INSTALLATION GUIDE OF CLOUD SCHEDULER

Version 1.0

August 19, 2022

CRITICALRIVER TECHNOLOGIES PVT LTD

Table of Contents

[Introduction 3](#_Toc112325846)

[Scope: 3](#_Toc112325847)

[Purpose: 3](#_Toc112325848)

[Audience: 3](#_Toc112325849)

[Environment Setup 4](#_Toc112325850)

[Google Cloud SDK: 4](#_Toc112325851)

[Prerequisites to install Google Cloud SDK: 4](#_Toc112325852)

[Installation of Google Cloud SDK using Apt Repository 4](#_Toc112325853)

[Initialize Cloud SDK 5](#_Toc112325854)

[AWS-CLI: 7](#_Toc112325855)

[Step 1: Curl Command 7](#_Toc112325856)

[Step 2: Unzip the Installer 7](#_Toc112325857)

[Step 3: Install Program 7](#_Toc112325858)

[Step 4: Move the file 7](#_Toc112325859)

[Step 5: Check the version 8](#_Toc112325860)

[Step 6: Authenticate AWS Account through AWS-CLI 8](#_Toc112325861)

[Terraform: 8](#_Toc112325862)

[Installation of Terraform on Ubuntu 8](#_Toc112325863)

[Service Account for terraform: 11](#_Toc112325864)

[Velero 13](#_Toc112325865)

[Create Google Service Account (GSA): 13](#_Toc112325866)

[Create Custom Role with Permissions for the Velero GSA: 14](#_Toc112325867)

[Grant access to Velero 14](#_Toc112325868)

[Velero on AWS 15](#_Toc112325869)

[Jenkins Pipeline Setup: 16](#_Toc112325870)

[Creating Pipeline Jobs in Jenkins: 16](#_Toc112325871)

[Cloud Scheduler 19](#_Toc112325872)

[Job 1: Auto Shutdown and Start-up 19](#_Toc112325873)

[Job 2: VM Start and Shutdown 19](#_Toc112325874)

[Job 3: Terraform restore and destroy 20](#_Toc112325875)

[Job 4: Velero Backup and Restore 20](#_Toc112325876)

[Appendix A: Installation of Jenkins in Ubuntu 24](#_Toc112325877)

[Prerequisites for Jenkins installation: 24](#_Toc112325878)

[Installation of Java: 24](#_Toc112325879)

[Jenkins Installation: 24](#_Toc112325880)

[Starting Jenkins: 25](#_Toc112325881)

[Setting Up Jenkins 25](#_Toc112325882)

[Glossary 30](#_Toc112325883)

[Revision History 31](#_Toc112325884)

**List of Figures**

[Figure 1: Terraform - Add-HashiCorp-signed-gpg-keys 9](#_Toc112243813)

[Figure 2: Terraform - Add-the-HashiCorp-repository 10](#_Toc112243814)

[Figure 3: Terraform - sudo-apt-update 11](#_Toc112243815)

[Figure 4: Terraform - Install-terraform-on-ubuntu-20.04 11](#_Toc112243816)

[Figure 5: Terraform - check-terraform-version 12](#_Toc112243817)

[Figure 6:credentials for Velero 17](#_Toc112243818)

[Figure 7:Jenkins - New Job 18](#_Toc112243819)

[Figure 8:Jenkins - Job Description 19](#_Toc112243820)

[Figure 9: Jenkins - pipeline 19](#_Toc112243821)

[Figure 10: Jenkins - Git repo 20](#_Toc112243822)

[Figure 11: Jenkins File - Auto shutdown and start-up 21](#_Toc112243823)

[Figure 12: Jenkins File - VM Start-up 22](#_Toc112243824)

[Figure 13: Jenkins File - VM Shutdown 22](#_Toc112243825)

[Figure 14: Jenkins File - Terraform 22](#_Toc112243826)

[Figure 15: Jenkins File - Velero Backup 23](#_Toc112243827)

[Figure 16: Jenkins File - Velero Restore 23](#_Toc112243828)

[Figure 17: Jenkins - Unlock Jenkins 25](#_Toc112243829)

[Figure 18: Jenkins - Select Plugin to Install 26](#_Toc112243830)

[Figure 19: Jenkins - Plugin Installation 26](#_Toc112243831)

[Figure 20: Jenkins - Create Admin User 27](#_Toc112243832)

[Figure 21: Jenkins - Admin User Details 27](#_Toc112243833)

[Figure 22: Jenkins - Jenkins URL 28](#_Toc112243834)

[Figure 23: Jenkins - Installation Finish 29](#_Toc112243835)

[Figure 24: Jenkins - Welcome Page 29](#_Toc112243836)

# Introduction

Cloud Scheduler is a customizable solution to auto schedule the active period of cloud compute services and clusters.

Cloud Scheduler provides a simple, straightforward approach to reduce cloud resources costs. With this cloud scheduler, you can schedule non-production resources when idle to avoid incurring additional costs during non-business hours. You can also request on-demand resources as per the requirement.

## Scope:

This document provides the detailed information about the installation and setup of the required software packages and configuration to achieve the process of cost optimization using cloud scheduler.

## Purpose:

The main purpose of this document is to guide the installation and setup of the cloud scheduler which can be used to achieve the cost optimization on various cloud providers like GCP and AWS.

## Audience:

This guide is intended for end-users, administrators, and technical personnel who uses Cloud Scheduler.

# Environment Setup

This section describes the software required for the environment setup and the procedure to install them.

## Google Cloud SDK:

**Google Cloud SDK** (Software Development Kit) provides a set of tools that are used to manage resources hosted on Google Cloud Platform (GCP). The SDK provides gcloud, gsutil, and bq commands with the ability to access the Google Cloud via the terminal.

If you are using instances in Google Cloud Platform Google Cloud SDK will be Preinstall in the instance.

Otherwise here are the steps to install Google Cloud SDK:

### Prerequisites to install Google Cloud SDK:

Login to your Ubuntu system with a sudo privileged account.

Open a terminal and execute the following commands to install required packages on your system.

**$ sudo apt update**

**$ sudo apt install apt-transport-https ca-certificates gnupg**

### Installation of Google Cloud SDK using Apt Repository

Follow below steps:

#### Step 1: Import GPG key

First, import the apt GPG key to your system with below command.

**$ curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add**

#### Step 2: Create a PPA file

Then create a PPA file in your system with referring to the cloud-sdk in Google packages repository

**$ echo "deb https://packages.cloud.google.com/apt cloud-sdk main" | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list**

#### Step 3: Install google-cloud-sdk

Finally, update the apt cache and install Google Cloud Packages on your Ubuntu system

**$ sudo apt update**

**$ sudo apt install google-cloud-sdk**

Press ‘y’ and hit enter for any confirmation asked by the installer.

### Initialize Cloud SDK

After you install Cloud SDK successfully on your system. The next step is to perform initialize the environment with **gcloud init** command. This authorizes Cloud SDK tools to use your Google account credentials to access Google Cloud and manage it.

To initialize the **gcloud CLI**:

#### Step 1: Run gcloud init:

**$ sudo gcloud init**

#### Step 2: Create or select a configuration if prompted

If you are initializing a new **gcloud CLI** installation**, gcloud init** creates a configuration named **default** for you and sets it as the active configuration. If you have existing configurations, **gcloud init** prompts you to choose between three options — re-initialize the active one, switch to another one and re-initialize it, or create a new one.

#### Step 3: Complete the authorization step when prompted

Depending on whether you have previously authorized access to Google Cloud, you might be prompted to log in and grant access in a web browser or to select an existing account.

**$ sudo gcloud init –console-only**

If you used the **--console-only** flag and login is required, you must then copy and paste the provided authorization URL into another browser window and follow the prompts provided.

When this step is completed**, gcloud init** sets the **account**property in the configuration to the specified account.

#### Step 4: Choose a current Google Cloud project if prompted

If you only have access to one project, including the default project for your user account**, gcloud init** selects it for you.

