

AGENDA

Monday, June 19, 2023 11:50 AM

MAIN AGENDA

1. Automating the build of cloud infrastructure
2. Automating the process of deploying the application and in the created infra resources
3. Deploying in the different environments like (dev, testing, prod)
4. No manual intervention involved

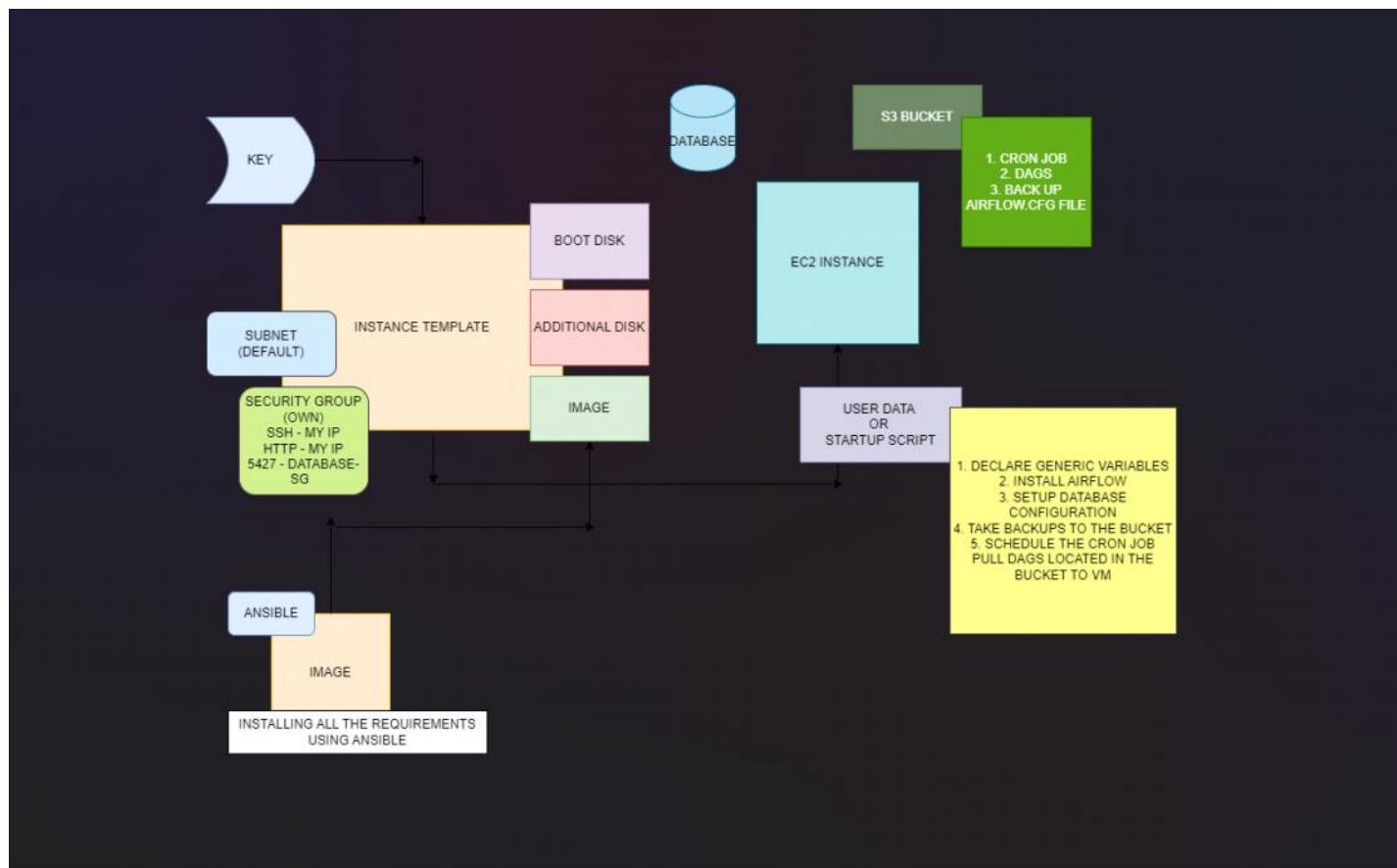
COMPONENTS :

1. AWS CLOUD CONSOLE
 - 1.1 EC2 INSTANCE
 - 1.2 INSTANCE TEMPLATE
 - 1.3 UBUNTU
 - 1.4 REMOTE DATABASE SERVICES
 - 1.5 S3 BUCKET
 - 1.6 LOAD BALANCER
 - 1.7 SECURITY GROUP
 - 1.8 AUTO SCALING
 - 1.9 HEALTH CHECKS
2. JENKINS
3. TERRAFORM
4. GIT
5. ANSIBLE
6. SHELL SCRIPTING

ADVANTAGES OF THIS :

ARCHITECTURE

Tuesday, December 19, 2023 10:20 PM



AWS CLI AUTHENTICATION

Monday, June 19, 2023 2:23 PM

- This is the Documentation steps on how to authenticate AWS CLI in our local machine.
- First, Open AWS Console and login to it as a root user
- Now, we have to create a IAM user, with the required permissions to it. So, whatever permissions are provided, it will allow the specified roles to that IAM user
- Now, Search for IAM > users

The screenshot shows the AWS Identity and Access Management (IAM) dashboard. On the left sidebar, under 'Access management', the 'Users' option is highlighted with a green arrow. The main content area displays the 'IAM dashboard' with sections for 'Security recommendations', 'IAM resources' (User groups: 0, Users: 1, Roles: 4, Policies: 0, Identity providers: 0), and 'What's new'. On the right side, there is an 'AWS Account' summary and a 'Quick Links' section.

- Click on Add Users

The screenshot shows the 'Add users' page in the AWS IAM console. The 'Users' option in the left sidebar is highlighted with a green arrow. The main content area shows a table with one user entry: 'aws-user1'. A green arrow points to the 'Add users' button at the top right of the table.

- Provide the user name and click on NEXT

The screenshot shows the 'Specify user details' step in the 'Create user' wizard. The 'User name' field contains 'aws-user-europeA' and is highlighted with a red box and a red arrow. The 'Next Step' button at the bottom right is also highlighted with a red box and a red arrow.

- Now, select the "Attach policies directly" and check mark the "AdministratorAccess"
- NOTE: You can select the Policy name as per your requirement, If you want to provide instance access only, then select the option "AmazonEC2ContainerRegistryFullAccess"
- Now click on NEXT

→ Step 1: Specify user details

→ Step 2: Set permissions

→ Step 3: Review and create

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

- Add user to group Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- Copy permissions Copy all group memberships, attached managed policies, and inline policies from an existing user.
- Attach policies directly Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (Selected 1/1102)

Choose one or more policies to attach to your new user.

Policy name	Type	Attached entities
<input checked="" type="checkbox"/> AdministratorAccess	AWS managed - job function	1
<input type="checkbox"/> AccessAnalyzerServiceRolePolicy	AWS managed	0
<input type="checkbox"/> AdministratorAccess-Amlnifv	AWS managed	0

1 → Attach policies directly
2 → Selected policy: AdministratorAccess

→ Now, review your policies and click on "CREATE USER"

→ Step 1: Specify user details

→ Step 2: Set permissions

→ Step 3: Review and create

Review and create

Review your choices. After you create the user, you can view and download the autogenerated password, if enabled.

User details

User name aws-user-europe	Console password type None	Require password reset No
------------------------------	-------------------------------	------------------------------

Permissions summary

Name AdministratorAccess	Type AWS managed - job function	Used as Permissions policy
-----------------------------	------------------------------------	-------------------------------

Tags - optional

No tags associated with the resource.

Add new tag
You can add up to 50 more tags.

Cancel Previous Create user

→ Once, user got created, click on that user name ()

→ Identity and Access Management (IAM)

→ Dashboard

→ Access management

→ Users

→ Roles

IAM > Users

Users (2) Info

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

User name	Groups	Last activity	MFA	Password age	Active key age
aws-user-europe	None	Never	None	None	-
aws-user1	None	20 hours ago	None	None	24 days ago

→ Go to SECURITY CREDENTIALS tab > Access Keys > Click on "CREATE ACCESS KEYS"

ARN: arn:aws:iam::157518895600:user/aws-user-europe
Created: June 19, 2023, 14:51 (UTC+09:30)
Console access: Disabled
Last console sign-in: -
Access key 1: Not enabled
Access key 2: Not enabled

Permissions | Groups | Tags | Security credentials | Access Advisor

Console sign-in
Console sign-in URL: https://157518895600.signin.aws.amazon.com/console
Console password: Not enabled
Enable console access

Multi-factor authentication (MFA) (0)
Use MFA to increase the security of your AWS environment. Signing in with MFA requires an authentication code from an MFA device. Each user can have a maximum of 8 MFA devices assigned. Learn more

Device type Identifier Created on
No MFA devices. Assign an MFA device to improve the security of your AWS environment
Assign MFA device

Access keys (0)
Use access keys to send programmatic calls to AWS from the AWS CLI, AWS Tools for PowerShell, AWS SDKs, or direct AWS API calls. You can have a maximum of two access keys (active or inactive) at a time. Learn more
No access keys
As a best practice, avoid using long-term credentials like access keys, instead, use tokens which provide short term credentials. Learn more
Create access key

→ Now, select "COMMAND LINE INTERFACE(CLI)" and (Check understood policies)click on NEXT

Step 1: Access key best practices & alternatives
Step 2 - optional: Set description tag
Step 3: Retrieve access keys

Access key best practices & alternatives
Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and alternatives.

Command Line Interface (CLI)
You plan to use this access key to enable the AWS CLI to access your AWS account.

Local code
You plan to use this access key to enable application code in a local development environment to access your AWS account.

Application running on an AWS compute service
You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.

Third-party service
You plan to use the access key to enable access for a third-party application or service that monitors or manages your AWS resources.

Application running outside AWS
You plan to use this access key to enable an application running on an on-premises host, or to use a local AWS client or third-party AWS plugin.

Other
Your use case is not listed here.

Alternatives recommended

- Use AWS CloudShell, a browser-based CLI, to run commands. Learn more
- Use the AWS CLI V2 and enable authentication through a user in IAM Identity Center. Learn more

I understand the above recommendation and want to proceed to create an access key.

Cancel Next Step

→ If you want to add any tags, add them and click on "CREATE ACCESS KEY"

Step 1: Access key best practices & alternatives
Step 2 - optional: Set description tag
Step 3: Retrieve access keys

Set description tag - optional
The description for this access key will be attached to this user as a tag and shown alongside the access key.

Description tag value
Describe the purpose of this access key and where it will be used. A good description will help you rotate this access key confidently later.
europe-key
Maximum 256 characters. Allowed characters are letters, numbers, spaces representable in UTF-8, and: _ . / = + @

Cancel Previous Create access key

→ Now, it will provide you the ACCESS KEY and SECRET KEY(COPY THEM)

→ Try to download the data in .CSV file by click the "DOWNLOAD .CSV" option. It will save in your local machine.

→ Click on DONE

→ By this, you got an user credentials. So, you can authenticate to aws cloud by using these.

→ The next step is to Authenticate the AWS Cloud in our local machine

→ Open GIT BASH in your computer

→ Run the below command

→ **\$ aws configure**

AWS Access Key ID [*****2GHR]: AKIASJLG*****

AWS Secret Access Key [*****x9DE]: LNKGrifq0L/*****

Default region name [us-east-1]: us-east-1

Default output format [json]: json

→ Enter your "ACCESS KEY", "SECRET KEY" , "REGION" and "OUTPUT FORMAT as : JSON"

→ Now, your authentication in your local machine is completed. So, you can access the aws cloud without any password

→ Run the below command to check you configuration

→ **\$ aws sts get-caller-identity**

```
{
    "UserId": "AIDASJLGXYZYZABCAB",
    "Account": "123445245600",
    "Arn": "arn:aws:iam::123445245600:user/aws-user-europe"
}
```

DATABASE SETUP

Thursday, June 22, 2023 12:08 PM

- Open AWS CONSOLE
- Search for RDS (REMOTE DATABASE SERVICES)

The screenshot shows the Amazon RDS service in the AWS Management Console. The left sidebar includes options like Dashboard, Databases, Query Editor, Performance insights, Snapshots, Exports in Amazon S3, Automated backups, Reserved instances, and Proxies. The main content area displays a message about Aurora I/O-Optimized and a note about creating a Blue/Green deployment. Below these are sections for 'Databases (1)' and 'Actions'. A prominent red box highlights the 'Create database' button at the top right of the main content area.

- Click on "CREATE DATABASE"
- Now, Create a database with the below parameters
- Select "STANDARD CREATE"

The screenshot shows the 'Create database' wizard. The first step, 'Choose a database creation method', has 'Standard create' selected (indicated by a blue circle and a red border). The 'Easy create' option is also shown. A red box highlights the 'Standard create' option. Below this, the 'Engine options' section lists various database engines: Aurora (MySQL Compatible), Aurora (PostgreSQL Compatible), MySQL, MariaDB, PostgreSQL (selected, indicated by a blue circle and a red border), Oracle, and Microsoft SQL Server. A red box highlights the 'PostgreSQL' option.

- Select the Engine type as "POSTGRESQL"

The screenshot shows the 'Create database' wizard. The 'Engine options' step is displayed, showing various engine types: Aurora (MySQL Compatible), Aurora (PostgreSQL Compatible), MySQL, MariaDB, PostgreSQL (selected, indicated by a blue circle and a red border), Oracle, and Microsoft SQL Server. A red box highlights the 'PostgreSQL' option.

- Here, you need to provide "DATABASE INSTANCE NAME", "MASTER USERNAME" and "MASTER PASSWORD"
- I provided the details as below
 - DATABASE INSTANE IDENTIFER : airflow-db
 - MASTER USERNAME : admin123
 - MASTER PASSWORD : admin@123

Services Search [Alt+S]

DB instance identifier [Info](#)
Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

airflow-db

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ Credentials Settings

Master username [Info](#)
Type a login ID for the master user of your DB instance.

admin123

1 to 16 alphanumeric characters. First character must be a letter.

→ Manage master credentials in AWS Secrets Manager
Manage master user credentials in Secrets Manager. RDS can generate a password for you and manage it throughout its lifecycle.

**ⓘ If you manage the master user credentials in Secrets Manager, some RDS features aren't supported.
[Learn more](#)**

Auto generate a password
Amazon RDS can generate a password for you, or you can specify your own password.

Master password [Info](#)

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

- Choose preferred storage, instance configuration, and dbclass as very minimum type (db.t3.micro)
→ Choose PUBLIC ACCESS : YES (or else you can't connect to your database)

Connectivity [Info](#)

Compute resource
Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.

- Don't connect to an EC2 compute resource
Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.
- Connect to an EC2 compute resource
Set up a connection to an EC2 compute resource for this database.

Network type: [Info](#)
To use dual-stack mode, make sure that you associate an IPv6 CIDR block with a subnet in the VPC you specify.

IPv4
Your resources can communicate only over the IPv4 addressing protocol.

Dual-stack mode
Your resources can communicate over IPv4, IPv6, or both.

