-241.ec2.internal dockerd[4418]: time="2025-02-16T12:23:50.298118636Z" level=info msg="Docker daemon" commit=b0...
Feb 16 12:23:50 ip-10-0-34-241.ec2.internal dockerd[4418]: time="2025-02-16T12:23:50.298202278Z" level=info msg="Daemon has completed initialization"
Feb 16 12:23:50 ip-10-0-34-241.ec2.internal dockerd[4418]: time="2025-02-16T12:23:50.331353064Z" level=info msg="API listen on /run/docker.sock"
Feb 16 12:23:50 ip-10-0-34-241.ec2.internal systemd[1]: Started docker.service - Docker Application Container Engine.
[ec2-user@ip-10-0-34-241 ~]$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b523e2c72dbc nginx "/docker-entrypoint..." 4 minutes ago Up 4 minutes 0.0.0.0:8080->80/tcp, :::8080->80/tcp boring_johnson
[ec2-user@ip-10-0-34-241 ~]$ █
```



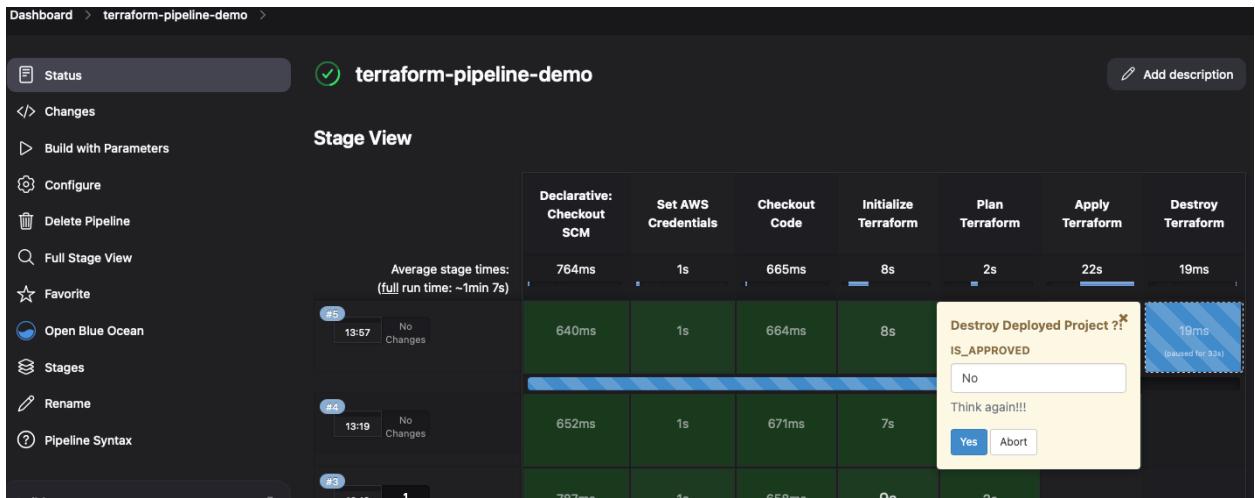
At this stage you now have a working Jenkins deployment that can run terraform and support various CI/CD workflows.

Terraform Destroy Deployed Project

Our final step is to cleanup this guides demonstration project. To do this we will have Jenkins perform the “terraform destroy” action.

WARNING: As a reminder terraform destroy will destroy your deployed project completely. Please be certain this is what you want as it can't be stopped from within Jenkins unlike when using Terraform Enterprise.

1. Select the Jenkins pipeline and under “ACTION”, select the dropdown to “destroy”.
2. Select the “Build” button.
3. You will be presented with the option to destroy your deployed project.

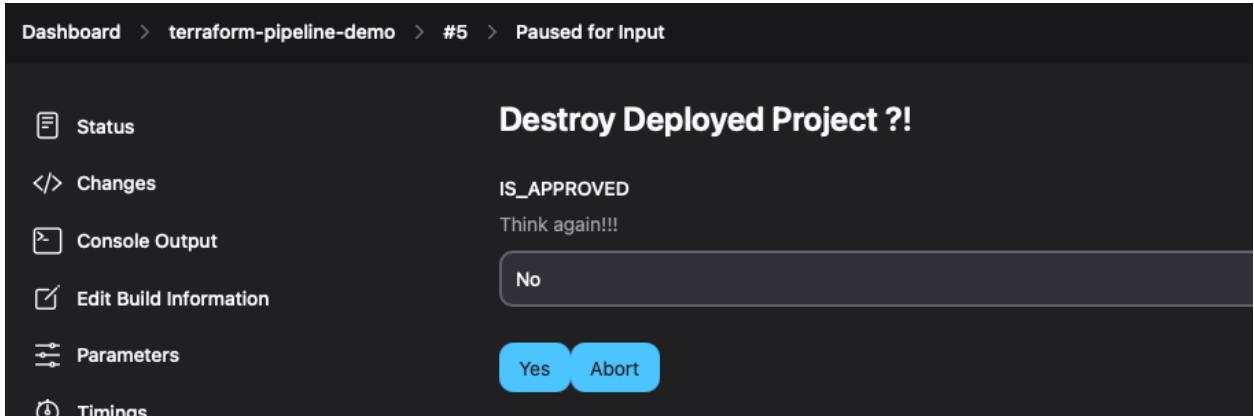


4. If you are in the console of Jenkins, you will see the message “Input requested”. Select the word “input requested”. It will take you to the “Paused for Input” section of the pipeline.