Otherwise, you can select a project from a list of projects for which you have **Owner**, **Editor** or **Viewer** permissions. **gcloud init** sets the **project**property in the configuration to the property you choose. If you have access to more than 200 projects, you will be prompted to enter a project id, create a new project, or list projects.

**This account has a lot of projects! Listing them all can take a while.**

**[1] Enter a project ID**

**[2] Create a new project**

**[3] List projects**

**Please enter your numeric choice:**

#### Step 5: Choose a default Compute Engine zone if prompted

If you don't have the Compute Engine API enabled or have a default zone in your project-level metadata, you will not see this step**. gcloud init** sets the region and zone properties in the configuration using the zone you choose.

When **gcloud init** finishes, it prints the properties in the active configuration to the terminal:

**[compute]**

**region = us-east1**

**zone = us-east1-b**

**[core]**

**account = user@google.com**

**disable\_usage\_reporting = False**

**project = example-project**

You can view these properties at any other time using the **[gcloud config list](https://cloud.google.com/sdk/gcloud/reference/config/list)**command.

## AWS-CLI:

The AWS Command Line Interface (AWS CLI) is an open-source tool that enables you to interact with AWS services using commands in your command-line shell. With minimal configuration, the AWS CLI enables you to start running commands that implement functionality equivalent to that provided by the browser-based AWS Management Console from the command prompt in your terminal program.

The installation of AWS-CLI can be done by executing the following commands:

### Step 1: Curl Command

Use the **curl** command – The **-o** option specifies the file name that the downloaded package is written to. The options on the following example command write the downloaded file to the current directory with the local name awscliv2.zip.

**$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"**

### Step 2: Unzip the Installer

Unzip the installer. If your Linux distribution doesn't have a built-in unzip command, use an equivalent to unzip it. The following example command unzips the package and creates a directory named **aws**under the current directory.

**$ unzip awscliv2.zip**

### Step 3: Install Program

Run the install program. The installation command uses a file named install in the newly unzipped **aws** directory. By default, the files are all installed to **/usr/local/aws-cli,** and a symbolic link is created in **/usr/local/bin**. The command includes **sudo** to grant write permissions to those directories.

**$ sudo ./aws/install**

### Step 4: Move the file

Move the downloaded file to the path, The default value is **/usr/local/bin**.

**$ ./aws/install -i /usr/local/aws-cli -b /usr/local/bin**

### Step 5: Check the version

Now, Check the version of the aws-cli by running the following command. And the output is given below.

**$ aws –version**

**aws-cli/2.4.5 Python/3.8.8 Linux/4.14.133-113.105.amzn2.x86\_64 botocore/2.4.5**

### Step 6: Authenticate AWS Account through AWS-CLI

To create and manage any AWS resource through AWS-CLI, you must authenticate first and this AWS account must be having related permissions for the services.

Run below command and provide the necessary details to authenticate.

**$ aws configure**

**AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE**

**AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY**

**Default region name [None]: us-west-2**

**Default output format [None]: json**

For more information on authenticating via AWS-CLI, please follow official documentation from AWS (link is below):

<https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html>

## Terraform:

There are many tools to implement infrastructure as code like **Ansible**, **CloudFormation**, **Azure Resource Manager**, **Chef**, **Google Cloud deployment manager** and more. Terraform is one of the powerful tools (by **HashiCorp**) that belong to this list. It helps to build, change, and manage your infrastructure easily with versioning manifest. Every cloud vendor has its own **IaC** tool but **Terraform** is cloud-agnostic. So, you can use a single tool to manage your multi-cloud environment. Infrastructure as Code (IaC) is a term used when you codify your whole infrastructure for automation with DevOps.

### Installation of Terraform on Ubuntu

#### **Step1- Register HashiCorp GPG keys**

To check the authenticity of packages, you must add **HashiCorp** **signed gpg keys** to your system. You will see the "**OK**" message if it is added successfully. Copy this command except for the**$** sign and paste it to your Ubuntu system.

**$ curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -**

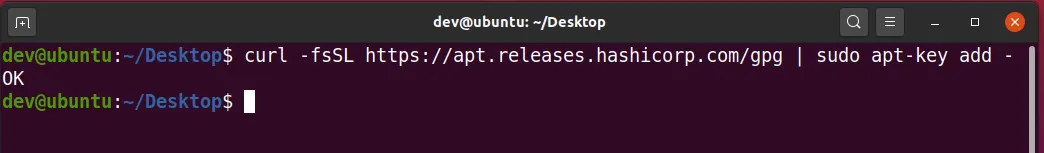


Figure 1: Terraform - Add-HashiCorp-signed-gpg-keys

\*\*\* If you are getting "**curl: command not found error**". Please install it first using below mentioned command.

**$ sudo apt install curl**

#### **Step2- Add HashiCorp package repository**

After adding the gpg key successfully. You need to add the *HashiCorp repository* to download and install terraform packages using **apt**.

**$ sudo apt-add-repository "deb [arch=$(dpkg --print-architecture)] https://apt.releases.hashicorp.com $(lsb\_release -cs) main"**

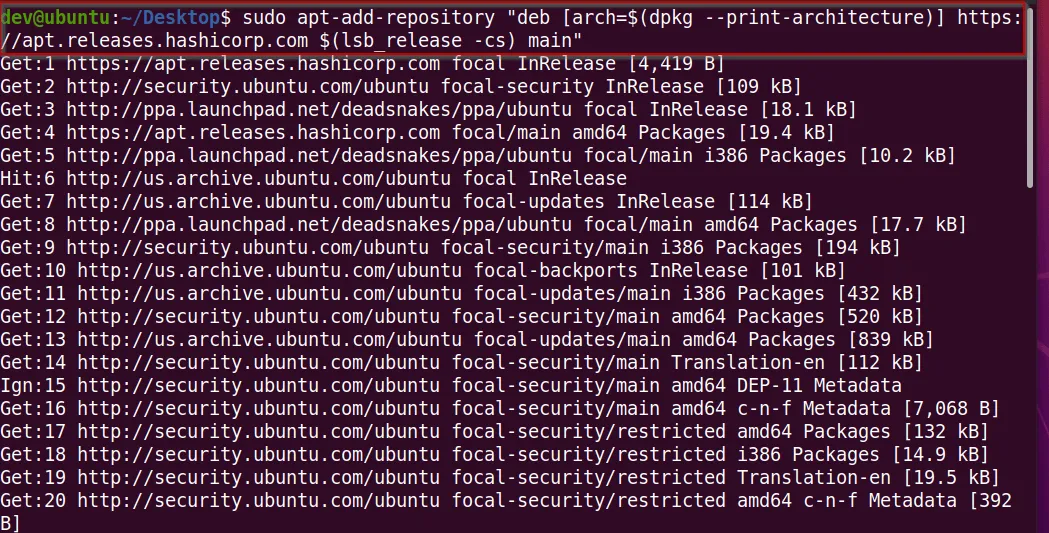


Figure 2: Terraform - Add-the-HashiCorp-repository

#### **Step3- Update "Ubuntu" packages list**

Always update packages list once you install and add any repository in Ubuntu.