Virtual private cloud (VPC): [Info](#)
Choose the VPC. The VPC defines the virtual networking environment for this DB instance.

Default VPC: (vpc-04207992bbc14dfs)
6 Subnets, 6 Availability Zones

Only VPCs with a corresponding DB subnet group are listed.

- ⓘ After a database is created, you can't change its VPC.**

DB subnet group: [Info](#)
Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.

default

Public access: [Info](#)
 Yes
RDS assigns a public IP address to the database. Amazon EC2 instances and other resources outside of the VPC can connect to your database. Resources inside the VPC can also connect to the database. Choose one or more VPC security groups that specify which resources can connect to the database.

No
RDS doesn't assign a public IP address to the database. Only Amazon EC2 instances and other resources inside the VPC can connect to your database. Choose one or more VPC security groups that specify which resources can connect to the database.

VPC security group (firewall): [Info](#)
Choose one or more VPC security groups to allow access to your database. Make sure that the security group rules allow the [Inbound traffic](#).

- Here, I'm creating my own SECURITY GROUP and providing my own rules to the database
→ So, select "CREATE NEW" and provide a name to your new SG : "airflow-db-sg"

VPC security group (firewall) [Info](#)

Choose one or more VPC security groups to allow access to your database. Make sure that the security group rules allow the appropriate incoming traffic.

<input type="radio"/> Choose existing Choose existing VPC security groups	<input checked="" type="radio"/> Create new Create new VPC security group
New VPC security group name → airflow-db-sg	

Availability Zone [Info](#)

No preference

RDS Proxy

RDS Proxy is a fully managed, highly available database proxy that improves application scalability, resiliency, and security.

Create an RDS Proxy [Info](#)

RDS automatically creates an IAM role and a Secrets Manager secret for the proxy. RDS Proxy has additional costs. For more information see [Amazon RDS Proxy pricing](#)

→ Now, set up the rule like below :

→ You can also set the rules within your IP as well

- TYPE : POSTGRESQL
- PROTOCOL : TCP
- PORT RANGE : 5432
- SOURCE : ANYWHERE (OR) MYIP

EC2 > Security Groups > sg-08e2223c3f76c2b11 - airflow-db-sg > Edit inbound rules

Edit inbound rules [Info](#)

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules Info					
Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info
sgr-0f0bc803889e34e4f	PostgreSQL	TCP	5432	Custom	<input type="text" value="0.0.0.0"/> X
<input type="button" value="Add rule"/> Cancel Preview changes Save rules					

→ Click on "SAVE RULES"

→ Select the DATABASE AUTHENTICATION as : "PASSWORD AUTHENTICATION"

Database authentication

Database authentication options [Info](#)

Password authentication

Authenticates using database passwords.

→ Password and IAM database authentication

Authenticates using the database password and user credentials through AWS IAM users and roles.

Password and Kerberos authentication

Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

→ Keep the other options as default and click on "CREATE DATABASE"

→ Once, the DB got created, you have to COPY the ENDPOINT URL and the port number of the db

- Now, let's try to connect with the PGADMIN
- (INSTALL PGADMIN in your computer)
- Create a password for PGADMIN (you have to enter the password everytime, when you entering into the server)
- MY PGADMIN PASSWORD : kishandev
-
- In PGADMIN, click on "ADD NEW SERVER" :

The screenshot shows the pgAdmin interface. At the top, there is a navigation bar with tabs: Dashboard, Properties, SQL, Statistics, Dependencies, Dependents, and Processes. Below the navigation bar, the title 'pgAdmin' is displayed with the subtitle 'Management Tools for PostgreSQL'. A sub-header 'Feature rich | Maximises PostgreSQL | Open Source' follows. A descriptive text block states: 'pgAdmin is an Open Source administration and management tool for the PostgreSQL database. It includes a graphical administration interface, an SQL query tool, a procedural code debugger and much more. The tool is designed to answer the needs of developers, DBAs and system administrators alike.' Under the 'Quick Links' section, there is a button labeled 'Add New Server' which is highlighted with a red box. To its right is another button labeled 'Configure pgAdmin'. Below this, under the 'Getting Started' section, there are links to 'PostgreSQL Documentation', 'pgAdmin Website', 'Planet PostgreSQL', and 'Community Support'.

- AF
- Give "HOSTNAME", "USERNAME" and "PASSWORD" which we provided in the AWS database

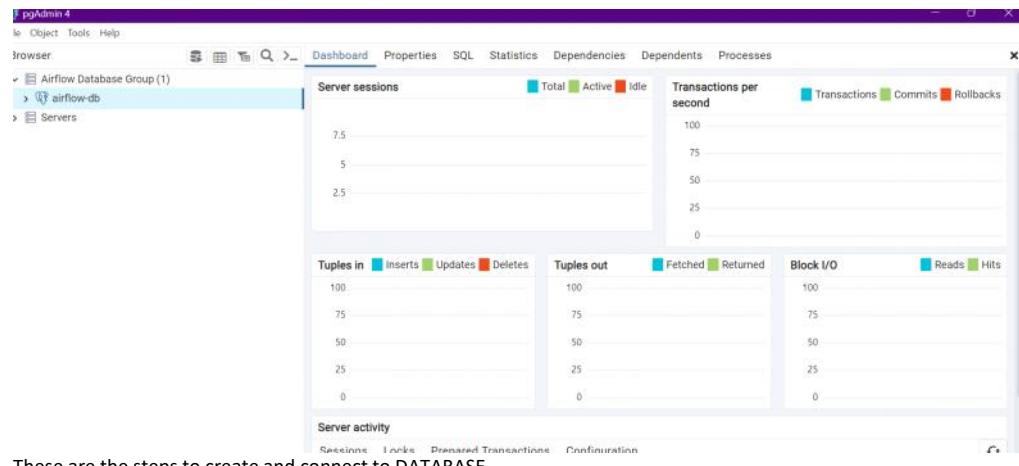
The screenshot shows the 'Register - Server' dialog box. The 'Connection' tab is selected. The form fields are as follows:

Host name/address	airflow-db.cyn1tccpnjs5.us-east-1.rds.amazonaws.com
Port	5432
Maintenance database	postgres
Username	admin123
Kerberos authentication?	<input checked="" type="checkbox"/>
Password
Save password?	<input checked="" type="checkbox"/>
Role	
Service	

At the bottom of the dialog box, there are three buttons: 'Close', 'Reset', and a blue 'Save' button.

HOST NAME : airflow-db.cyn1tccpnjs5.us-east-1.rds.amazonaws.com
 USERNAME : admin123
 Password : admin123

- Click on save
- It will connect to your database :



→ These are the steps to create and connect to DATABASE

STANDARD SETUP OF JENKINS

Monday, June 19, 2023 12:04 PM

JENKINS :

Jenkins is a Java-based open-source automation platform with plugins designed for continuous integration. It is used to continually create and test software projects, making it easier for developers and Devops Engineers to integrate changes to the project and for consumers to get a new build. It also enables you to release your software continuously by interacting with various testing and deployment methods

From <<https://www.spiceworks.com/tech/devops/articles/what-is-jenkins/>>

1. This is a Continuous Integration/ Continuous Delivery (CI/CD) tool.
2. Deploying code into production
3. Enabling task automation
4. Reducing the time it takes to review a code
5. Driving continuous integration

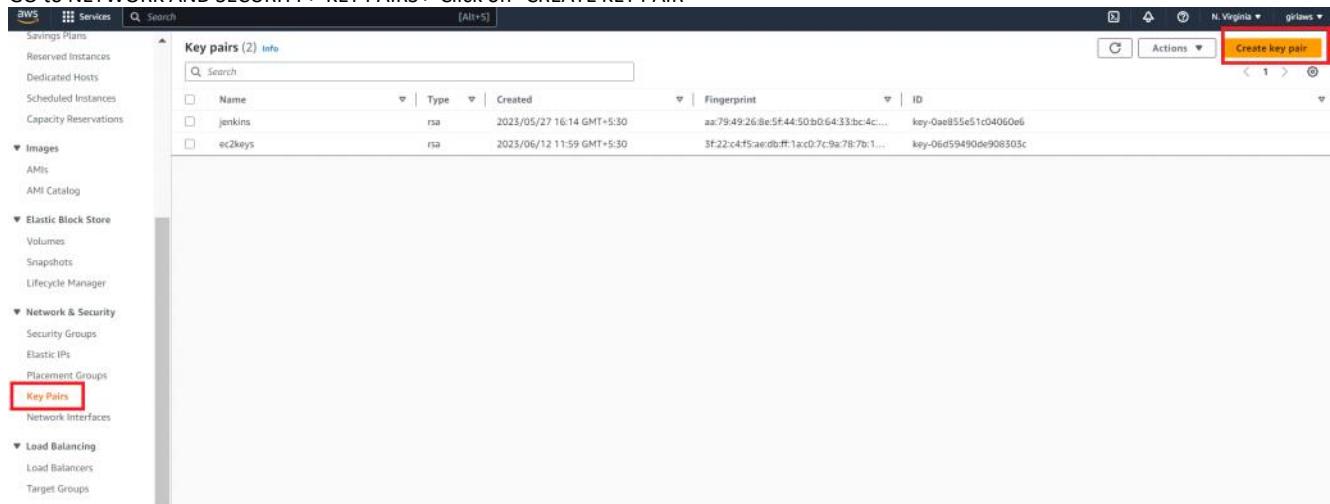
HOW TO SETUP A JENKINS SERVER IN EC2 INSTANCE

→ REQUIREMENTS :

- EC2 INSTANCE
- SECURITY GROUP
- KEY-PAIR
- JAVA 11
- JENKINS INSTALLATION SCRIPT

→ CREATING THE KEY-PAIR (MANUALLY):

→ GO to NETWORK AND SECURITY > KEY PAIRS > Click on "CREATE KEY PAIR"



→ PROVIDE THE FOLLOWING DETAILS :

- NAME : jenkins
- KEY PAIR TYPE : RSA
- PRIVATE KEY FORMAT : .PEM

Create key pair Info

Key pair

A key pair, consisting of a private key and a public key, is a set of security credentials that you use to prove your identity when connecting to an instance.

Name

jenkins

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type Info RSA ED25519

Private key file format

 .pem

For use with OpenSSH

 .ppk

For use with PuTTY

Tags - optional

No tags associated with the resource.

[Add new tag](#)

You can add up to 50 more tags.

Cancel

[Create key pair](#)

- Click on CREATE KEY PAIR. It will create a key-pair and it will download a .pem file to your local machine. Based on the .pem file you can able to SSH into the VM

→ **CREATING THE SECURITY GROUP (MANUALLY):**

- Go to NETWORK AND SECURITY > CREATE SECURITY GROUP

Name	Security group ID	Security group name	VPC ID	Description	Owner	Inbound rules count	Outbound rules co...
sg-08e223c3f76c2b11	vpc-042b7992bbcb14df5	airflow-db-sg	vpc-042b7992bbcb14df5	Created by RDS manag...	157518895600	1 Permission entry	1 Permission entry
sg-028425f170948908c	vpc-042b7992bbcb14df5	launch-wizard-1	vpc-042b7992bbcb14df5	launch-wizard-1 create...	157518895600	1 Permission entry	1 Permission entry
sg-099e6429b5dd90808	vpc-042b7992bbcb14df5	manual-instances-sg	vpc-042b7992bbcb14df5	This is for the instance...	157518895600	3 Permission entries	1 Permission entry
sg-08c191d0590bc4e51	vpc-042b7992bbcb14df5	launch-wizard-2	vpc-042b7992bbcb14df5	launch-wizard-2 create...	157518895600	3 Permission entries	1 Permission entry
sg-0e4e4a21b2948efbe	vpc-042b7992bbcb14df5	default	vpc-042b7992bbcb14df5	default VPC security gr...	157518895600	1 Permission entry	1 Permission entry

→ PROVIDE THE DETAILS :

- SECURITY GROUP NAME : manual-instances-sg
- DESCRIPTION : This SG is used for manual created instance
- VPC : <KEEP DEFAULT VPC ONLY>

EC2 > Security Groups > Create security group

Create security group Info

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To create a new security group, complete the fields below.

Basic details

Security group name Info
manual-instances-sg
Name cannot be edited after creation.

Description Info
This SG is used for manual created instance

VPC Info
vpc-042b7992bbcb14df5

Inbound rules Info

This security group has no inbound rules.

Add rule

→ ADD THE BELOW RULES :

- SSH-PORTRANGE:22-ANYWHERE
- SSH-PORTRANGE:22-MYIP
- CUSTOMTCP(OR)-PORTRANGE:8080-MYIP

EC2 > Security Groups > sg-099e6429b5dd90808 - manual-instances-sg > Edit inbound rules

Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules <small>Info</small>		Type <small>Info</small>	Protocol <small>Info</small>	Port range <small>Info</small>	Source <small>Info</small>	Description - optional <small>Info</small>	
→	sgr-06ab785cb460435f1	SSH	TCP	22	Custom	0.0.0.0/0 X	Delete
→	sgr-026ecf8714b6cc557	SSH	TCP	22	Custom	49.37.134.186/32 X	Delete
→	sgr-0fabf2bd5eab6b375	Custom TCP	TCP	8080	Custom	49.37.134.186/32 X	Delete

Add rule

→ Click on "CREATE SECURITY GROUP". It will create the Security Group (FIREWALL)

→ CREATING AN EC2 INSTANCE (MANUALLY) :

- First, Login to AWS CONSOLE
- Go to EC2 SECTION

New EC2 Experience Tell us what you think

Instances (3) Info

Find instance by attribute or tag (case-sensitive)

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...	Elastic IP
jenkins-test	i-0e0690dbeaa05c720	Stopped	t2.small	-	No alarms +	us-east-1b	-	-	-
Ansible-test2	i-001e080221599eba2	Stopped	t2.micro	-	No alarms +	us-east-1b	-	-	-
Ansible-test	i-06992e9f80810c213	Stopped	t2.micro	-	No alarms +	us-east-1a	-	-	-

Launch instances

→ In Instances -> CLICK on create in LAUNCH INSTANCE

- Provide the below parameters :

 - NAME : Jenkins-server
 - Image : UBUNTU
 - Number of instances : 1

Launch an instance

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags

Name: Jenkins-server

Application and OS Images (Amazon Machine Image)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below.

Search our full catalog including 1000s of application and OS images

Quick Start

Amazon Linux, macOS, Ubuntu, Windows, Red Hat, S...