```
[Pipeline] getContext
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Destroy Terraform)
[Pipeline] script
[Pipeline] {
[Pipeline] input
Input requested
```

Note: The Jenkins file is set to “No” by default for *SAFETY*.

If you are sure you want to destroy your project, then at the “Paused for Input” you will have to change the option from “No” to “Yes” and then select the “Yes” button. Just selecting the “Yes” button will not override the value in the dialog box.



Option changed from “No” to “Yes” to proceed with destruction of the project.

The screenshot shows a Jenkins dashboard for a project named 'terraform-pipeline-demo' with build number '#5'. The build status is 'Paused for Input'. A modal dialog box titled 'Destroy Deployed Project ?!' is open. It contains several fields: 'Status' (checkbox), 'Changes' (checkbox), 'Console Output' (checkbox), 'Edit Build Information' (checkbox), 'Parameters' (checkbox), and 'Timings' (checkbox). Below these fields is a text area containing 'IS_APPROVED' and 'Think again!!!'. At the bottom of the dialog are two buttons: 'Yes' (highlighted in blue) and 'Abort'.

After inputting “Yes” and selecting the “Yes” button, you will be taken to the Jenkins console output where you can monitor the destruction of the project.

```
[1maws_default_security_group.default-sg: Destroying... id=sg-032c84a7f8d42008e] [1maws_default_security_group.default-sg: Destruction complete after 0s] [0m
[0m[1maws_subnet.myapp-subnet-1: Destruction complete after 1s] [0m
[0m[1maws_vpc.myapp-vpc: Destroying... [id=vpc-0a3c5aa2cd566cd2e] [0m[0m
[0m[1maws_vpc.myapp-vpc: Destruction complete after 1s] [0m
[0m[1m[32m
Destroy complete! Resources: 6 destroyed.
[0m
[Pipeline]
[Pipeline] // withCredentials
[Pipeline]
[Pipeline] // script
[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // withEnv
[Pipeline]
[Pipeline] // withEnv
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

You should check in your AWS environment that the destruction was successful.

The screenshot shows the AWS CloudWatch Instances page. The heading 'Instances (1/1)' has an 'Info' link. To the right, there are filters for 'Last updated' (less than a minute ago) and a 'Configure' button. Below the heading is a search bar with placeholder text 'Find Instance by attribute or tag (case-sensitive)'. A dropdown menu shows 'All states ▾'. The main table lists one instance:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input checked="" type="checkbox"/> demo-server	i-0516a0e6e6a699d7c	Terminated	t2.medium	-	View alarms +	us-east-1f