**$ sudo apt update**

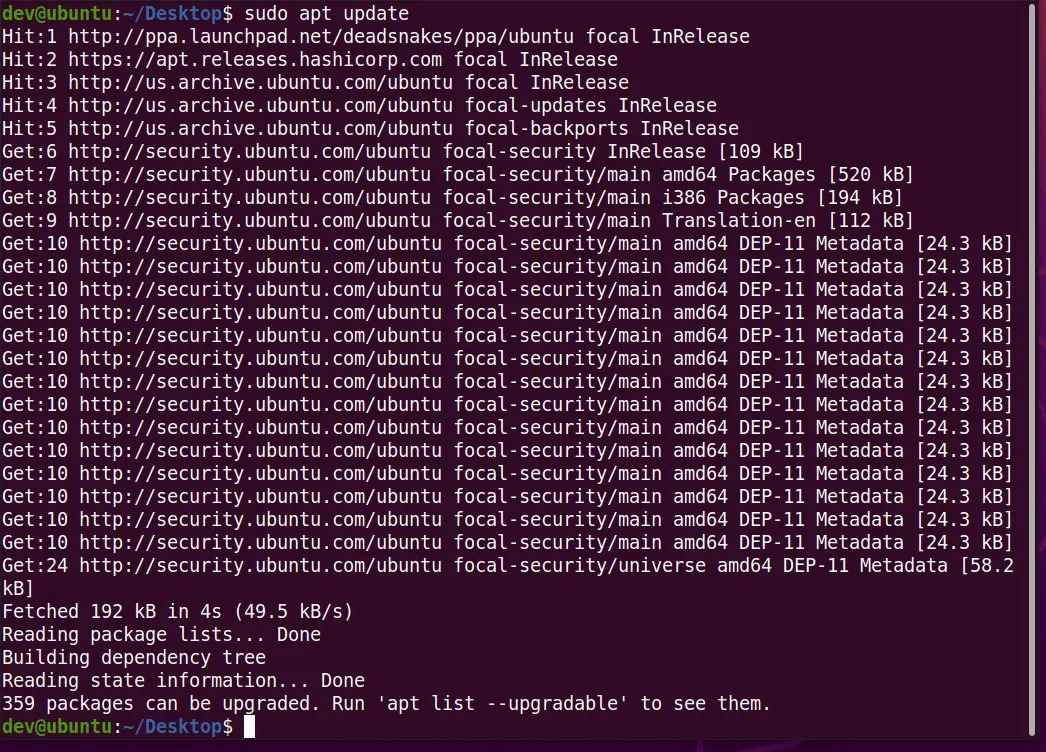


Figure 3: Terraform - sudo-apt-update

#### **Step4- Install Terraform on Ubuntu**

Please run following command to install Terraform.

**$ sudo apt install terraform**

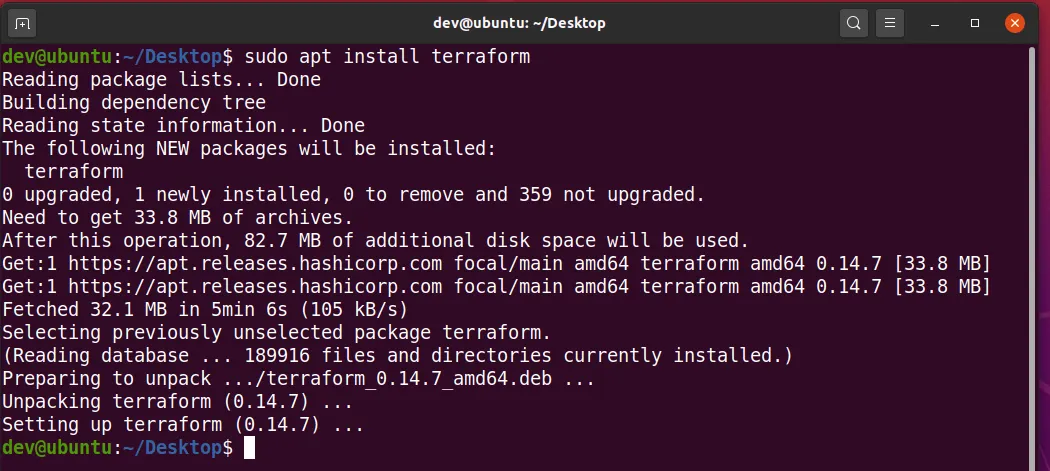


Figure 4: Terraform - Install-terraform-on-ubuntu-20.04

#### **Step5 - Check Terraform version**

Run**"terraform -v"**command to get the version.

Also, you can run "**which"**command to check the "**PATH**" of terraform command. It is**"/usr/bin/terraform**" as shown in image.

**# Check version of Terraform**

**$ terraform-v**

**# Check PATH of Terraform**

**$ which**

Text

Description automatically generated

Figure 5: Terraform - check-terraform-version

### Service Account for terraform:

#### Google Cloud Platform:

In Google Cloud Platform you need to create a service account with the following permissions for terraform to create resources

You need to create a service account for terraform with the following permissions:

* 1. compute.admin
  2. iam.serviceAccountUser
  3. resourcemanager.projectIamAdmin
  4. container.clusterAdmin
  5. compute.vieyour
  6. compute.securityAdmin
  7. container.developer
  8. iam.serviceAccountAdmin
  9. resourcemanager.projectIamAdmin
  10. storage.admin

To create service account, follow the below steps:

##### Step1: Create service account

To create the service account, run the **gcloud iam service-accounts create**command:

**gcloud iam service-accounts create SERVICE\_ACCOUNT\_ID \  
    --description="DESCRIPTION" \  
    --display-name="DISPLAY\_NAME"**

Replace the following values:

* + *SERVICE\_ACCOUNT\_ID*: the ID for the service account
  + *DESCRIPTION*: an optional description of the service account
  + *DISPLAY\_NAME*: a service account name to display in the console

##### Step2: Grant Policy to service account

To grant your service account an **IAM role** on your project, run the **gcloud projects add-iam-policy-binding** command:

**gcloud projects add-iam-policy-binding *PROJECT\_ID* \  
    --member="serviceAccount:*SERVICE\_ACCOUNT\_ID*@*PROJECT\_ID*.iam.gserviceaccount.com" \  
    --role="*ROLE\_NAME*"**

Replace the following values:

* + *PROJECT\_ID*: the project ID
  + *SERVICE\_ACCOUNT\_ID*: the service account ID
  + *ROLE\_NAME*: a role name, such as roles/compute.admin

#### AWS:

You need to create an IAM User for Terraform with the following permissions:

* IAM User with Administrative Access
* Amazon EC2 Full Access
* Amazon S3 Full Access
* Amazon EKS Full Access
* Amazon DynamoDB Full Access
* Amazon VPC Full Access

The above-mentioned permissions need to be attached to the IAM User created for Provisioning AWS Resources through Terraform and AWS-CLI.

To create service account, run below commands in AWS-CLI:

##### Step 1: Create User

**$ aws iam create-user**

##### Step 2: Create Role

**$ aws iam create-role --role-name EXAMPLE\_ROLE**

##### Step 3: Attach Policy to Role

**$ aws iam attach-role-policy --policy-arn arn:aws:iam::ACCOUNT\_ID:policy/EXAMPLE\_POLICY --role-name AmazonEKS\_EBS\_CSI\_DriverRole**

## Velero

### Create Google Service Account (GSA):

To integrate Velero with GCP, create a Velero-specific Service Account. Please run following commands in GCloud-CLI:

#### Step 1: View the current config

View your current config settings:

**gcloud config list**

#### Step 2: Variable

Store the project value from the results in the environment variable $PROJECT\_ID.

**PROJECT\_ID=$(gcloud config get-value project)**

#### Step 3: Create a service account:

**GSA\_NAME=velero**

**gcloud iam service-accounts create $GSA\_NAME \**

**--display-name "Velero service account"**

Then list all accounts and find the velero account you just created:

**gcloud iam service-accounts list**

Set the $SERVICE\_ACCOUNT\_EMAIL variable to match its email value.

**SERVICE\_ACCOUNT\_EMAIL=$(gcloud iam service-accounts list \**

**--filter="displayName:Velero service account" \**

**--format 'value(email)')**

### Create Custom Role with Permissions for the Velero GSA:

These permissions are required by Velero to manage snapshot resources in the GCP Project. Paste below lines in CLI and run:

**```bash**

**ROLE\_PERMISSIONS=(**

**compute.disks.get**

**compute.disks.create**

**compute.disks.createSnapshot**

**compute.snapshots.get**

**compute.snapshots.create**

**compute.snapshots.useReadOnly**

**compute.snapshots.delete**

**compute.zones.get**

**storage.objects.create**

**storage.objects.delete**

**storage.objects.get**

**storage.objects.list**

**)**

**gcloud iam roles create velero.server \**

**--project $PROJECT\_ID \**

**--title "Velero Server" \**

**--permissions "$(IFS=","; echo "${ROLE\_PERMISSIONS[\*]}")"**

**gcloud projects add-iam-policy-binding $PROJECT\_ID \**

**--member serviceAccount:$SERVICE\_ACCOUNT\_EMAIL \**

**--role projects/$PROJECT\_ID/roles/velero.server**

**```**

### Grant access to Velero

This involves creating a Google Service Account Key and using it as --secret-file during installation.