Ubuntu (selected)

Browse more AMIs Including AMIs from AWS, Marketplace and the Community

Summary

Number of instances: 1

Software Image (AMI): Canonical, Ubuntu, 22.04 LTS, ami-053b0d53c279acc90

Virtual server type (instance type): t2.micro

Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Cancel Launch instance Review commands

- INSTANCE TYPE : t2.micro
- NOTE: JENKINS HAS MINIMUM REQUIREMENT OF HAVING INSTANCE TYPE : t2.small
- KEYPAIR : jenkins

Instance type

Instance type: t2.micro (selected)

Family: t2: 1 vCPU 1 GiB Memory Current generation: true
On-Demand Windows pricing: 0.0162 USD per Hour
On-Demand SUSE pricing: 0.0116 USD per Hour
On-Demand RHEL pricing: 0.0716 USD per Hour
On-Demand Linux pricing: 0.0116 USD per Hour

Free tier eligible

All generations

Compare instance types

Key pair (login)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required: jenkins

Create new key pair

Network settings

Network: manual-instances-sg (selected)

Summary

Software Image (AMI): Canonical, Ubuntu, 22.04 LTS, ami-053b0d53c279acc90

Virtual server type (instance type): t2.micro

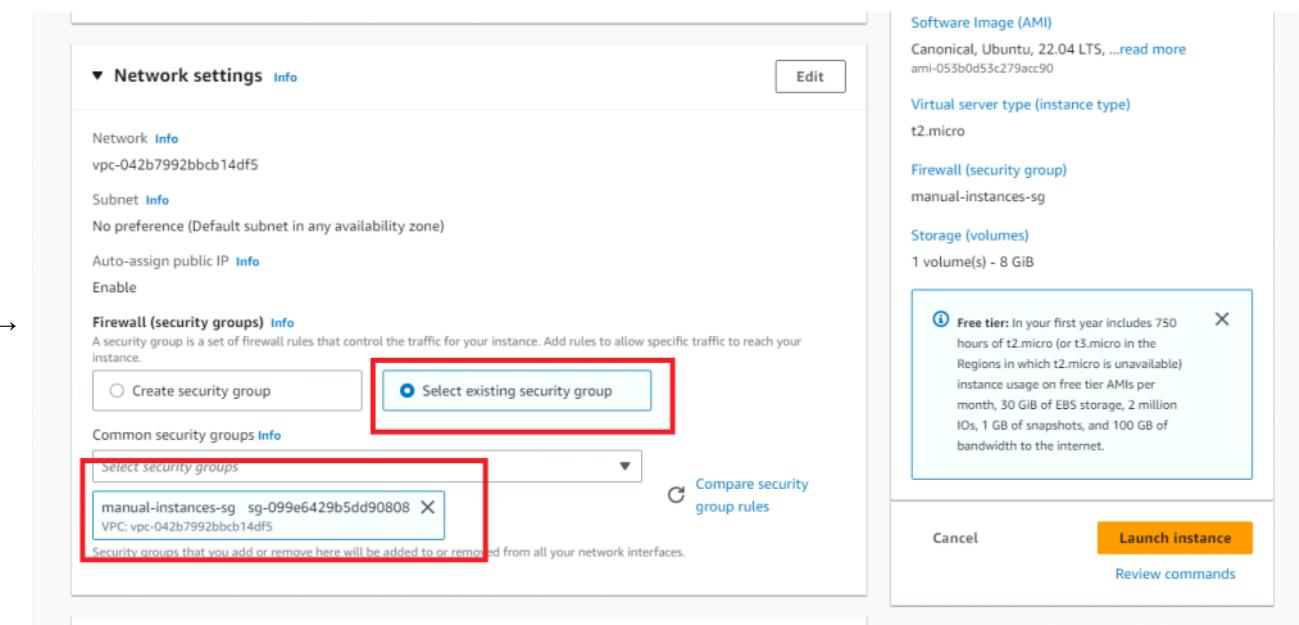
Firewall (security group): New security group

Storage (volumes): 1 volume(s) - 8 GiB

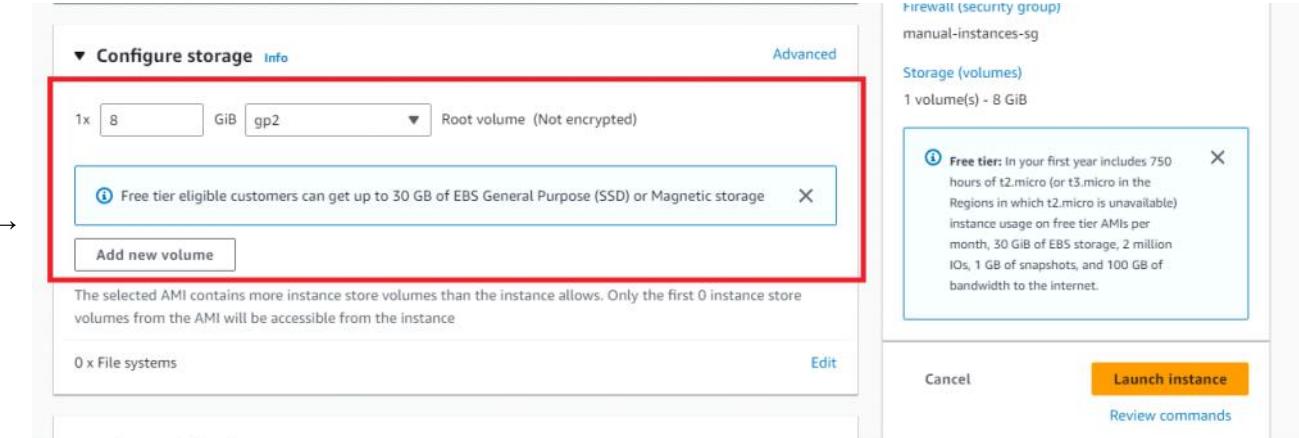
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Cancel Launch instance Review commands

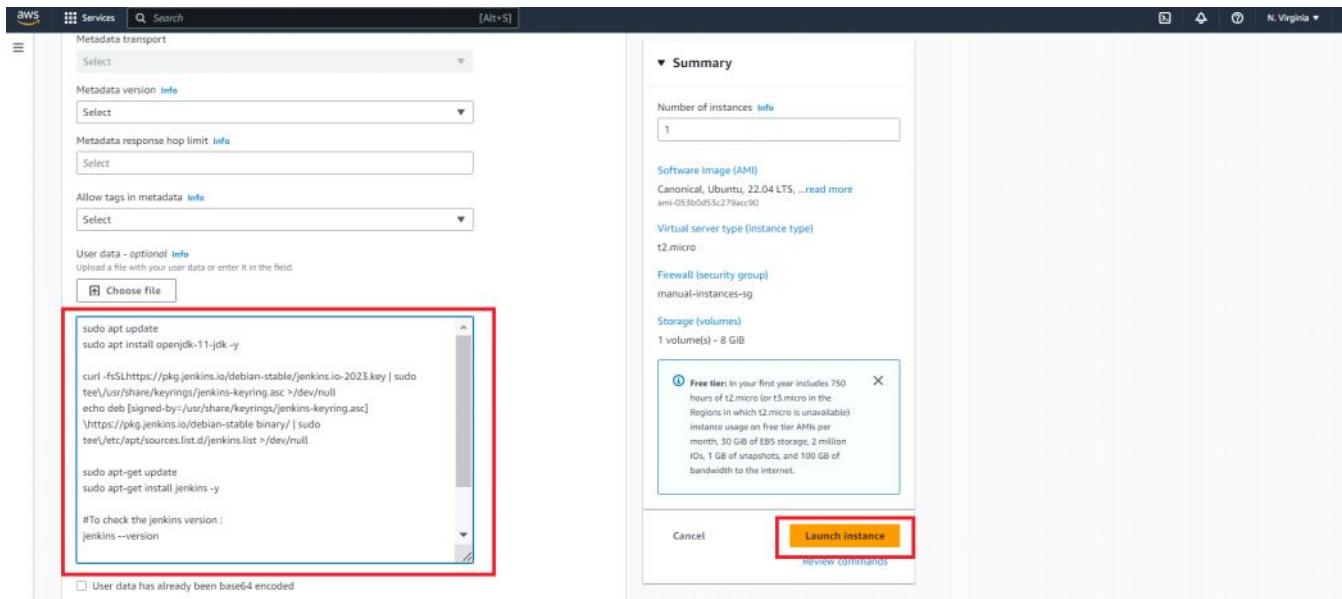
- SECURITY GROUP : (FIREWALL) - SELECT EXISTING SECURITY GROUP
- CHOOSE : "manual-instances-sg" (we created earlier)



- The STORAGE : 8GB



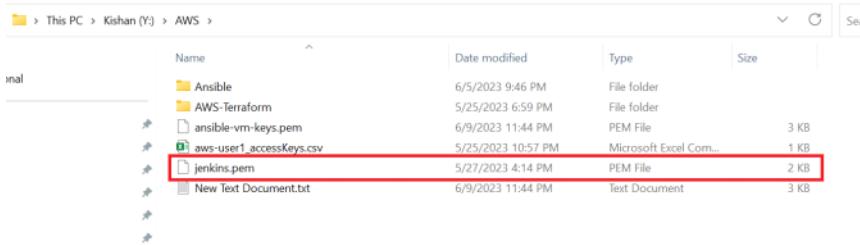
- CONFIGURING THE JENKINS SETUP INSIDE THE EC2 VM (MANUALLY)
- There are 2 ways to configure the script :
 - THROUGH USER DATA
 - VIA SSH INTO VM AND SETTING UP THE JENKINS INSTALLATION VIA COMMANDS
- **METHOD : 1 - THROUGH USER DATA**
- **In USER DATA :**
- **ADD THE INSTALLATION SCRIPT and click on "LAUNCH INSTANCE"**



→ Once, the instance got created, the script in the user data will automatically run the instance

→ **METHOD: 2 - VIA SSH INTO VM AND SETTING UP THE JENKINS INSTALLATION VIA COMMANDS**

- Login to the VM via SSH
- Copy the VM "PUBLIC IP" : 3.80.89.180
- Go to the path where your jenkins.pem file located



→

- Open GIT BASH in this path and run the below command
- `$ ssh -i jenkins.pem ubuntu@3.80.89.180`



- Click on ENTER, it will ask for fingerprint: <TYPE : YES>
- Now, you'll be able to login to the VM
- **NOTE: IF YOU UNABLE TO LOGIN TO THE VM, THEN CHECK THE SECURITY GROUP RULES. MOSTLY YOUR LOCAL IP CHANGES EVERYTIME. SO, SET IT AGAIN.**
- Now, Run the commands one after one with (#ROOT USER - `$sudo su`) and it will install the jenkins in your VM

→ **JENKINS SCRIPT :**

```
#THIS COMMAND IS FOR REGULAR UPDATES FOR UBUNTU
sudo apt update

#JENKINS REQUIRED THE JAVA-11 VERSION. THIS COMMAND WILL INSTALL JAVA-11
```

```
(JDK-11)
sudo apt install openjdk-11-jdk -y

#THIS WILL SET UP THE JENKINS CONFIGURATIONS INSIDE THE VM
curl -fsSLhttps://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee
/usr/share/keyrings/jenkins-keyring.asc >/dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-
stable binary/ | sudo tee/etc/apt/sources.list.d/jenkins.list >/dev/null

#AGAIN RUNNING THIS COMMAND FOR REGULAR UPDATES
sudo apt-get update

#THIS WILL INSTALL JENKINS IN YOUR VM
sudo apt-get install jenkins -y

#To check the jenkins version :
jenkins --version

#THESE COMMANDS WILL ENABLE THE JENKINS SERVER
systemctl start jenkins
systemctl enable jenkins
systemctl status jenkins
```

- JENKINS will run in the port: 8080
- Now, copy the Public IP and search <http://<public-ip>:8080>
- <http://44.204.172.179:8080/>
- <http://44.204.172.179:8080/>
- Start setting up the jenkins with standard default plugins only
- Create the user and password for it



- REFERENCE LINKS :
- <https://www.jenkins.io/doc/book/installing/linux/>
- <https://www.jenkins.io/doc/book/installing/linux/#debianubuntu>
- [How to Setup & Install Jenkins in AWS EC2 Ubuntu Instance | DevOps](#)

JENKINS INTEGRATING WITH GIT

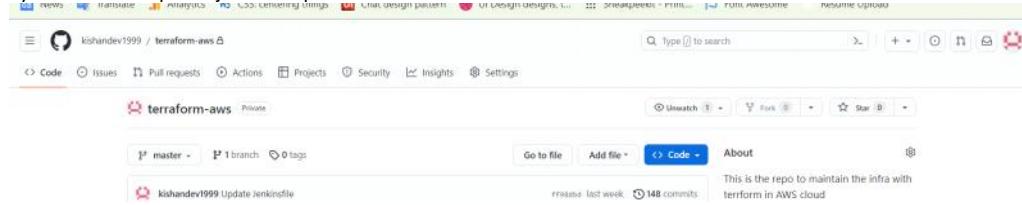
Sunday, June 25, 2023 5:23 PM

AUTHENTICATING GIT TO JENKINS WITH SSH KEYS :

→ REQUIREMENTS :

- GIT
 - GIT REPO (PRIVATE MODE)
 - SSH KEYS
 - PERSONAL ACCESS TOKEN (FROM GIT)
 - JENKINS SERVER

→ Create your own GIT repo for jenkins in private mode



→ GENERATING SSH KEYS :

→ Open git bash in your local computer machine (not in VM) and generate the ssh keys by using below command

→ \$ssh-keygen

- If the keys are already exists, then it will ask to overwrite the file or else, it will create the new key pairs

```
usus@KishanDev MINGW64 /y/AWS
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/y/Devops Udemy/ssh-keys-folder/.ssh/id_rsa):
/y/Devops Udemy/ssh-keys-folder/.ssh/id_rsa already exists.
```

→ Go to the path and check your keys

```
asus@KishanDev MINGW64 ~/ssh
$ ls
id_rsa  id_rsa.pub  known_hosts  known_hosts.old
```

→ You'll find the keys : "id_rsa", "id_rsa.pub", "known_hosts"

→ Id_rsa.pub : This is the public key. Very short key

```
$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAgQCiugC91d3DJRt6Fyo+9gs4+dp+7Ex8pZSz7Jdi0S
rLrxrbWTFgtRvZnURzTnKLDbd4t6wdI6TnKzQVwPMGej6FcSa086ATHwxnSKOPMkmp
I4Vm5gtJ0fK9tNL6DfpbeoE2D5ViCnghqAv705ha2rEogNIq+PahVg0di5CnLk1MwPiYnb
• GqzcSp41yRU4IpJegix7goySh7GqIkv9XgqTxz94nUkx0tCunrh4djks07bc9E1YtzAZvgjRNArB
cMu2P3b8MEBVrangYsuxXgP8C2Z5jSD3nAndwKt3oTM+kzqjE3t4W0g8CII6he+AElqwZAUsZMpkvd
3G162ThIHRchmUC6CLPSM466PEPaw1bEqdopYxfFeG+OFw1NgJrqRoFt2EHz/PbxRNTpYR8rJiJ5oR7
XuwnGiH86to+khpG/B/ygIwbYugaNwBDxSB9g9dgkFB1QKOSZw+0VgcplieA7yAkDaIsucuk= asus@shanDev
```

→ Id_rsa : This is private key. Very long key

```

asus@KishanDev MINGW64 ~/.
$ cat id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3B1bnNzaC1rZXktdjEAAAAABG5vbmuAAAEBm9uZQAAAAAAAAABAABlw
NhAAAAAwEAQAAAYEAoroAvZXdwyUbehcPvYLOPnafxMfKwUs+yXYtEk
YIrUb8wAOEcLVEWr5yiw23qbeHsHeiE55GUFcD1TBhI+hxEgDvOgE4VsZ
OL5oxrSdH9CvbTs+gxaw3qBng+VYyDYL+zuYwtq6XXIDSKvjz0WFYD
BT4mJ23cNBqs3EgeNckVOCKSXoIse4MKoexqjL/Vxqk18/ejylJF6E3F
3PRNWLcwGb4I0TQKwU23DLtj92/DBAVa2p4GLM1MRqfAtmeY0g95wDXCCr
+FtBqPAiCooXvgBK1m0ErmtKZFXyaNxpetk4sB0oQzAugizjooujxdwM
hvjhctNTYCUakaH09hb8/z28TU6WEfIyCeaEex3V7sJxoh/OraPpiYaF2
8UgfYPXYJBQYkCjks1vjlYHKYng08gJA2iLLnArpaAAFiGch5G1nIeRtAA
EAAAGBAKK6AL2V3cMlg3oXkj72Czj52n7sTHy1lLps12LRJLSSuvEftJSV
C1RFm0ecosNt0G3h7B3oh0eR1BXA9UwYSpovXIA7zoBOFBGdIIo48ySan6
r20voMV21t6gTPlwInw2GAC/s7mFrauss1A0ir49qFhwA52LkkcgutUw
rNxKnjxJFTgik16CLHucjJKHsaois/1capNfp3icpsRehnxsEuHh2N2Syj
CNE0CsFNtwy7Y/dvwvFWtqeBizNTEanwLznmNIPecA13Aq3ehMz6T0qMT
F74ASpZk8k5kymRv2GjcaXrZOEgdeKGZoLoIs9Izjro808DCVsRB2i1hd8
pgh4PYQfP89vFE101hHyMgnmhhsd1e7Ccaifzq2j6SGmA9iAjBt16Bo1Y
GJAo5LnB45WBym24DV1CQNoi5Wk6QAAAAMBAEAAAGAcj/tDAQEk+Yido
• ARUUiectTiRO1eb6EV5APV7CmDeYVZdsyuLuXAZ5pt70ubq2aRxHHq+hkP
1p3Iysth1lnuqntbMvk9w1GRUEX7s2pdFz1s2sYpA9cDKoDh3KxCQoDz1
SOBr1Ig/Mer3IjiCjiu6ScBd2o9CuV+6XP2PhVKwsILBswwrSbWGfQ8Ht
BrtzzB+5xzcs7bPEsbyP+Sj0ywAOKwfrrsN9PyelPS06Tu1c/fo8KtgY
einzt6/hmaEeat/HpkQwxXwfs1Ca+YOIOQ0TMDE3geuYJfxGw/CVw/Zco21
+pCoZbg5VdLhAeFHwBekPgT4C1jff4xBixyRTB/qcK5b8u35iduz1LZXZ
bookmbITkz0XtdU7xMi2TsV7G9vPVDwai1kCOFFcX7YXaJ6AwjkWkZz/OU
wBR4sWMIQbnv5y/ZVMakktzHG5UrfcK7mYloAtpPsPEWqsKwUUsvUNzG
UAiMGo0bKFcy3fl8LeUr6/Nqu2iu7baFB/vhFTM1Z1NaGoIZXroj3Wdq
prunY69UEscZxh1ENGuahqQqmwy4uqGCI6ig5mTk cwdHyyTo4IyguLZ+Nr
cVPqMXZQL76dzS6ac91/ek+Tp6eYgjajahNOEHxbnImvvAAAMEAzadG
MeeEXYLuToothz7XbMyNeHzfOn5yFOC2KBcmWpVfToW+1BDAPJqzmA1
3ZFrVaBsre7lymbYr7EzI/NvtzfcKzTQuBsdHbmBxJfhWopFzwz+xFrq
41tpwMgcT95yA3VQixBHEtzcyUuwWq8SDAwz2dTGOBh4a2RYdq0Zgi5eo
rDfetw+H5/6ypvr+/9HnP6kwfM1j5AAAawQDKKGMVvEs491HPtZLJxhsx
grdB8/55OKBHCs9C8wv0ot2kWdyXU2XTSxhZLr4CKBIGT27ce4yUGVC+Eh
zQQnzzAsRNHufqS3hSXEdIN8DG97/16B7biis88VN5cfeqzsLIwSxn0qHQ
JN1U1QEn5wTxkSnztGrEmF17XDKzo93PGXSFxjy1wUe0xEUFAY0Xnc5Pn
+PQJrY/htn3/EAAAQYXN1c0BLaxNoYw5EZXYBAgMEBQ==

-----END OPENSSH PRIVATE KEY-----

```

→ Known-hosts will store the hosting ip's and login data

POINTS TO REMEMBER :

1. Why does the public and private key change for a user in the same machine?
2. In continuation of that question, why can't I use my old public and private key to access a repository when nothing else has changed?
3. Will the keys change if I create them again using ssh-keygen -t rsa?

ANSWERS :

1. A key pair is not tied to you as a user or your machine. Imagine having the keys to your house stolen. The keys aren't bound to you, anyone can use them to unlock your doors. In the exact same way, if I get my hands on your private key, I can use it to make changes to your Github repositories.
2. If you wish to access your repositories using both your new keys and your old ones, you need to make sure the *public* keys for *both* key pairs are uploaded to Github. If Github only has your new key, it cannot authenticate you if you use your old key pair.
3. Yes, the keys will change. That's sort of the point. Imagine buying a new lock for your door. If your old keys are the same as your new ones, it won't do you any good, esp. if you were changing the lock because someone stole your old key. If you were to have one of your ssh keys compromised, you could simply generate new keys & upload the corresponding public keys to Github after deleting the old ones. You can also have multiple key pairs. Generate as many as you wish and use them for different purposes. This is similar to having multiple keys for multiple locks in your house.
Also NEVER EVER give away your private keys. The private key is what is used to authenticate you as a person. If I get my hands on your private key, I can fake being you. Github can't tell the difference. No one but you needs to know what your private key is. If you have uploaded your private key somewhere, I recommend deleting it, generating a new pair & uploading the new public key just to be safe.

From <<https://unix.stackexchange.com/questions/121311/does-public-and-private-key-remain-the-same-for-a-user-or-system>>

→ ADDING THE LOCAL MACHINE KEYS TO GIT REPO :

- Now go to git hub > settings > SSH AND GPG KEYS >
- Click on the NEWSSHKEY and add your "PUBLIC KEY" over there :

The screenshot shows the GitHub Settings page for the user Pithani.Kishan Dev (kishandev1999). The left sidebar includes links for Public profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, and SSH and GPG keys (which is highlighted with a red box). The main content area displays the SSH keys section, which lists one key: Jenkins-key (SHA256:vyBjJp9mt1bzEd4ZI3kq0WEFw+ImBy51goh54os). It shows the key was added on Jun 6, 2023, and last used within the last 2 weeks. A 'Delete' button is available for this key. Below this, there's a link to a guide for generating SSH keys or troubleshooting common SSH problems. The right sidebar features a 'Vigilant mode' section.

- Give the TITLE
 - KEY TYPE : <AUTHENTICATION KEY>
 - KEY : <ADD YOUR PUBLIC KEY> id_rsa.pub (short key)
 - Click on ADD SSH KEY

The screenshot shows the GitHub Settings interface. On the left, there's a sidebar with sections like 'Settings', 'Public profile', 'Account', 'Appearance', 'Accessibility', 'Notifications', 'Access' (with 'Billing and plans', 'Emails', 'Password and authentication', 'Sessions', 'SSH and GPG keys' selected), 'Organizations', 'Moderation', and 'Code, planning, and automation' (with 'Repositories' selected). The main area is titled 'SSH keys / Add new'. It has fields for 'Title' (set to 'Jenkins-key') and 'Key type' (set to 'Authentication Key'). A large text area is labeled '<ADD YOUR PUBLIC KEY HERE>' with a sample key value 'ssh-rsa AAAQABAAQ...' visible. At the bottom is a blue button labeled 'Add SSH key'.

→ GIT INSTALLATION IN JENKINS SERVER VIA JENKINS CONSOLE :

- Login to jenkins with created username and password
 - Firstly, we need to seed install the GIT in Jenkins
 - Go to DASHBOARD > MANAGE JENKINS > TOOLS

The screenshot shows the Jenkins Manage Jenkins interface. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins' (which is highlighted with a red box), and 'My Views'. Below that are sections for 'Build Queue' (empty) and 'Build Executor Status' (1 idle, 2 idle). The main content area has a title 'Manage Jenkins' and a message about a reverse proxy setup being broken. It includes a link to documentation for setting up distributed builds. Under 'System Configuration', there are four boxes: 'System' (Configure global settings and paths), 'Tools' (Configure tools, their locations and automatic installers, also highlighted with a red box), 'Plugins' (Add, remove, disable or enable plugins that can extend the functionality of Jenkins), and 'Nodes and Clouds' (Add, remove, control and monitor the various nodes that Jenkins runs jobs on). At the top right, there are buttons for 'More Info', 'Set up agent', 'Set up cloud', and 'Delete'.

- There setup the GIT as below
 - This will install GIT automatically into your server

Git installations

The screenshot shows the Jenkins 'Git installations' configuration page. A new item named 'git-tool' is being added. The 'Path to Git executable' field contains 'git'. The 'Install automatically' checkbox is unchecked. There is a button labeled 'Add Git' at the bottom left.

→ PLUGINS :

- Go to Jenkins console
- We need to install the required plugins in GIT for JENKINS
- In Jenkins DASHBOARD > Go to manage JENKINS



+ New Item

People

Build History

Project Relationship

Check File Fingerprint

Manage Jenkins

My Views

→ Go to PLUGINS

The screenshot shows the Jenkins 'Manage Jenkins' page. The 'Plugins' section is highlighted with a red box. It displays a warning about building on the built-in node being a security issue and provides links to documentation, setup agent, setup cloud, and dismiss. The 'System Configuration' section includes links for System, Tools, and Nodes and Clouds.

→ Install the below plugins :

Name	Enabled
Git client plugin 4.4.0 Utility plugin for Git support in Jenkins. Report an issue with this plugin	<input checked="" type="checkbox"/> 4.4.0
Git plugin 5.1.0 This plugin integrates Git with Jenkins. Report an issue with this plugin	<input checked="" type="checkbox"/> 5.1.0
GitHub API Plugin 1.314-431.v78d72a_3fe4c3 This plugin provides GitHub API for other plugins. Report an issue with this plugin	<input checked="" type="checkbox"/> 1.314-431.v78d72a_3fe4c3
GitHub Branch Source Plugin 1728.v859147241f49 Multibranch projects and organization folders from GitHub. Maintained by CloudBees, Inc. Report an issue with this plugin	<input checked="" type="checkbox"/> 1728.v859147241f49
GitHub plugin 1.37.1 This plugin integrates GitHub to Jenkins. Report an issue with this plugin	<input checked="" type="checkbox"/> 1.37.1
Pipeline: GitHub Groovy Libraries 42.v0739460cda_c4 Allows Pipeline Groovy libraries to be loaded on the fly from GitHub. Report an issue with this plugin	<input checked="" type="checkbox"/> 42.v0739460cda_c4

Git client pluginVersion4.4.0 4.3.0
Utility plugin for Git support in Jenkins
[Report an issue with this plugin](#)

Git pluginVersion5.1.0 5.0.2
This plugin integrates [Git](#) with Jenkins.
[Report an issue with this plugin](#)

GitHub API PluginVersion1.314-431.v78d72a_3fe4c3
This plugin provides [GitHub API](#) for other plugins.
[Report an issue with this plugin](#)

GitHub Branch Source PluginVersion1728.v859147241f49 1725.vd391eef681
Multibranch projects and organization folders from GitHub.
Maintained by CloudBees, Inc.
[Report an issue with this plugin](#)

GitHub pluginVersion1.37.1
This plugin integrates [GitHub](#) to Jenkins.
[Report an issue with this plugin](#)

Pipeline: GitHub Groovy LibrariesVersion42.v0739460cda_c4
Allows Pipeline Groovy libraries to be loaded on the fly from GitHub.
[Report an issue with this plugin](#)

→ CREDENTIALS SETUP :

- Now, we need to setup the CREDENTIALS for this GIT
 - For GIT, There are 2 ways to setup the credentials
 1. VIA SSH KEYS (SSH)
 2. VIA PERSONAL ACCESS TOKEN (HTTP)
- We will create the CREDENTIALS for both
- 1. **VIA SSH KEYS (SSH)**
- Go to DASHBOARD > MANAGE JENKINS > CREDENTIALS

The screenshot shows the Jenkins 'Manage Jenkins' interface under 'System Configuration'. On the left sidebar, 'Manage Jenkins' is highlighted with a red box. In the main content area, the 'Credentials' link is also highlighted with a red box. Other visible sections include 'System', 'Tools', 'Plugins', 'Nodes and Clouds', 'Security', 'Credential Providers', and 'Users'.

- Click on ADD CREDENTIALS

The screenshot shows the Jenkins 'Global credentials (unrestricted)' page. The top navigation bar includes 'Dashboard', 'Manage Jenkins', 'Credentials', 'System', and 'Global credentials (unrestricted)'. A red box highlights the 'Add Credentials' button at the bottom right of the page.

- Add the CREDENTIALS as below
 - SCOPE : GLOBAL

- ID : gitsshkey (provide a ID by your own)
- Description : git-key-priv
- Username : kishandev1999 (YOUR GIT USER NAME)

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▾

ID ?

gitsshkey

Description ?

git-key-priv

Username **kishandev1999** GIT USER NAME

→ In PRIVATE KEY > Add your private key of your local machine :

denials / System / Global credentials (unrestricted) / kishandev1999 (git-key-priv)

kishandev1999

Treat username as secret ?

Private Key

Enter directly

Key

 Concealed for Confidentiality Replace

Passphrase

- Leave the passphrase as empty
- Click on "CREATE"
- The credentials will be created as below screenshot



- **Credentials**

T	P	Store ↴	Domain	ID	Name
		System	(global)	gitsshkey	kishandev1999 (git-priv-ssh-key)

- Through this, we will setup the credentials with SSH KEYS

2. VIA PERSONAL ACCESS TOKEN (HTTP)

- Go to GIT > SETTINGS > DEVELOPER SETTINGS > PERSONAL ACCESS TOKEN (TOKENS- CLASSIC)
- Provide a NAME for your token and select the expire period for this token (30 days or 90 days)
- Select the required permission for this token to provide the access in git (I selected all)

The screenshot shows the GitHub 'Personal access tokens' section. On the left, there's a sidebar with 'GitHub Apps', 'OAuth Apps', and 'Personal access tokens' (selected). Under 'Personal access tokens', there are 'Fine-grained tokens' and 'Tokens (classic)' options. A 'Beta' button is visible. The main area is titled 'New personal access token (classic)'. It includes a note about personal access tokens functioning like OAuth tokens. A 'Note' input field is present, followed by a 'What's this token for?' field. An 'Expiration *' dropdown set to '30 days' with a note 'The token will expire on Wed, Jul 26 2023'. A 'Select scopes' section allows choosing from three categories: 'repo', 'workflow', and 'write:packages'. Each category has several sub-scopes listed.