Create a service account key, specifying an output file (credentials-velero) in your local directory:

**gcloud iam service-accounts keys create credentials-velero \**

**--iam-account $SERVICE\_ACCOUNT\_EMAIL**

### Velero on AWS

#### Creating AWS IAM Account:

To integrate Velero with AWS, create a Velero-specific IAM User. Please run following commands**:**

**aws iam create-user –user-name velero**

#### Create Custom Role policies for Velero IAM User:

Add the following permissions to the Velero IAM User

**{  
 "Version":"2012-10-17",  
 "Statement":[  
 {  
 "Effect":"Allow",  
 "Action":[  
 "ec2:DescribeVolumes",  
 "ec2:DescribeSnapshots",  
 "ec2:CreateTags",  
 "ec2:CreateVolume",  
 "ec2:CreateSnapshot",  
 "ec2:DeleteSnapshot"  
 ],  
 "Resource":"\*"  
 },  
 {  
 "Effect":"Allow",  
 "Action":[  
 "s3:GetObject",  
 "s3:DeleteObject",  
 "s3:PutObject",  
 "s3:AbortMultipartUpload",  
 "s3:ListMultipartUploadParts"  
 ],  
 "Resource":[  
 "arn:aws:s3:::${VELERO\_BUCKET}/\*"  
 ]  
 },  
 {  
 "Effect":"Allow",  
 "Action":[  
 "s3:ListBucket"  
 ],  
 "Resource":[  
 "arn:aws:s3:::${VELERO\_BUCKET}"  
 ]  
 }  
 ]  
 }**

#### Configuring AWS Credentials:

To configure Velero with AWS, please run following command:

**aws configure**

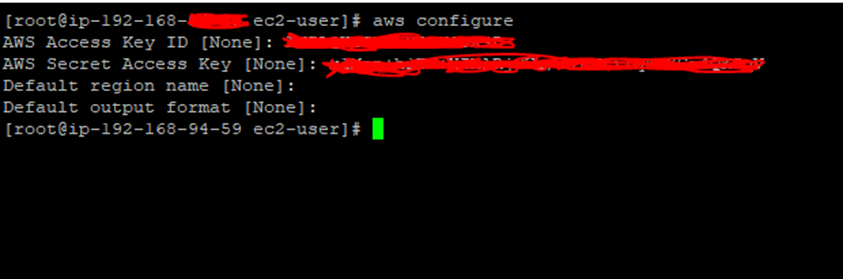


Figure 6: Credentials for Velero

Replace the Access key, secret key, default region and default output format with specific values.

## Jenkins Pipeline Setup:

Please refer “Appendix A: Installation of Jenkins in Ubuntu” for the steps to install and configure the Jenkins in Ubuntu. For other distributions or OS, please refer official documentation.

### Creating Pipeline Jobs in Jenkins:

**Step 1:** In Jenkins, click on**‘New Item’** to create a new project.

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 7:Jenkins - New Job

**Step 2: Give your project a name, then choose ‘Pipeline’ and finally, click on ‘OK’.**

Graphical user interface, text

Description automatically generated with medium confidence

Figure 8:Jenkins - Job Description

**Step3:** In the Definition choose Pipeline script from SCM from the Drop Down and Git in the SCM from the drop down.

**Step4:** Input the GitHub private SSH repo URL in Repository URL. In the credentials field choose the GitHub credentials from the drop down.

Graphical user interface, text, application, email

Description automatically generated

Figure 9: Jenkins - pipeline

**Step 5**: Specify the branch name of GitHub and the pipeline script path.

Graphical user interface, text, application, email, Teams

Description automatically generated

Figure 10: Jenkins - Git repo

# Cloud Scheduler

For the cloud scheduler you must store the code in the GitHub repository. And create four pipeline jobs in Jenkins.

## Job 1: Auto Shutdown and Start-up

The Auto Shutdown and Start-up job will schedule the resources with the help of cron jobs. You need to give the start-up and shut down time in the auto start-up and shutdown Jenkins file (autostsh.jenkins). In the Jenkins file under **pipelineTriggers** you need to make the changes. You need to provide the time as cron job and the time zone of the resources. Along with the time in cron job you must provide the variable and the value of the variable.

Below is as example:

Text

Description automatically generated with low confidence

Figure 11: Jenkins File - Auto shutdown and start-up

In this example time zone is set as Asia/Calcutta i.e., **TZ=Asia/Calcutta**

For the start-up job of SBX environment in GCP you are setting time as 09:15 AM every day from Monday to Friday and passing the variable and it parameter as **Environs=SBX;Mode=STARTUP;Provier=GCP.**

For the start-up job of SBX environment in GCP you are setting time as 06:45 PM every day from Monday to Friday passing the variable and it parameter as **Environs=SBX;Mode=STARTUP;Provier=GCP.**

For cron job you can refer this website to convert time to cron [Crontab.guru - The cron schedule expression editor](https://crontab.guru/)

## Job 2: VM Start and Shutdown

The VM Start ad shutdown job will start and shutdown the VM’s as per the schedule. In the Jenkins file (vmstsh.jenkins) you have to provide the desired VM details. In the Jenkins file under the **stage,** you must provide the instance details such as: instance name, instance zone and the instance project id.

Below is as example:

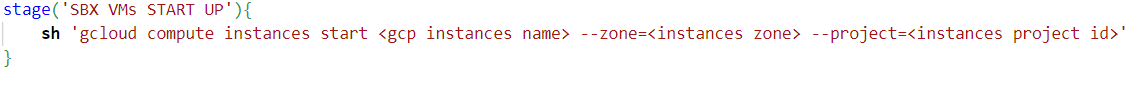


Figure 12: Jenkins File - VM Start-up

In the example you must provide the instance name, instances zone and instances project id for the instance you like to include in SBX environment.

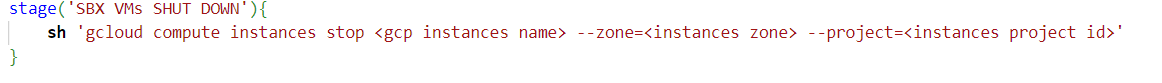


Figure 13: Jenkins File - VM Shutdown

Same way for shutdown also you must provide the instance name, instances zone and instances project id for the instance you like to include in SBX environment.

## Job 3: Terraform restore and destroy

The Terraform restore and destroy pipeline job will create the cluster upon receiving restore parameter and will destroy the cluster upon receiving destroy parameter. In the Jenkins file tfcrdt.jenkins you have to provide the cluster name and the zone of the cluster under stage.

Text

Description automatically generated

Figure 14: Jenkins File - Terraform

For this example of SBX Environment you must provide the GCP cluster name and the zone of the cluster.

You have to provide the details of the cluster in the variables\_auto.tfvars file.

## Job 4: Velero Backup and Restore

In the Velero backup and restore job will perform the backup of cluster and restoring of the cluster. In the Jenkins file vlrcbc.jenkins you have to provide the cluster name and the bucket name.

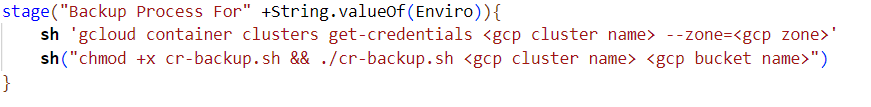


Figure 15: Jenkins File - Velero Backup

In the example you must provide the GCP cluster name and GCP zone and GCP bucket name

Same way for restore also you must provide the details.