- Click on "GENERATE TOKEN"
- It will generate a token, Copy the token Immediately
- EXAMPLE (MY TOKEN) : ghp_ffKgN6hjQlcLjiZcIkkSZQsL6aXxRW3kEG9M

The screenshot shows the GitHub 'Developer settings' page under 'Personal access tokens (classic)'. It lists a single token named 'For Jenkins' with the following details: scope 'admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, delete_packages, delete_repo, gist, notifications, project, repo, user, workflow, write:packages', last used 'within the last week', and expiration 'Fri, Sep 15 2023'. There are 'Generate new token' and 'Revoke all' buttons at the top right.

- Go to DASHBOARD > MANAGE JENKINS > CREDENTIALS
- Click on ADD CREDENTIALS
- Add the CREDENTIALS as below :
 - SCOPE : GLOBAL
 - USERNAME : kishandev1999 (YOUR GIT USER NAME)
 - Check (treat username as secret)
 - PASSWORD : ghp_ffKgN6hjQlcLjiZcIkkSZQsL6aXxRW3kEG9M (YOUR PERSONAL ACCESS TOKEN GENERATED FROM GIT)
 - ID : git-cred-id (GIVE YOUR OWN ID)
 - DESCRIPTION : THIS CREDENTIALS ARE FOR GIT HTTP AUTHENTICATIONS
 - CLICK ON "SAVE"

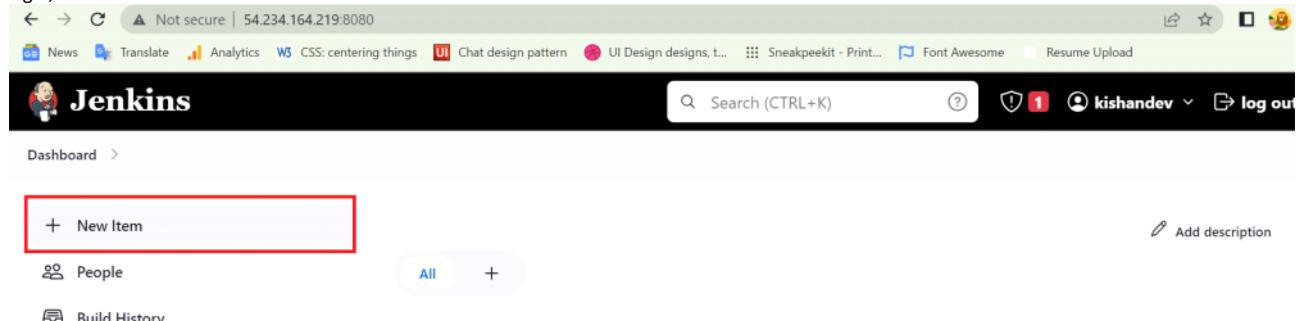
The screenshot shows the Jenkins 'Update credentials' page under 'Global credentials (unrestricted)'. The page title is 'This keys are for git http authentication'. It has fields for 'Scope' (set to 'Global'), 'Username' (set to 'kishandev1999'), 'Treat username as secret' (checked), 'Password' (set to 'Concealed'), 'ID' (set to 'git-cred-id'), and 'Description' (set to 'This keys are for git http authentication'). A 'Save' button is at the bottom.

→ Your Credentials will create like this

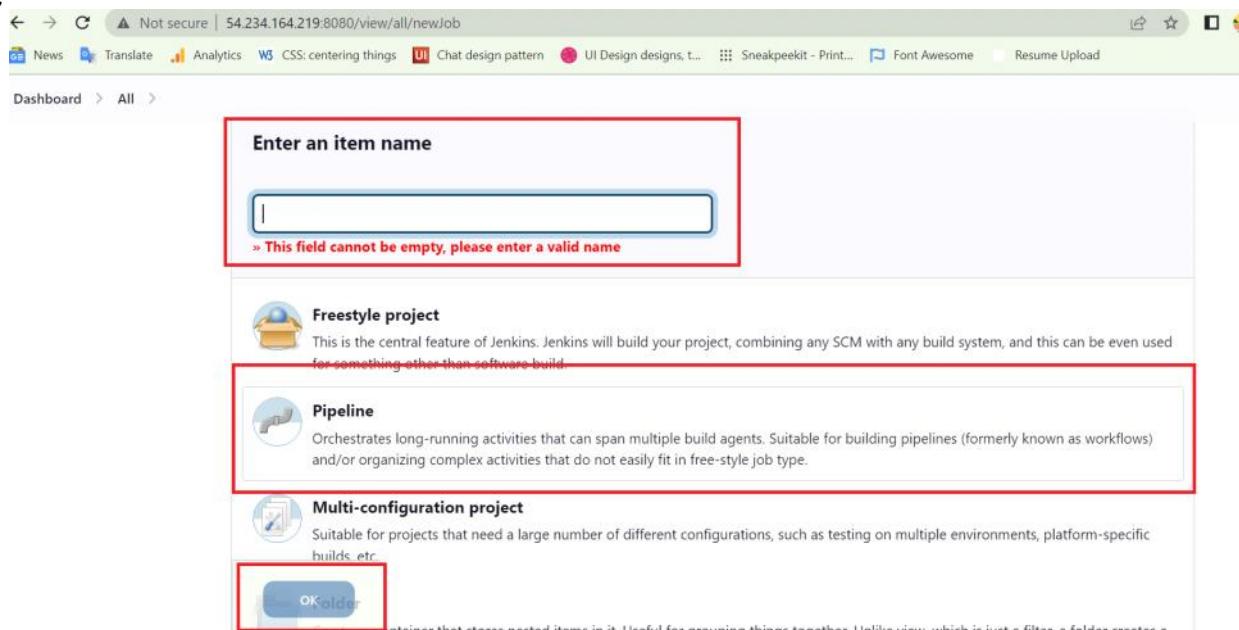


→ CREATING THE JENKINS PIPELINE WITH GIT AS SCM :

→ After login, Click on "NEW ITEM"



→ Now,



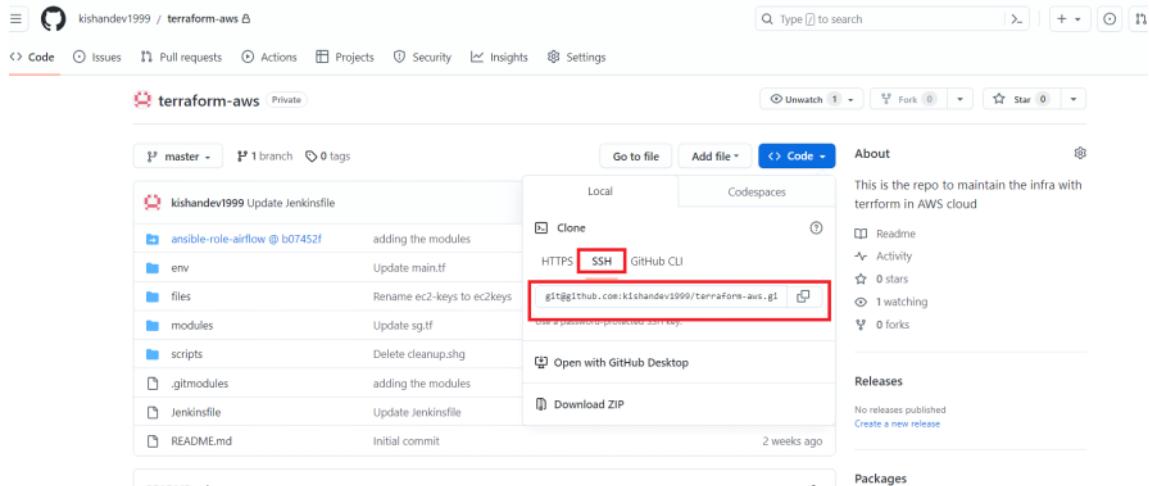
→ Now go to your pipeline > configure

→ Now, there are 2 ways to setting up the git SCM

→ As we created the 2 credentials will see the both ways of setting up the Jenkins with git

1. VIA SSH KEYS :

→ If you are choosing the SSH KEYS as authenticating credentials, then use this link from your GIT repo



→ In the Jenkins > Go to your pipeline > configure

→ Provide the REPOSITORY URL and select the credentials which we created with private keys

The screenshot shows the Jenkins Pipeline configuration page. Under the 'Pipeline' tab, the 'Definition' is set to 'Pipeline script from SCM'. In the 'SCM' section, 'Git' is selected. The 'Repositories' list contains one entry: 'Repository URL: git@github.com:kishandev1999/terraform-aws.git' and 'Credential: kishandev1999 (git-priv-ssh-key)'. There is also an 'Advanced' dropdown and a 'Add Repository' button.

→ NOTE : YOU WILL GET AN ERROR LIKE THIS BELOW

```
① Failed to connect to repository : Command "git ls-remote -h -- git@github.com:geekscodebook/arn-integrations.git HEAD"
returned status code 128:
stdout:
stderr: git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

→ This is due to may be the credentials setup you did wrongly. Cross check once again

→ Or This would be the main reason : You need to set up the accept first connection in jenkins

1. Go to manage jenkins > (security)configure global security {this is main step}

The screenshot shows the Jenkins 'Manage Jenkins' dashboard. Under 'System Configuration', the 'Security' section is highlighted with a red box. It contains a note: 'Building on the trust in Node can be a security issue. You should set up short-lived builds. See the documentation.' Below this are sections for 'Tools', 'Plugins', 'Nodes and Clouds', and 'Users'. A sidebar on the left lists 'New Item', 'People', 'Project History', 'Check File Fingerprint', 'Manage Jenkins' (which is selected and highlighted with a red box), and 'My Views'.

2. Select Git Host key verification configuration: Accept first connection

The screenshot shows the Jenkins 'Security' configuration page. Under 'SSH Server', 'SSH Port' is set to 'Fixed'. The 'Git Host Key Verification Configuration' section is highlighted with a red box. It contains a dropdown for 'Host Key Verifier Strategy' with 'Accept First Connection' selected. At the bottom are 'Save' and 'Apply' buttons.

→ Now, that error will disappear automatically.

2. VIA PERSONAL ACCESS TOKEN :

→ Now, copy the HTTP URL of your git repo :

The screenshot shows a GitHub repository page for 'terraform-aws'. In the 'Clone' section, the 'HTTPS' link is highlighted with a red box. The URL 'https://github.com/kishandev1999/terraform-aws' is displayed below it.

→ Now, configure your pipeline as below :

Dashboard > aws-infra-build > Configuration

Configure Pipeline

Definition

SCM : Git

Repository URL : <https://github.com/kishandev1999/terraform-aws.git>

Credentials : This keys are for git http authentication

Add Advanced Add Repository

- Provide the HTTP URL of your git repo and credentials you created with personal access token
- Click on save
- Now, your GIT is authenticated to Jenkins pipeline

→ Here, I explained 2 ways of authenticating the Git with Jenkins. You can prefer your own way to authenticate. For this project, I preferred 2nd method of authenticating (VIA PERSONAL ACCESS TOKEN). Because, It helped me to fetch the submodules data which we are using in the main module. This will be cover up in submodules section

➤ REFERENCES :

<https://www.guru99.com/jenkins-github-integration.html>

Authenticating git to the jenkins

[SSH integration between Jenkins & GitHub | Jenkins GitHub Integration using SSH Keys | Jenkins GitHub](#)

[Jenkins #5 | Integrate GitHub private Repository with Jenkins job using SSH Keys](#)

[How to Add Git Credentials in Jenkins](#)

JENKINS INTEGRATING WITH AWS

Monday, June 26, 2023 11:30 PM

INTEGRATING AWS WITH THE JENKINS :

PLUGINS REQUIRED :

1. CLOUD BEE AWS CREDENTIALS
2. TERRAFORM

- FIRST CREATE AN IAM ROLE AND GIVE THE ADMINISTRATIVE ACCESS TO IT
- Check the "AWS CLI AUTHENTICATION" document on how to create the role and setup
- Get the ACCESS KEY and SECRET KEY for it
- Now go to jenkins -> settings -> Plugin manager -> Install "Cloudbees AWS credentials"

The screenshot shows the Jenkins Plugin Manager interface. The search bar at the top contains the text "AWS c". Below the search bar, there are tabs for "Available plugins" and "Installed plugins", with "Installed plugins" currently selected. A list of installed plugins is displayed, including "Amazon Web Services SDK :: EC2" and "Amazon Web Services SDK :: Minimal". The "CloudBees AWS Credentials Plugin" is highlighted with a red box around its description and status. The plugin's name is "CloudBees AWS Credentials Plugin 191.vcb_f183ce58b_9". Its description states: "Allows storing Amazon IAM credentials within the Jenkins Credentials API. Store Amazon IAM access keys (AWSAccessKeyId and AWSSecretKey) within the Jenkins Credentials API. Also support IAM Roles and IAM MFA Token." The "Enabled" switch is checked.

- Now, we will setup the AWS credentials in jenkins
- Go to MANAGE JENKINS > "credentials"

The screenshot shows the Jenkins Manage Jenkins page. The left sidebar has a "Manage Jenkins" link highlighted with a red box. The main content area is titled "System Configuration" and includes sections for "Build Queue", "Build Executor Status", "Security", "Tools", "Plugins", "Nodes and Clouds", "Credential Providers", and "Users". The "Credentials" link under the "Security" section is also highlighted with a red box.

- Setup the credentials as below
 - KIND : AWS CREDENTIALS
 - SCOPE : GLOBAL
 - ID : AWS-JENKINS-CREDS
 - DESCRIPTION : INTEGRATING JENKINS AWS ACCOUNT
 - ACCESS KEY ID : <YOUR ACCESS KEY>
 - SECRET KEY ID : <YOUR SECRET KEY>

Jenkins

Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

New credentials

Kind: AWS Credentials

Scope: Global (Jenkins, nodes, items, all child items, etc)

ID: aws-jenkins-creds

Description: Integrating Jenkins and AWS account

Access Key ID: AKIASUJGTHYMTKZDOP

Secret Access Key: Please specify the Secret Access Key

IAM Role Support

Advanced

→ Click on SAVE

→ Now, we will create a new pipeline

- So, click on new and go to the bottom, you'll find pipeline syntax, click on it, you'll see this page.
- Fill with credentials and select your AWS credentials
- When you click on generate syntax. It will generate the syntax

News Translate Analytics CSS: centering things UI Chat design pattern UI Design designs, t... Sneakpeekit - Print... Font Awesome Resume Upload

Dashboard > credit-check-demo > Pipeline Syntax

This StepGenerator will help you learn the Pipeline Script code which can be used to define various steps. Pick a step you are interested in from the list, configure it, click Generate Pipeline Script, and you will see a Pipeline Script statement that would call the step with that configuration. You may copy and paste the whole statement into your script, or just the options you care about. Most parameters are optional and can be omitted in your script, leaving them at default values.

Steps

Sample Step: withCredentials Bind credentials to variables

withCredentials Secret values are masked on a best-effort basis to prevent accidental disclosure. Multiline secrets, such as the contents of a SSH private key file, are not masked. See the inline help for details and usage guidelines.

Bindings

AWS access key and secret

Access Key Variable: AWS_ACCESS_KEY_ID
Secret Key Variable: AWS_SECRET_ACCESS_KEY
Credentials: AKIASUJGTHYMTKZDOP (Integrating Jenkins and AWS account)

Generate Pipeline Script