Text

Description automatically generated with medium confidence

Figure 16: Jenkins File - Velero Restore

In the example you must provide the GCP cluster name, GCP zone and GCP bucket name.

**Note:**

1. The same pipeline jobs can be configured for AWS with the help of provided pipeline Jenkins templates in the artifacts. Similar configurations can be maintained for EC2 instances, S3 buckets for backups and restoration of EKS clusters through Velero.
2. You can verify the backups by checking the S3 buckets in AWS or Cloud Storage in GCP.
3. The scripts and templates for AWS is provided in the mentioned GitHub Repository used for this cloud scheduler and packaged as Artifacts.

**Troubleshoot:**

**Issues facing while installing or updating AWS-CLI:**

While setting up AWS-CLI make sure you are installing the latest version. And you may also phase some common errors like not moving the downloaded binary file to the correct location. And check the version after moving the file to a particular location.

Correct location: **$ ./aws/install -i /usr/local/aws-cli -b /usr/local/bin**

**Issues facing while authenticating AWS accounts through AWS-CLI:**

While running the $ aws configure command need to double-check whether we are pasting the right credentials like access key, secret access key, and region code.

**Issues facing while installing Terraform:**

While installing Terraform make sure you are installing the latest version of Terraform. And while moving the downloaded binary file make sure that you are moving the file to the correct location.

Correct location: **sudo mv terraform /usr/local/bin**

Check Version: **terraform -version**

**Terraform v1.2.6**

**on linux\_amd64**

**Issues facing while creating IAM Users, Roles, and Policies through AWS-CLI:**

While creating IAM Users, Roles, and Policies through AWS-CLI make sure you installed and configured the latest version of AWS-CLI.

Run the below command and provide the necessary details to authenticate.

**$ aws configure**

**AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE**

**AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY**

**Default region name [None]: us-west-2**

**Default output format [None]: json**

**Issues facing while authenticating Velero to perform operations:**

Make sure that you have attached enough permissions for IAM User Velero which is mentioned in Velero part for AWS.

**Issues facing while creating Pipeline Jobs:** Before creating the Pipeline jobs in Jenkins the recommended way is to create two worker nodes separated for both AWS and GCP cloud environments to avoid conflicts while performing the operations. For creating the worker nodes or Slave nodes under Jenkins master please refer to this link and follow the steps provided in it.

<https://chathura-siriwardhana.medium.com/step-by-step-guide-to-add-jenkins-slave-nodes-f2e756c8849e>

1. While creating Pipeline Job 1 named Auto startup and shutdown we need to specify the cron jobs correctly according to the requirements. Need to check the timing format carefully when to start and when to stop.

For reference purposes please use this link <https://crontab.guru/#5_01_1-31_1-12_1-5>

1. While creating Pipeline Job 2 named VM Start and Shutdown we need to make sure that you have installed the latest version of AWS-CLI in the worker node which we are going to perform the operations. If not, please follow the steps provided for installing and configuring AWS-CLI. And we need to provide the correct instance Ids which to start and which to stop.
2. While creating Pipeline Job 3 named Terraform restore and destroy we need to make sure that the latest version of Terraform is installed and configured in the worker nodes. And make sure that provided terraform templates are valid and working and the Jenkins file script path and working directory are placed correctly which is located in SCM.
3. While creating Pipeline Job 4 named Velero Backup and restore we need to provide the provider and resources details which should take the backup and restore. If you are facing any type of issues while creating this pipeline first check that Velero is working locally. To check Velero backup and restore are working locally please follow this link and execute the steps provided.

Velero installation and working reference links

* <https://katharharshal1.medium.com/backup-and-restore-eks-kubernetes-using-velero-32b11cb55b81>
* <https://velero.io/docs/v1.8/>

# Appendix A: Installation of Jenkins in Ubuntu

## Prerequisites for Jenkins installation:

Minimum hardware requirements:

* 256 MB of RAM
* 1 GB of drive space

Recommended hardware configuration for a small team:

* 4 GB+ of RAM
* 50 GB+ of drive space

## Installation of Java:

Java is required to run Jenkins.

To install Java, run following in terminal:

**$ sudo apt update**

**$ sudo apt install openjdk-11-jre**

## Jenkins Installation:

The version of Jenkins included with the default Ubuntu packages is often behind the latest available version from the project itself. To ensure you have the latest fixes and features, use the project-maintained packages to install Jenkins.

First, add the repository key to the system:

**$ wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add –**

After the key is added the system will return with OK.

Next, let’s append the Debian package repository address to the server’s ‘**sources.list’**:

**$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'**

After both commands have been entered, you’ll run update so that apt will use the new repository.

**$ sudo apt update**

Finally, you’ll install Jenkins and its dependencies.

**$ sudo apt install jenkins**

Now that Jenkins and its dependencies are in place, you’ll start the Jenkins server.

## Starting Jenkins:

Let’s start Jenkins by using [systemctl](https://www.digitalocean.com/community/tutorials/how-to-use-systemctl-to-manage-systemd-services-and-units):

**$ sudo systemctl start Jenkins**

Since **systemctl** doesn’t display status output, you’ll use the status command to verify that Jenkins started successfully:

**$ sudo systemctl status Jenkins**

Now that Jenkins is up and running, let’s adjust your firewall rules so that you can reach it from a web browser to complete the initial setup.

## Setting Up Jenkins

To set up your installation, visit Jenkins on its default port, 8080, using your server domain name or IP address: http://your\_server\_ip\_or\_domain:8080

You should receive the Unlock Jenkins screen, which displays the location of the initial password:

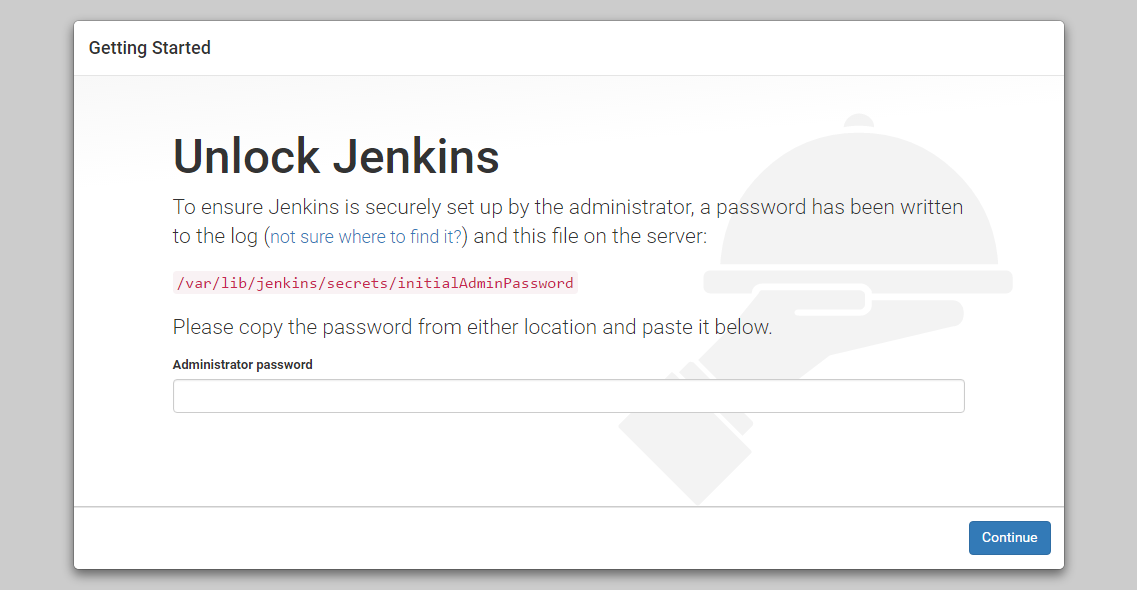


Figure 17: Jenkins - Unlock Jenkins

In the terminal window, use the **cat** command to display the password:

**$ sudo cat /var/lib/Jenkins/secrets/initialAdminPassword**

Copy the 32-character alphanumeric password from the terminal and paste it into the Administrator password field, then click Continue.