```
withCredentials([object of type com.cloudbees.jenkins.plugins.awscredentials.AmazonWebServicesCredentialsBinding]) {
    // some block
}
```

→ You'll get the syntax like this :

```
withCredentials([<object of type com.cloudbees.jenkins.plugins.awscredentials.AmazonWebServicesCredentialsBinding>]) {
    // some block
}
In the <object of type, -- you need to add appropriate syntax value
```

→ The above syntax is not completely correct syntax

→ The proper syntax is mentioned below

```
pipeline{
    agent any
    stages{
        stage("AWS DEMO") {
            steps{
                withCredentials([
                    $class: 'AmazonWebServicesCredentialsBinding',
                    credentialsId: 'aws-jenkins-creds',
                    ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
                    SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
                ])
                sh "aws s3 ls"
            }
        }
    }
}
```

```

        sh "aws ec2 describe-instances --region=us-east-1"
    }
}
}
}
}

```

- Before, running the pipeline. We need to install the AWS CLI in the jenkins VM
- So, SSH into the VM and install the AWS CLI
- **\$apt install awscli**

→ Check the below screenshots for reference :

```

Last login: Mon May 29 09:11:04 2023 from 49.37.132.20
ubuntu@ip-172-31-82-1:~$ sudo su
root@ip-172-31-82-1:/home/ubuntu# aws s3 ls
Command 'aws' not found, but can be installed with:
snap install aws-cli  # version 1.15.58, or
apt install awscli   # version 1.22.34-1
See 'snap info aws-cli' for additional versions.
root@ip-172-31-82-1:/home/ubuntu# apt install awscli
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:

```

- - No VM guests are running outdated hypervisor (qemu) binaries on this host.
 - root@ip-172-31-82-1:/home/ubuntu# aws --version
 - aws-cli/1.22.34 Python/3.10.6 Linux/5.19.0-1025-aws botocore/1.23.34
 - root@ip-172-31-82-1:/home/ubuntu# aws s3 ls
 - Unable to locate credentials. You can configure credentials by running "aws configure".
- Now trigger the pipeline, it will work

- **NOTE :**
- You need "t2.small" range instance
- This is the minimum instance type to run the Jenkins server

- HELPFUL LINKS :
- <https://andresaap.medium.com/how-to-install-docker-aws-cli-eksctl-kubectl-for-jenkins-in-linux-ubuntu-18-04-3e3c4ceeb71>
- [Demo - How to integrate AWS & Jenkins | AWS Credentials Plugin | For DevOps Training - 9886611117](#)



TERRAFORM SETUP IN JENKINS

Monday, June 26, 2023 11:28 PM

- Firstly we need to set up the terraform inside the jenkins :
- You need to install this terraform plugin from "plugin manager":

The screenshot shows the Jenkins 'Plugin Manager' interface. The left sidebar has tabs for 'Updates', 'Available plugins' (which is selected), 'Installed plugins', and 'Advanced settings'. The main area has a search bar with 'terraform' typed in. A table lists the 'Terraform 1.0.10' plugin, which is checked for installation. The table columns are 'Install', 'Name', and 'Released'. Below the table, a note says: 'This plugin provides a build wrapper for Terraform to launch and destroy infrastructure.' At the bottom are two buttons: 'Install without restart' and 'Download now and install after restart'. To the right of these buttons is a status message: 'Update information obtained: 5 min 7 sec ago' and a 'Check now' link.

- Click on install without restart.
- Once, it got installed, then you need to set up the terraform installation using the "global tool configuration"
Go to manage jenkins > "global tool configuration" > Terraform
 - Here you can choose the terraform to install automatically
 - But, we'll try to install the terraform inside the VM and then we'll provide the path in the jenkins
 - Go to this page : https://developer.hashicorp.com/terraform/downloads?product_intent=terraform

The screenshot shows the Terraform official website. The top navigation bar includes links for News, Translate, Analytics, CSS: centering things, Chat design pattern, UI Design designs, UI Design, Sneakpeekit - Print..., Font Awesome, and Resume Upload. The main content area has a sidebar with links for Terraform Home, Install Terraform (selected), Getting Started, What is Terraform?, Terraform Tutorials, Terraform Cloud Tutorials, Resources, Tutorial Library, and Certifications. The main content area shows a terminal window with commands: '\$ echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://releases.hashicorp.com/terraform stable main" | sudo tee /etc/apt/sources.list.d/terraform.list && sudo apt update && sudo apt install terraform'. Below this is a section titled 'Binary download for Linux' with three options: '386 Version: 1.4.6 Download', 'AMD64 Version: 1.4.6 Download', and 'ARM Version: 1.4.6 Download'. To the right, there's a sidebar with sections for 'About Terraform', 'Featured docs', and 'CLI'. A context menu is open over the ARM download button, with options like 'Open link in new tab', 'Save link as...', 'Copy link address', 'Inspect', and 'Preferences'. At the bottom right of the menu is a blue 'ACCEPT' button.

- Select the linux and copy the link of 64 bit
- Run the below commands in your jenkins machine after doing ssh

```
$ ssh -i jenkins.pem ubuntu@3.91.226.196
$ wget https://releases.hashicorp.com/terraform/1.4.6/terraform_1.4.6_linux_amd64.zip
$ which terraform
$ apt install unzip
```

```
$unzip terraform_1.4.6_linux_amd64.zip  
$mv terraform /usr/bin  
$which terraform  
$terraform version
```

LOGS :

```
asus@KishanDev MINGW64 /y/AWS  
$ ssh -i jenkins.pem ubuntu@3.91.226.196  
The authenticity of host '3.91.226.196 (3.91.226.196)' can't be established.  
ED25519 key fingerprint is SHA256:vaxdHTDOt0t7s5CYH0SHkbLQh2JB5h7bN8sik4dCbQ.  
This host key is known by the following other names/addresses:  
~/.ssh/known_hosts:108: 44.204.172.179  
~/.ssh/known_hosts:111: 54.172.187.122  
~/.ssh/known_hosts:112: 54.235.229.128  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added '3.91.226.196' (ED25519) to the list of known hosts.  
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-1025-aws x86_64)
```

* Documentation: <https://help.ubuntu.com>
* Management: <https://landscape.canonical.com>
* Support: <https://ubuntu.com/advantage>

System information as of Mon May 29 09:11:02 UTC 2023

```
System load: 0.0      Processes:    98  
Usage of /: 37.1% of 7.57GB  Users logged in:  0  
Memory usage: 73%      IPv4 address for eth0: 172.31.82.1  
Swap usage: 0%
```

* Ubuntu Pro delivers the most comprehensive open source security and compliance features.

<https://ubuntu.com/aws/pro>

Expanded Security Maintenance for Applications is not enabled.

8 updates can be applied immediately.

To see these additional updates run: `apt list --upgradable`

Enable ESM Apps to receive additional future security updates.

See <https://ubuntu.com/esm> or run: `sudo pro status`

Last login: Sun May 28 12:30:37 2023 from 49.37.132.20

ubuntu@ip-172-31-82-1:~\$ sudo su

```
root@ip-172-31-82-1:/home/ubuntu# wget https://releases.hashicorp.com/terraform/1.4.6/terraform_1.4.6_linux_amd64.zip  
--2023-05-29 09:13:16-- https://releases.hashicorp.com/terraform/1.4.6/terraform_1.4.6_linux_amd64.zip  
Resolving releases.hashicorp.com (releases.hashicorp.com)... 108.138.85.30, 108.138.85.31, 108.138.85.53, ...  
Connecting to releases.hashicorp.com (releases.hashicorp.com)|108.138.85.30|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 20779821 (20M) [application/zip]  
Saving to: 'terraform_1.4.6_linux_amd64.zip'
```

```
terraform_1.4.6_lin 100%[=====] 19.82M --.KB/s in 0.1s
```

```
2023-05-29 09:13:16 (146 MB/s) - 'terraform_1.4.6_linux_amd64.zip' saved [20779821/20779821]
```

```
root@ip-172-31-82-1:/home/ubuntu# which terraform
root@ip-172-31-82-1:/home/ubuntu# ls
jenkins-setup.sh terraform_1.4.6_linux_amd64.zip
root@ip-172-31-82-1:/home/ubuntu# unzip terraform_1.4.6_linux_amd64.zip
Command 'unzip' not found, but can be installed with:
apt install unzip
root@ip-172-31-82-1:/home/ubuntu# apt install unzip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Suggested packages:
  zip
The following NEW packages will be installed:
  unzip
0 upgraded, 1 newly installed, 0 to remove and 8 not upgraded.
Need to get 174 kB of archives.
After this operation, 385 kB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates/main amd64 unzip amd64 6.0-26ubuntu3.1 [174 kB]
Fetched 174 kB in 0s (6580 kB/s)
Selecting previously unselected package unzip.
(Reading database ... 66753 files and directories currently installed.)
Preparing to unpack .../unzip_6.0-26ubuntu3.1_amd64.deb ...
Unpacking unzip (6.0-26ubuntu3.1) ...
Setting up unzip (6.0-26ubuntu3.1) ...
Processing triggers for man-db (2.10.2-1) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

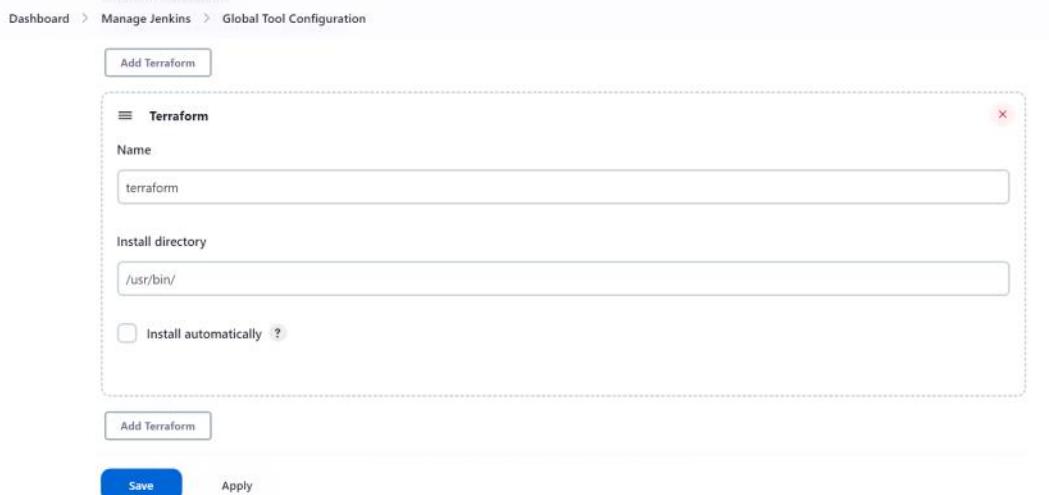
No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@ip-172-31-82-1:/home/ubuntu# ls
jenkins-setup.sh terraform_1.4.6_linux_amd64.zip
root@ip-172-31-82-1:/home/ubuntu# unzip terraform_1.4.6_linux_amd64.zip
Archive: terraform_1.4.6_linux_amd64.zip
  inflating: terraform
root@ip-172-31-82-1:/home/ubuntu# ls
jenkins-setup.sh terraform_1.4.6_linux_amd64.zip
root@ip-172-31-82-1:/home/ubuntu# mv terraform /usr/bin
root@ip-172-31-82-1:/home/ubuntu# ls
jenkins-setup.sh terraform_1.4.6_linux_amd64.zip
root@ip-172-31-82-1:/home/ubuntu# which terraform
/usr/bin/terraform
root@ip-172-31-82-1:/home/ubuntu# terraform version
Terraform v1.4.6
on linux_amd64
```

root@ip-172-31-82-1:/home/ubuntu#

- Now, the terraform is in this path : /usr/bin/
- Provide this path in your jenkins, your jenkins will take terraform from there



- Click and save and apply

➤ REFERENCES :

- [Launch AWS EC2 with Jenkins | DevOps Tutorial](#)

<https://registry.terraform.io/>

[Automated Deployment from GitHub to EC2 instance using Jenkins CI/CD | Part 3](#)

[Jenkins and AWS EC2 Connection Over SSH | Jenkins CI/CD EC2 Instance |](#)

[Automate EC2 setup using Jenkins & Ansible | Automate Infrastructure setup using Ansible and Jenkins](#)

This link will have all the cloud related examples for terraform

For AWS :

<https://registry.terraform.io/providers/hashicorp/aws/latest>

<https://registry.terraform.io/providers/hashicorp/aws/latest/docs>

ADDING KEYS

Thursday, June 22, 2023 10:22 PM

- First select a folder and generate the keys by using the below command:

ssh-keygen

- Once the keys are generated, provide the name as them as you like
(EX : ec2keys)

\$ ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/y/Devops Udemy/ssh-keys-folder/.ssh/id_rsa): ec2keys

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in ec2keys

Your public key has been saved in ec2keys.pub

The key fingerprint is:

SHA256:C5OTu9x97QUQNVQHNOWynDgpcxxo1PA009s/YGwjTN4 asus@KishanDev

The key's randomart image is:

+---[RSA 3072]---

```
| oo==B++|  
| .=+.+.|  
| * *.o.|  
| o . = E.+.|  
| * S o X B .|  
| = . + . o|  
| .. . o|  
| . o . . .|  
| o . . . .|
```

asus@KishanDev MINGW64 /y/AWS/AWS-Terraform/env/dev/files (master)

\$ ssh-keygen

Generating public/private rsa key pair.

Enter file in which to save the key (/y/Devops Udemy/ssh-keys-folder/.ssh/id_rsa): ec2keys

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in ec2keys

Your public key has been saved in ec2keys.pub

The key fingerprint is:

SHA256:C5OTu9x97QUQNVQHNOWynDgpcxxo1PA009s/YGwjTN4 asus@KishanDev

The key's randomart image is:

+---[RSA 3072]---

```
| oo==B++|  
| . =+.+.|  
| * *.o.|  
| o . = E.+.|  
| * S o X B .|  
| = . + . o|  
| .. . o|  
| . o . . .|  
| o . . . .|
```

+---[SHA256]---

- In terraform, create the key-pair using these generated keys (in modules section) :

```
#KEY-PAIR  
resource "aws_key_pair" "instance-key" {  
    key_name = "${local.aws_prefix}-keys"  
    public_key = var.public-key  
}
```

```
 1 #NAMING  
 2 locals {  
 3     aws_prefix = "aws-${substr(var.region,0,2)}-${substr(var.environment,0,2)}-${var.environment}-airflow"  
 4 }  
 5  
 6  
 7 #KEY-PAIR  
 8 resource "aws_key_pair" "instance-key" {  
 9     key_name = "${local.aws_prefix}-keys"  
10     public_key = var.public-key  
11 }
```

- Attach these keys to the "instance-template"

→ key_name = aws_key_pair.instance-key.key_name

→ Copy the keys from your local machine (ec2keys and ec2keys.pub) and save them in this files section.

The screenshot shows a GitHub repository interface. On the left, there's a sidebar with a 'Code' tab and a 'master' branch dropdown. Below it are folder icons for 'env', 'dev', 'prod', 'testing', 'files', 'ec2keys', and 'ec2keys.pub'. The main area displays a list of files under the 'files' directory. Two files are listed: 'ec2keys' with a commit message 'Rename ec2-keys to ec2keys' and 'Create ec2keys.pub' at 2 weeks ago; and 'ec2keys.pub' with a commit message 'Create ec2keys.pub' at 3 weeks ago. At the top right, there are buttons for 'Add file' and 'History'.

This screenshot shows three separate GitHub code editor windows. The top window shows the 'main.tf' file with Terraform code defining AWS resources. The middle window shows the 'ec2keys' file containing a single line of text: '-----BEGIN RSA PRIVATE KEY-----'. The bottom window shows the 'ec2keys.pub' file containing a long string of RSA public key characters.

→ Apply the security group rules

→ SSH-PORT:22-MY IP

This screenshot shows a code editor with the 'main.tf' file open. The file contains Terraform configuration for an AWS security group named 'instance-sg'. It includes three ingress rules: one for SSH (port 22), one for HTTP (port 8080), and one for connecting to a database (port 5432). The 'from_port' and 'to_port' fields for the first two rules are both set to 22, which is highlighted in red. The 'security_groups' field for the database rule is set to '["sg-08e2223c3f76c2b11"]'. The code editor has tabs for 'inflow-testing logs.txt', 'main.tf', 'modules', 'ec2keys', and 'ec2keys.pub'.

EC2 > Security Groups > sg-02d6c7eeb6380008c - aws-us-d-airflow-sg > Edit inbound rules

Edit inbound rules Info

Inbound rules control the incoming traffic that's allowed to reach the instance.

This screenshot shows the 'Edit inbound rules' page for a specific security group. It lists three existing rules:

- Rule ID: sgr-03d8506295bf2deb6, Type: Custom TCP, Protocol: TCP, Port range: 8080, Source: 49.37.154.254/32, Description: 'HTTP from VPC'.
- Rule ID: sgr-085591c28899fa166, Type: SSH, Protocol: TCP, Port range: 22, Source: Custom, Description: 'SSH from VPC'.
- Rule ID: sgr-0285532f79c918954, Type: PostgreSQL, Protocol: TCP, Port range: 5432, Source: Custom, Description: 'connecting DATABASE and VM'.

At the bottom, there are buttons for 'Add rule', 'Cancel', 'Preview changes', and 'Save rules'.

Run this command to connect or SSH into the VM:

```
$ ssh -i ec2keys ubuntu@ec2-44-204-154-190.compute-1.amazonaws.com
```

Or

```
$ ssh -i ec2keys ubuntu@44.204.154.190
```

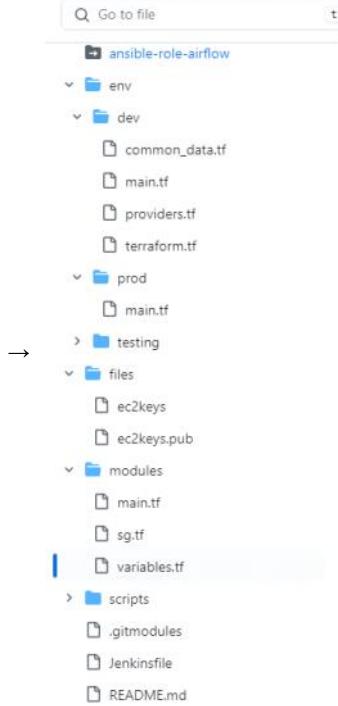
```
$ ssh -l <key-name> username@public-ip
```

```
ubuntu@ec2-44-204-154-190.compute-1.amazonaws.com: Permission denied (publickey).  
asus@KishanDev MINGW64 /y/AWS/AWS-Terraform/env/dev/files (master)  
$ ssh -i ec2keys ubuntu@ec2-44-204-154-190.compute-1.amazonaws.com  
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-1028-aws x86_64)  
  
* Documentation: https://help.ubuntu.com
```

TERRAFORM

Monday, June 26, 2023 11:33 PM

- In GIT -> create a private repo. Here I created a repo with name "terraform-aws"
- Created file architecture as below :



→ MODULES

→ (main.tf)

- In terraform, Modules are the centralized code where we can recall their values and repeatedly and created no.of resources as per our requirement
- In this Repo, I created a modules section with main.tf, sg.tf, variables.tf
- Now, the main code will be scripted in main.tf
- The security policies (firewall rules) will be in the sg.tf
- Variables are defined in variables.tf

- LOCALS

```
#FOR NAMING THE AWS COMPONENTS WE ARE USING THESE LOCALS
#LOCALS
locals {
  aws_prefix = "aws-${substr(var.region,0,2)}-${substr(var.environment,0,1)}-${var.application}"
  #aws-eu-west2-airflow
}
```

- This locals are like reusable variables. If you want to define the repeated values in your code again and again, then just simply define in locals and use the local values in your code.

- I created this local for the standard naming convention

- "aws-
○ \${substr(var.region,0,2)}-
○ \${substr(var.environment,0,1)}-
○ \${var.application}"
○ The final output will be like :
- //This will simply give name aws
//This will take the value for region and cut the first two values as substring EX. europe - it will take (eu)
//This will take the first letter of the environment variable

//this will attach the application name mentioned in the variables
#aws-eu-west2-airflow

- KEY-PAIRS :

```
#KEY-PAIR
#THIS WILL CREATE KEY PAIR FOR THE INSTANCES
resource "aws_key_pair" "instance-key" {
  key_name = "${Local.aws_prefix}-keys"
  public_key = var.public-key
}
```

- This will create the key-pairs
- The key name will be created with the locals defined above
- If you want to call anything defined in the locals, you need to call them as \${local.<defined_variable_name>}
- The key we are calling from the variables

- ADDITIONAL DISK

```
#ADDITIONAL DISK CREATION
#THIS WILL CREATE AN ADDITIONAL DISK
resource "aws_ebs_volume" "additional-disk" {
  availability_zone = var.zone
  size              = var.add-disk-size
  tags = {
    Name = "${local.aws_prefix}-ini-additional-disk"
    Application = var.application
    device_name = var.add-disk-device-name
  }
}
```

- This will create an ADDITIONAL DISK
 - Availability zone will be defined in the variables
 - Size also we are defining in static
 - Tags - We are using the locals to declared the name accordingly, Application will be the application name defined in the variables
 - device_name [DEFINED IN TAGS SECTION ONLY]= (This is the main important time in declaring the device name here)
- For this ADDITIONAL DISK, the devicename defined as "/dev/sdb"

```
# variable defined in variables.tf
variable "add-disk-device-name" {
  default = "/dev/sdf"
}
```

- REFER THE TABLE FOR DEVICE NAMES

Type of Virtualization	Available String Blocks	Reserved for Root	Instance Store Volumes	Recommended for EBS Volumes
Paravirtual	/dev/sd[a-z] /dev/sd[a-z][1-15] /dev/hd[a-z] /dev/hd[a-z][1-15]	/dev/sda1	/dev/sd[b-e]	/dev/sd[f-p] /dev/sd[f-p][1-6]
HVM	/dev/sd[a-z] /dev/xvd[b-c][a-z]	Differs by AMI /dev/sda1 or /dev/xvda	/dev/sd[b-e] /dev/sd[b-y] (hs1.8xlarge)	/dev/sd[f-p]

REFERENCE <https://www.softnas.com/docs/softnas/v3/html/ebs_volumes_and_device_mapping_print.html>
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/device_naming.html>

- LAUNCH TEMPLATE :

```
resource "aws_launch_template" "linux" {
  #count = var.linux ? 1 : 0
  name = "${local.aws_prefix}-linux-machine-templates"
  instance_type = var.instance-type          #t2.micro
  key_name = aws_key_pair.instance-key.key_name
  image_id = var.image-id
  vpc_security_group_ids = [aws_security_group.instance-sg.id]

  #BOOT DISK
  block_device_mappings {
    device_name = var.boot-disk-device-name      #/dev/sda1
    ebs {
      volume_size = var.boot-disk-size           #10
      delete_on_termination = true
      volume_type = var.volume-type            #STANDARD
    }
  }
  monitoring {
    enabled = true
  }
  placement {
    availability_zone = var.zone
  }
}
```

```

tag_specifications {
  resource_type = "instance"
  tags = {
    Name = "${local.aws_prefix}"
    script = "${var.application}-init"
    image = var.image_name
  }
}

#DISK TAGS
tag_specifications {
  resource_type = "volume"
  tags = {
    #count = length(var.disk-device-name-list)
    Name = "${local.aws_prefix}-ini-boot-disks"
    #script = "${var.application}-init"
    #image = var.image_name
  }
}
#user_data = var.init-script
#aws_launch_template.linux.tag_specifications.tags.script.value
}

```

→ In this template, We configure the below components

- Keys
- Instance Type
- Image
- Security group
- Boot Disk
- Disk Tags
- User data

→ KEYS : We are using the key-pairs which are already created above in KEY PAIRS section

→ Instance Type: It is taken from the variables (t2.micro, t2.small...)

→ Image : Here we are using UBUBTU image declared in the variable section

→ Security Group : This will map the security group which we create in sg.tf file with its id "aws_security_group.instance-sg.id"

→ Boot Disk : Instance definitely needs ROOT VOLUME (without this we can't create the instance)

- The device name will be mapped as "/dev/sda1"
- SIZE : 10GB
- Volume Type : STANDARD

→ TAGS SPECIFICATION : This will map the tag to only instance template for instance and volume separately

→ USER DATA: If you want to add any script manually, you can uncomment and use this variable. Now, here I'm provisioning through ansible and shell scripting. So, I commented this.

○ LAUNCHING INSTANCE FROM TEMPLATE:

```

resource "aws_instance" "instance-linux" {
#count = 1
availability_zone = var.zone
launch_template {
  name = aws_launch_template.linux.name
  #id = aws_launch_template.linux.id
}
connection {
  type      = "ssh"
  host     = self.public_ip
  user     = "ubuntu"
  private_key = var.private-key
  timeout   = "4m"
}
provisioner "local-exec" {
  command = "echo ${self.private_ip} >> /var/lib/jenkins/workspace/aws-infra-build/private_ip.txt"
}

provisioner "local-exec" {
  command = "echo ${self.public_ip} >> /var/lib/jenkins/workspace/aws-infra-build/public_ip.txt"
}
tags = {
  Name = "${local.aws_prefix}-instance"
  #Name = "${local.aws_prefix}-instance-${count.index}"
  #Script = var.source_script
}
}

```

→ AVAILABILITY ZONE: provide through variables

- LAUNCH TEMPLATE : calling the created template via name
- CONNECTION : via SSH and providing the key
- LOCAL-EXEC : Getting the public IP and private IP of the VM and pasting in the files
- TAGS : For NAME of the VM

○ **ATTACHING THE ADDITIONAL DISK AFTER VM GOT CREATED :**

- Once, the VM gets created based on the template attached. After that we are attaching the created ADDITIONAL DISK to this VM

```
resource "aws_volume_attachment" "ebs-additional-disk" {
  device_name = "/dev/sdf"
  volume_id = aws_ebs_volume.additional-disk.id
  instance_id = aws_instance.instance-linux.id

}
```

- DEVICE NAME : "/dev/sdf" <REFER THE TABLE>
{DEVICE NAME you have to define here. Because, the instance has to use this additional volume apart from the root volume}
- VOULME_ID : attach the additional disk with ID reference
- INSTACE_ID : attach the ID of instance to which this ADDITIONAL DISK you are planning to attach

→ **(Sg.tf)**

- Creating the security group (Firewall Rules) :

```
resource "aws_security_group" "instance-sg" {
  name      = "${local.aws_prefix}-sg"
  description = "Allow MYIP inbound traffic"
  vpc_id    = "vpc-042b7992bbcb14df5" #ATTACH THE EXISTING VPC ID

#INBOUND RULE
  ingress {
    description  = "SSH from VPC"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = [var.myip]
    #ipv6_cidr_blocks = [var.myip]
  }
  ingress {
    description  = "SSH from VPC"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    #ipv6_cidr_blocks = [var.myip]
  }
  ingress {
    description  = "HTTP from VPC"
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = [var.myip]
    #ipv6_cidr_blocks = [var.myip]
  }
  ingress {
    description  = "connecting DATABASE and VM"
    from_port   = 5432
    to_port     = 5432
    protocol    = "tcp"
    security_groups = ["sg-08e2223c3f76c2b11"]
    #ipv6_cidr_blocks = [var.myip]
  }

#OUTBOUND RULE
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
  }
}
```

```

    cidr_blocks  = ["0.0.0.0/0"]
    ipv6_cidr_blocks = "::/0"
}

tags = {
  Name = "airflow-security-group"
}

```

- VPC : Using the default VPC id
- Apply only inbound rules
- TCP - 22 - MYIP (IP DEFINED IN VARIABLES)
- HTTP - 8080 - ["0.0.0.0/0"] (Means anywhere fromIPv4)
- DATABASE - 5232 - "DATABASE SECURITY GROUP" > "CONNECTING INSTANCE TO DATABASE "
- DON'T DO ANY CHANGES IN THE OUTBOUND RULES

→ **(Variables.tf)**

```

variable zone {
  default = ""
}

variable region {
  default = ""
}

variable AMI-ID {
  default = ""
}

variable project-name {
  default = ""
}

variable instance-type {
  default = ""
}

variable "environment" {
  default = ""
}

variable "project" {
  default = ""
}

variable volume-type {
  default = ""
}

variable linux {
  default = ""
}

variable key-name {
  default = ""
}

variable add-disk-size {
  default = ""
}

variable boot-disk-size {
  default = ""
}

variable image-id {
  default = ""
}

variable subnet-id {
  default = ""
}

variable public-key {
  default = ""
}

```

```

}

variable application {
  default = ""
}

variable security-group {
  default = ""
}

variable myip {
  default = ""
}

variable private-key {
  default = ""
}

variable user {
  default = ""
}

variable out-pub-ip {
  default = ""
}

variable out-priv-ip {
  default = ""
}

variable source_script {
  default = ""
}

variable destination_loc {
  default = ""
}

variable init-script {
  default = ""
}

variable image_name {
  default = ""
}

variable "boot-disk-device-name" {
  default = "/dev/sda1"
}

variable "add-disk-device-name" {
  default = "/dev/sdb"
}

```

FROM THE FOLDERS : ENV -> DEV :

→ Files architecture



→ **(Providers.tf)**