The next screen presents the option of installing suggested plugins or selecting specific plugins:

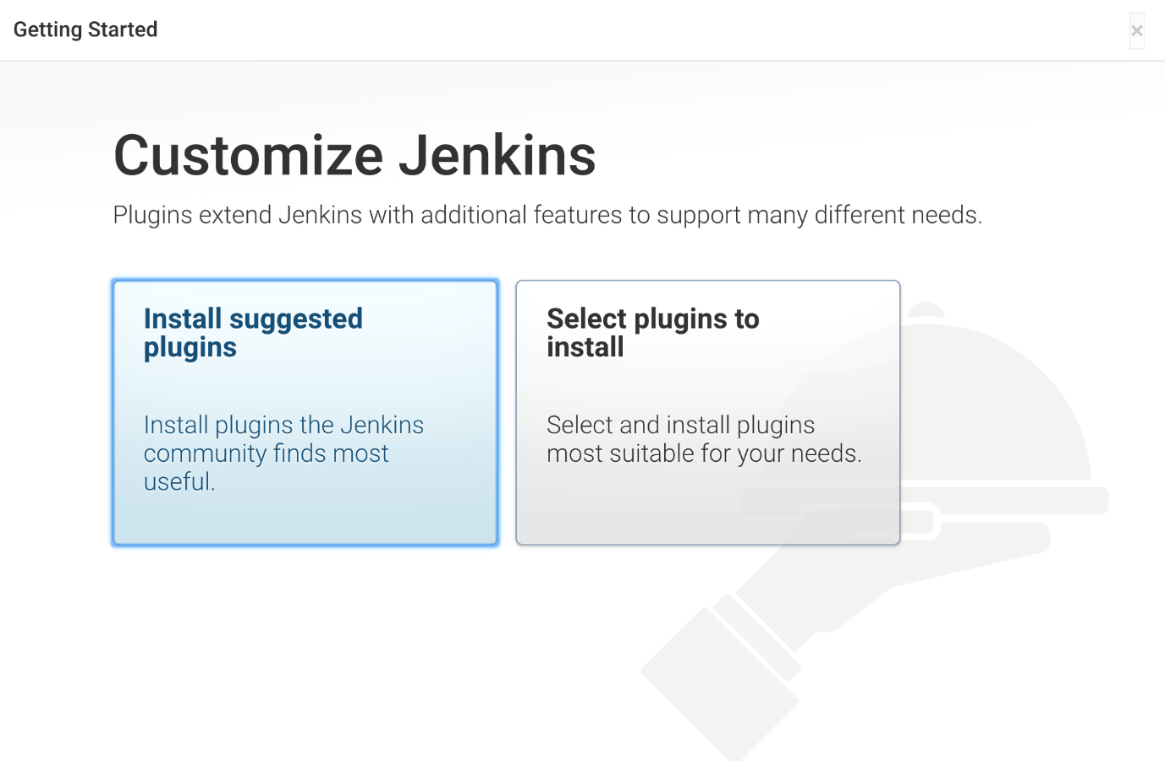


Figure 18: Jenkins - Select Plugin to Install

Click the Install suggested plugins option, to begin the installation process.

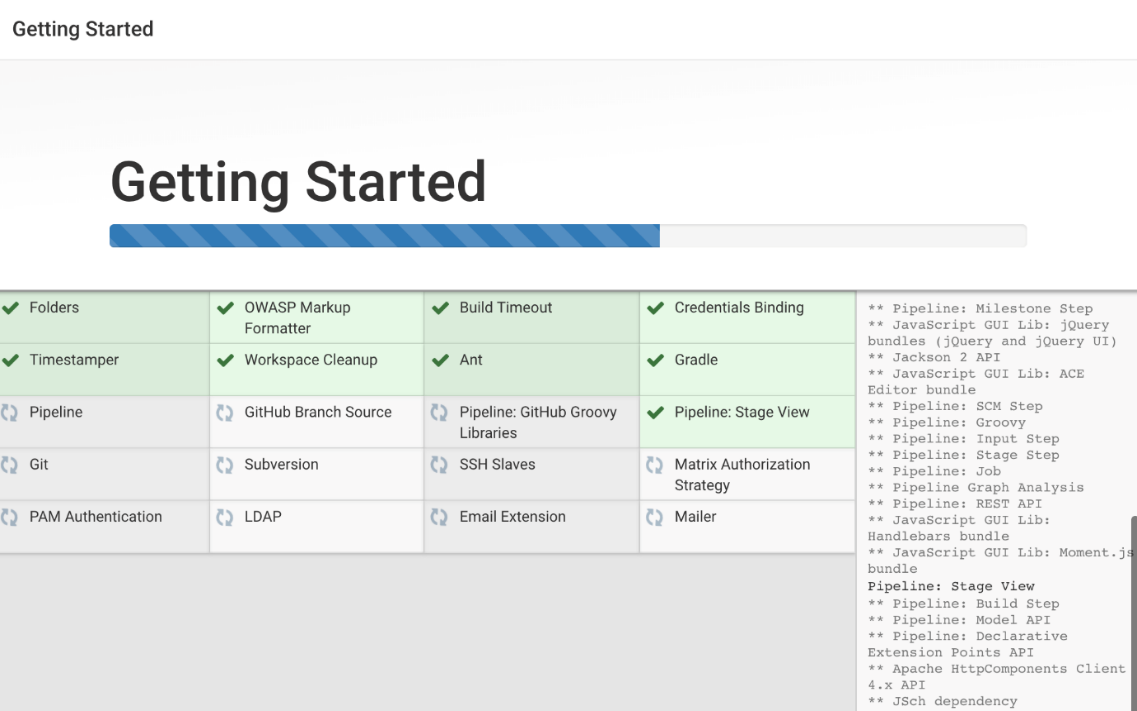


Figure 19: Jenkins - Plugin Installation

When the installation is complete, you’ll be prompted to set up the first administrative user. It’s possible to skip this step and continue as admin using the initial password you used above, but you’ll take a moment to create the user.

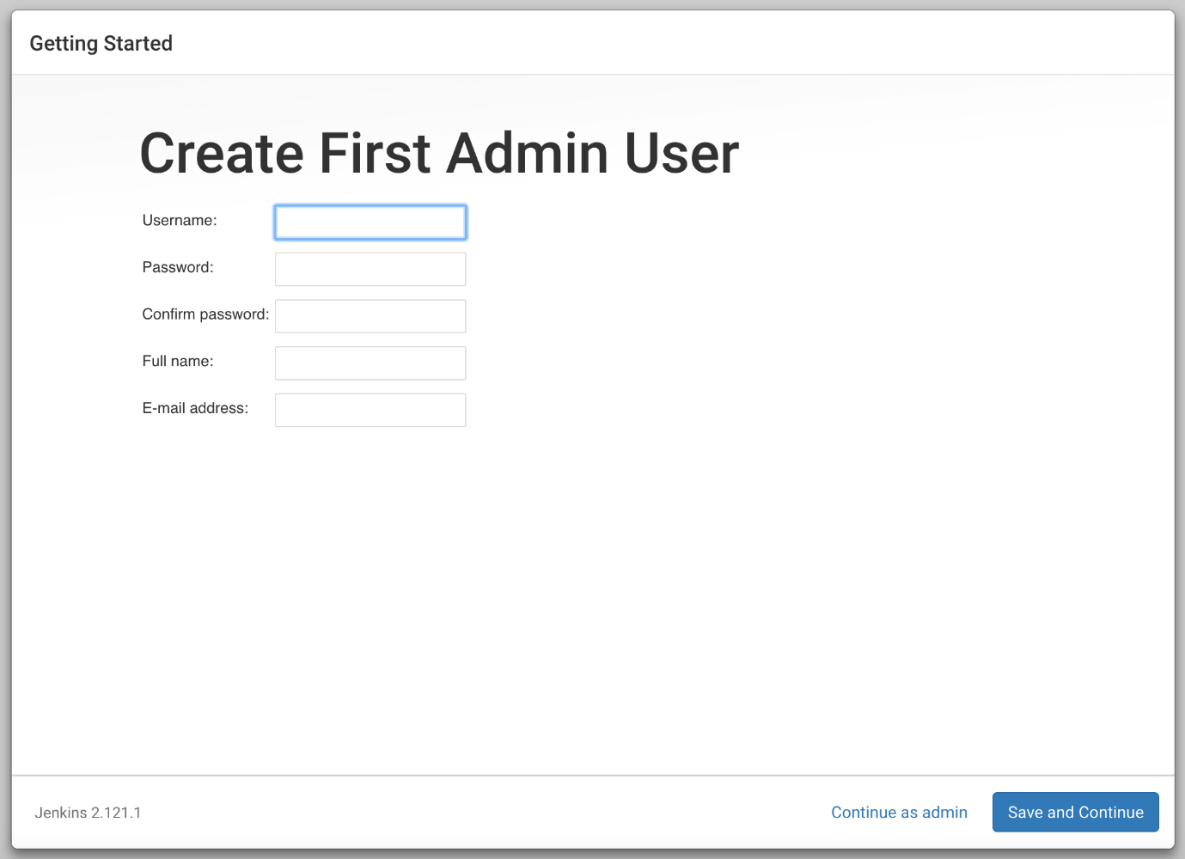


Figure 20: Jenkins - Create Admin User

Enter the name and password for your user:

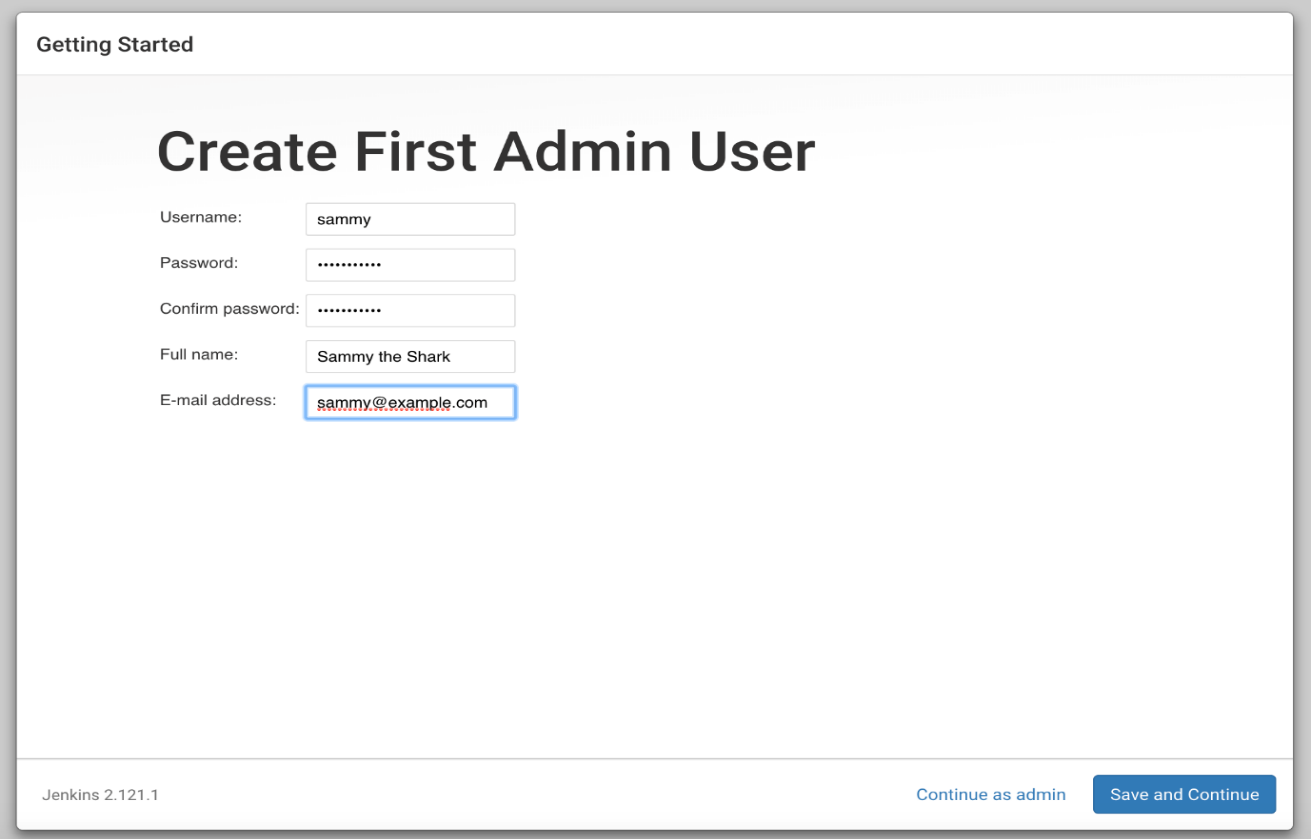


Figure 21: Jenkins - Admin User Details

You’ll receive an **Instance Configuration** page that will ask you to confirm the preferred URL for your Jenkins instance. Confirm either the domain name for your server or your server’s IP address:

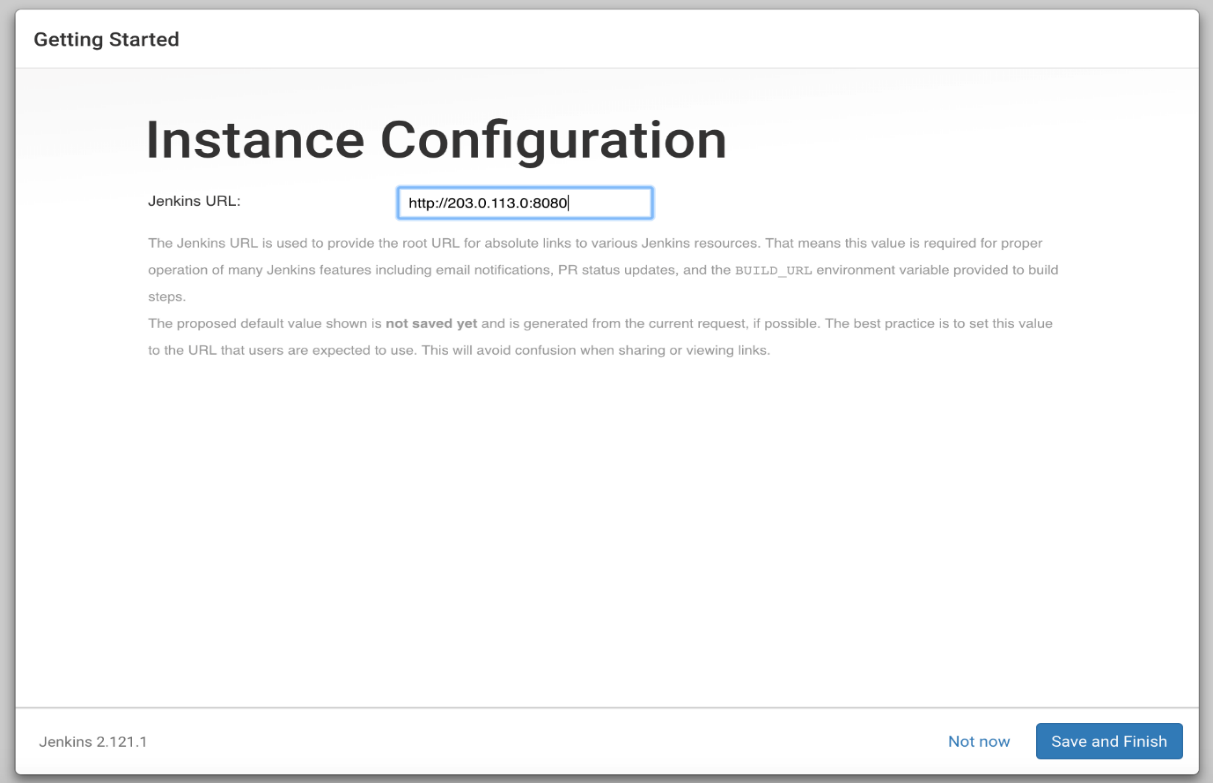


Figure 22: Jenkins - Jenkins URL

After confirming the appropriate information, click **Save and Finish**. You’ll receive a confirmation page confirming that **“Jenkins is Ready!”:**

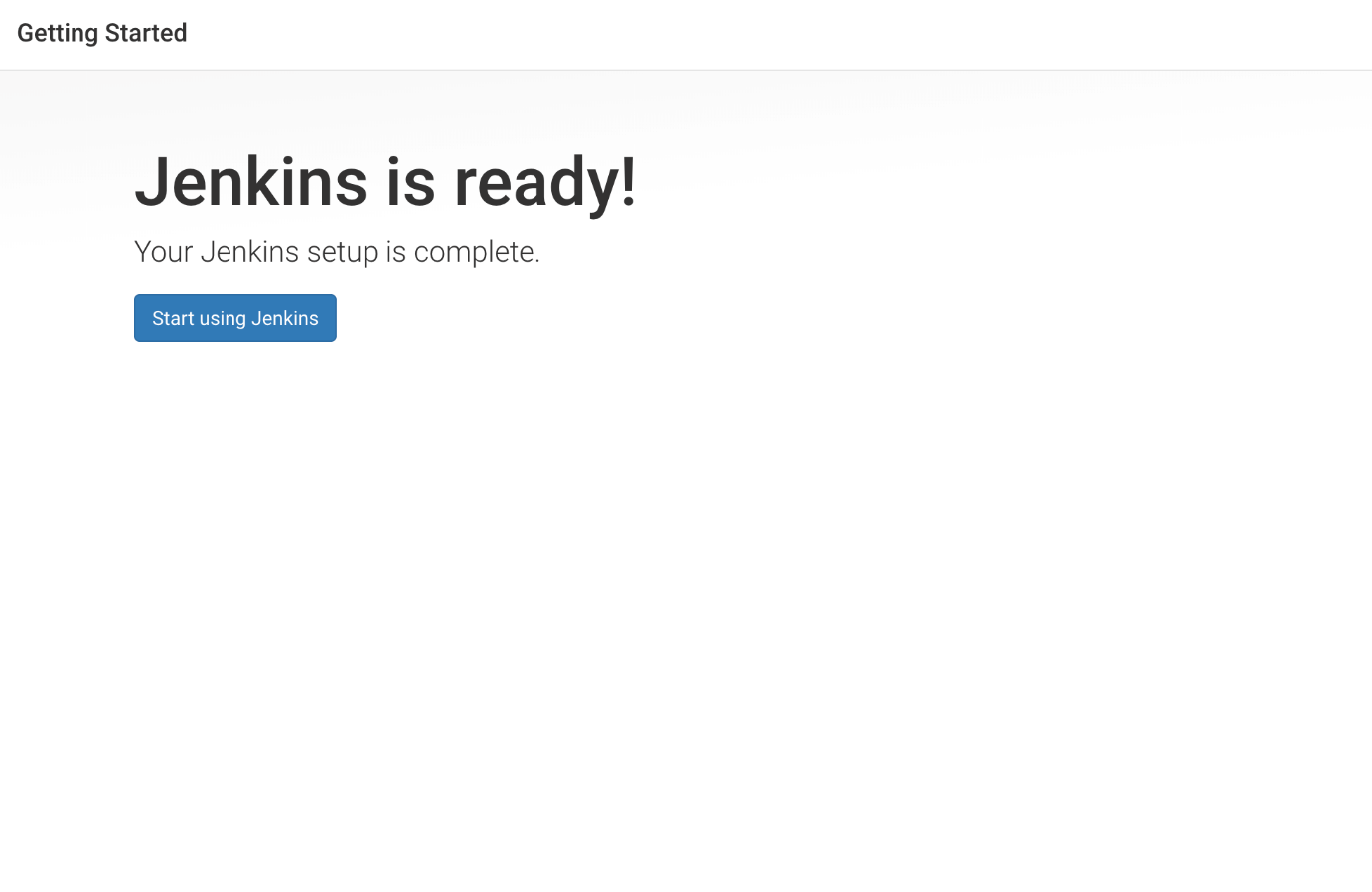


Figure 23: Jenkins - Installation Finish

Click **Start** using **Jenkins** to visit the main Jenkins dashboard:

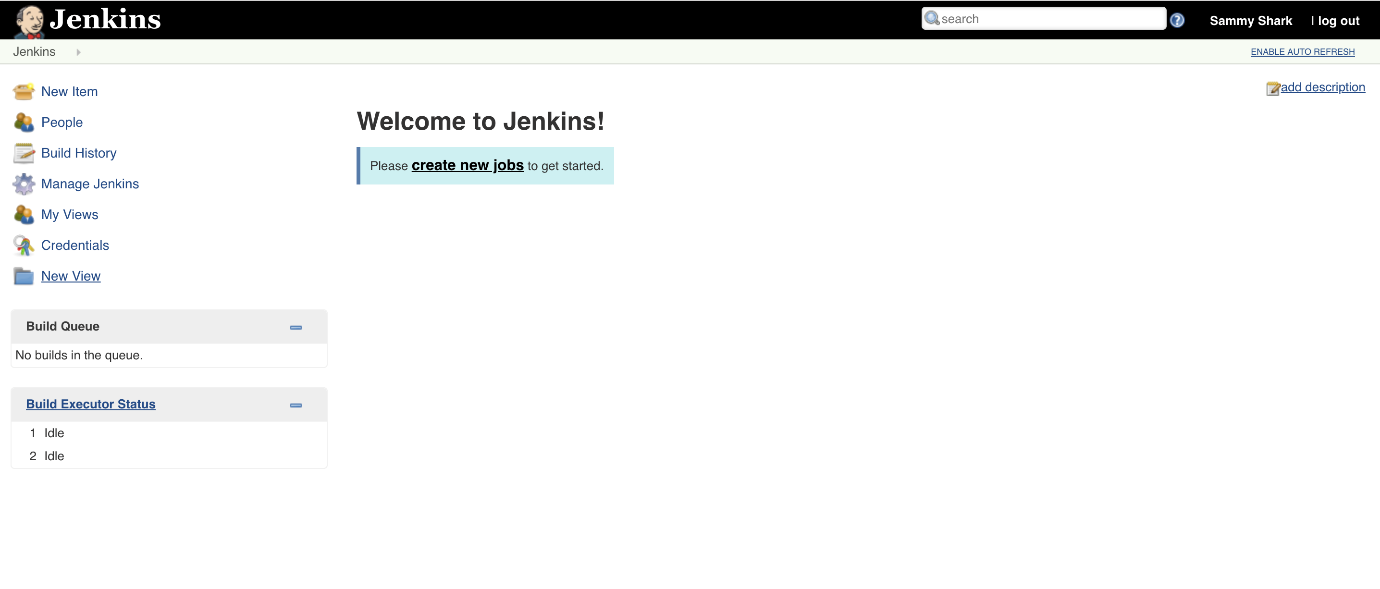


Figure 24: Jenkins - Welcome Page

At this point, you have completed a successful installation of Jenkins.

|  |  |
| --- | --- |
| **TERM** | **DEFINITION** |
| GCP | Google Cloud Platform |
| AWS | Amazon Web Services |
| AWS-CLI | AWS-Command Line Interface |
| SBX | Sand Box Environment |
| VM | Virtual Machine |
| SDK | Software Development Kit |

# Glossary

# Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Date** | **Version** | **Section Changed** | **Description** | **Author** |
| 23 Aug. 2022 | 1.0 | All | Initial Draft | Ashok Das, Tinku Manivikesh Chukkapalli |
|  |  |  |  |  |