```
provider "aws" {
  region = var.region
}
```

- In this providers.tf, we will mention the cloud provider and the region

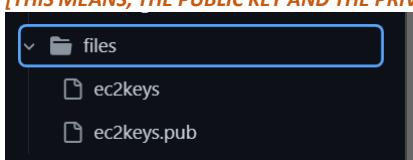
→ **(Main.tf)**

```
#Airflow#
module "airflow" {
  source    = "../../modules"
  region    = var.region
  zone      = var.zone
  application = "airflow"
  instance-type = "t2.micro"
  environment = var.environment
  volume-type = "standard"
  image-id   = var.image
  image_name = "ubuntu"
  add-disk-size = 10
  boot-disk-size = 10
  public-key  = data.template_file.ssh-ec2keys-pub.rendered
  private-key = data.template_file.ssh-ec2keys-priv.rendered
  myip       = "49.37.134.186/32" #"192.168.29.40/32"
  security-group = var.security-group
  user        = var.user
  #source_script = "../scripts/airflow-init.sh"
  #destination_loc = "/tmp/airflow-init.sh"
  #init-script = base64encode(data.template_file.airflow-script.rendered)
 #${base64encode(data.template_file.test.rendered)}
}
```

- In main.tf, we will call the modules with convenient name [module "airflow"]

- Source : "call the modules (folder)" [NOTE : ../../ This means the modules folder is in the 2 steps back from the current path. So, please give the ../../ accordingly]
- Region : calling region from variables in terraform.tf files
- Zone : calling region from variables in terraform.tf files
[NOTE: Declare the common values, EX. ZONE, REGION in variables]
- Environment : calling from variables (development, testing, production)
- Volume-type : standard
- Image : providing the image ID in terraform.tf [ami-0557a15b87f6559cf]
- Add-disk-size (additional disk size) : 10
- Boot-disk-size (boot disk size) : 10
- Public-key = **data.template_file.ssh-ec2keys-pub.rendered**
- Private-key = **data.template_file.ssh-ec2keys-priv.rendered**

[THIS MEANS, THE PUBLIC KEY AND THE PRIVATE KEY ARE CALLING FROM THE FILES PRESENT IN THE BELOW SCREENSHOT]



- **[THESE KEYS ARE RENDERED IN THE commondata.tf]**

terraform-aws / env / dev / common_data.tf

kishandev1999 Update common_data.tf

Code Blame 18 lines (14 loc) · 523 Bytes

```

1   data "template_file" "ssh-ec2keys-pub" {
2     template = file("../files/ec2keys.pub")
3   }
4
5   data "template_file" "ssh-ec2keys-priv" {
6     template = file("../files/ec2keys")
7   }
8
9

```

- My-ip = "enter your computer IP ADDRESS MANUALLY"
- Security group = (No need to define anything. It is already declared in launch template)
- User = defined in terraform.tf [ubuntu]

→ If you are declaring the startup script in the user data use this values

```
#source_script = "../scripts/airflow-init.sh"
#destination_loc = "/tmp/airflow-init.sh"
#init-script = base64encode(data.template_file.airflow-script.rendered)
#"${base64encode(data.template_file.test.rendered)}"
```

→ This is the source script (present in /scripts/folder in git repo and placing the destination location /tmp/airflow)

→ The initscript should be in base64decoder

→ (common_data.tf)

```
data "template_file" "ssh-ec2keys-pub" {
  template = file("../files/ec2keys.pub")
}

data "template_file" "ssh-ec2keys-priv" {
  template = file("../files/ec2keys")
}
```

→ We are configuring our keys in the VM through this common_data.tf

→ Generate the keys in your local machine by : **\$ssh-keygen**

→ Store the public and private key in this files folder

terraformer-aws / files /

kishandev1999 Rename ec2-keys to ec2keys 3bca4d8 · 2 weeks ago History

Name	Last commit message	Last commit date
..		
ec2keys	Rename ec2-keys to ec2keys	2 weeks ago
ec2keys.pub	Create ec2keys.pub	3 weeks ago

→ **template = file("../files/ec2keys")**

→ [note: the path ../files/ec2keys, This/ means the files folder located two steps back from the present director]

→ This will map the public key which mentioned in the modules

Modules/main.tf

#KEY-PAIR

#THIS WILL CREATE KEY PAIR FOR THE INSTANCES

```
resource "aws_key_pair" "instance-key" {
  key_name = "${Local.aws_prefix}-keys"
  public_key = var.public-key
}
```

→ This public key will render the keys in the files folder and create the key-pair, the common_data.tf will help to fetch the keys from the file using "data"

→ (terraform.tf)

```
variable "region" {
  default = "us-east-1"
}

variable "zone" {
```

```

    default = "us-east-1a"
}

variable "user" {
  default = "ubuntu"
}

variable "environment" {
  default = "development"
}

variable "subnet" {
  default = ""
}

variable "image" {
  default = "ami-0557a15b87f6559cf"
}
#ami-0557a15b87f6559cf -- UBUNTU

variable "ssh" {
  default = ""
}

variable "security-group" {
  default = ""
}

variable "source_script" {
  default = ""
}

variable "destination_loc" {
  default = ""
}

variable "public-key" {
  default = ""
}

variable "private-key" {
  default = ""
}

variable "myip" {
  default = ""
}

```

SCRIPTS :

- [scripts/ansible-airflow-config.sh](#)
- This is the script to run the Ansible inside the JENKINS

```

#!/bin/bash
pwd
##Getting the IP address :
present_path="/var/lib/jenkins/workspace/aws-infra-build"
cd $present_path
##cd /var/lib/jenkins/workspace# cd aws-infra-build
#pwd

private_ip=`cat private_ip.txt`
echo $private_ip

public_ip=`cat public_ip.txt`
echo $public_ip

chmod 700 /var/lib/jenkins/workspace/aws-infra-build/files/ec2keys
ansible_dir="/var/lib/jenkins/workspace/aws-infra-build/ansible-role-airflow/tasks/main.yaml"
keys_dir="/var/lib/jenkins/workspace/aws-infra-build/files/ec2keys"
sleep 45

```

```
ansible-playbook -i ${public_ip}, -u ubuntu --private-key=${keys_dir} ${ansible_dir}
echo "ansible-playbook -i ${public_ip}, -u ubuntu --private-key=${keys_dir} ${ansible_dir}"
```

- First we are collecting the IP address of created VM, which is copied in the public.txt file
- `public_ip=`cat public_ip.txt``
- Now, Providing the permissions for the private key file
- `chmod 700 /var/lib/jenkins/workspace/aws-infra-build/files/ec2keys`
- This is the ansible command to run the playbook in the VM
- `$ ansible-playbook -i ${public_ip}, -u ubuntu --private-key=${keys_dir} ${ansible_dir}`
- SYNTAX : ansible-playbook -i <\${PUBLIC IP OF VM}> -u ububtu --private-key=\${PRIVATE-KEY OF VM} \${ANSIBLE MAIN.YAML FILE}
- i is inventory, -u is username of VM

→ `scripts/cleanup.sh`

- This cleanup.sh will clear the files like pubilc_ip.txt and private_ip.txt, in order to repeat the occurrence of IP address

```
##Getting the IP address :
present_path_cu="/var/lib/jenkins/workspace/aws-infra-build"
cd $present_path_cu
pwd

priv_file=private_ip.txt
pub_file=public_ip.txt

if [ -f "$priv_file" ]; then
rm private_ip.txt
else
echo "file not exists"
fi

if [ -f "$pub_file" ]; then
rm public_ip.txt
else
echo "file not exists"
fi
```

ANSIBLE

Monday, June 26, 2023 11:34 PM

→ For Ansible configuration, create a new repo in git:

The screenshot shows a GitHub repository interface. The repository name is 'ansible-role-airflow'. The 'Code' tab is selected, showing the 'master' branch. On the left, there are two main directory structures: 'files' and 'tasks'. Under 'files', there are 'init-script.sh' and 'rc-local.service'. Under 'tasks', there are 'main.yaml' and 'README.md'. On the right, a list of commits is displayed. One commit by 'kishandev1999' is highlighted, showing the update of 'init-script.sh' and the creation of 'rc-local.service'.

- There are two folders : files and tasks
- In files, We have init-script(STARTUP SCRIPT) and rc-local.service
- In tasks, you will find main.yaml file
- First We, will see main.yaml configuration

→ Main.yaml

→ Here

```
--  
- name : Updating the packages  
hosts : all  
become: yes  
  
vars :  
- remote_install_path: /var/lib/jenkins/workspace/aws-infra-build/ansible-role-airflow/files/  
  
tasks:  
- name : update and upgrade apt packages  
apt :  
  upgrade : yes  
  update_cache : yes  
  
- name : install python  
apt :  
  name : python3-pip  
  state: latest  
  
- name : install sqlite3  
apt :  
  name : sqlite3  
  state : latest  
  
- name : install airflow  
pip :  
  name : apache-airflow  
  
- name : install postgres  
apt :  
  name : postgresql-contrib  
  state : latest  
  
- name : set the initialisation of startup script  
block :  
- name : copy init-script file  
copy :  
  src : "{{ remote_install_path }}/init-script.sh"  
  dest : /tmp/  
  owner : root  
  group : root  
  mode : '0775'  
  
- name : setting up the rc-local. service file
```

```

block :
- name : copy rc-local.service
copy :
src : "{{ remote_install_path }}/rc-local.service"
dest : /etc/systemd/system/
owner : root
group : root
mode : '0775'

- name : Create a rc.local file
copy :
dest : /etc/rc.local
content: |
#!/bin/bash

- name : Changing the permissions for rc.local
file :
path : /etc/rc.local
state : file
owner : root
group : root
mode : '0755'

- name : Enable a service
service :
name : rc-local
enabled : yes

- name : start the rc.local service
service :
name : rc-local
state: started

- name : Copy the content of init script to rc.local
copy :
src : "{{ remote_install_path }}/init-script.sh"
dest: /etc/rc.local
owner : root
group : root
mode : '0775'

- name : Message
debug :
msg : "Now, This machine will reboot. SSH connection will be disconnected, ignore the below error"

- name : rebooting the machine
command : reboot

```

```

=====
- name : Updating the packages
hosts : all
become: yes

vars :
- remote_install_path: /var/lib/jenkins/workspace/aws-infra-build/ansible-role-airflow/files/
→ This code is to configure the hosts. Here, we mentioned all. If you have any specific hosts mentioned in the inventory file, provide the name here in hosts.
→ Become: yes -> Means it will become root user
→ Vars : (means defining the variable here)
○ -variable-name : value
=====

tasks:
- name : update and upgrade apt packages
apt :
upgrade : yes
update_cache : yes

→ Tasks, These task are mentioned on what are the packages we are planning to install or setup through ansible
→ -name : (Provide the name of the task)
→ For ubutbu VM , we are generally updating the apt packages
→ FOR THIS COMMAND : $sudo apt update
=====

- name : install python
apt :
name : python3-pip
state: latest

```

→ This task will install PYTHON PIP
→ TASK FOR THIS COMMAND : **\$ sudo apt install python3-pip -y**
→ State: latest means, it will install latest python package

```
=====
- name : install sqlite3
  apt :
    name : sqlite3
    state : latest
```

→ This task will install SQLITE3 database required for airflow
→ COMMAND FOR THIS TASK : **\$ sudo apt install sqlite3 -y**

```
=====
- name : install airflow
  pip :
    name : apache-airflow
```

→ Installing APACHE-AIRFLOW using PIP
→ COMMAND FOR THIS TASK IS : **\$ pip install "apache-airflow[postgres]==2.5.1**

```
=====
- name : install postgre
  apt :
    name : postgresql-contrib
    state : latest
```

→ This will install postgresql database
→ COMMAND FOR THIS TASK IS : **\$ sudo apt install postgresql postgresql-contrib -y**

```
=====
- name : set the initialisation of startup script
  block :
    - name : copy init-script file
      copy :
        src : "{{ remote_install_path }}/init-script.sh"
        dest : /tmp/
        owner : root
        group : root
        mode : '0775'
```

→ This will copy the init-script.sh file in the path mentioned in the variable {{remote_install_path}} to destination "/tmp/" in the VM
→ Providing the permissions 0775
[The chmod 775 command will grant the read, write, and execute permissions to the owner of the file or directory and to the group associated to it. The '5' in this command will grant both read and execute permissions (4+1) to other users.]

```
=====
- name : setting up the rc-local. service file
  block :
    - name : copy rc-local.service
      copy :
        src : "{{ remote_install_path }}/rc-local.service"
        dest : /etc/systemd/system/
        owner : root
        group : root
        mode : '0775'
```

→ This will copy The rc-local.service file in the path mentioned in the variable {{remote_install_path}} to destination "/etc/systemd/system/" in the VM
→ Providing the permissions 0775

```
=====
- name : Create a rc.local file
  copy :
    dest : /etc/rc.local
    content: |
      #!/bin/bash
```

→ This will create a file (rc.local) in the path /etc/ and will paste the content #!/bin/bash

```
=====
- name : Changing the permissions for rc.local
  file :
    path : /etc/rc.local
    state : file
    owner : root
    group : root
    mode : '0755'
```

→ This will Provide the permission for the file rc.local

```
=====
- name : Enable a service
  service :
    name : rc-local
    enabled : yes
```

→ Enable the rc-local service

```
=====
- name : start the rc.local service
  service :
    name : rc-local
    state: started
```

→ Start the rc-local service

```
=====
- name : Copy the content of init script to rc.local
  copy :
    src : "{{ remote_install_path }}/init-script.sh"
    dest: /etc/rc.local
    owner : root
    group : root
    mode : '0775'
```

→ This will copy the content in the init-script.sh to the /etc/rc.local file

→ Providing the permissions for it

```
=====
- name : Message
  debug :
    msg : "Now, This machine will reboot. SSH connection will be disconnected, ignore the below error"
```

→ This will just print the message as we are rebooting the machine in next task, VM will get disconnected

```
=====
- name : rebooting the machine
  command : reboot
```

→ This will reboot the machine

→ INIT-SCRIPT.SH

```
#!/bin/bash

#IT WILL INITILIZE THE AIRFLOW DATABASE
airflow db init

#THESE ARE COMMANDS JUST FOR REFERENCE PURPOSE TO CREATE DB AND USERS (IGNORE THESE)
#sudo -i -u postgres
#sudo -i -u postgres psql -c "CREATE DATABASE airflow;"
#sudo -i -u postgres psql -c "CREATE USER airflow WITH PASSWORD 'airflow';"
#sudo -i -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE airflow TO airflow;"

#CHANGING THE DIRECTIORY. ALL AIRFLOW CONFIGURATION FILES WILL BE STORE IN THIS FOLDER
cd /root/airflow

#SEARCHING THE SQLALCHEMY_CONNECTION IN airflow.cfg file
grep sql_alchemy_conn airflow.cfg
#OUTPUT FOR THIS COMMAND
#sqlite:///root/airflow/airflow.db

#DECLARING IN THE BELOW VARIABLE FOR THIS COMMAND : grep "sqlalchemy_conn = /root/airflow/airflow.cfg"
airflow_initial_db=`grep "sqlalchemy_conn = /root/airflow/airflow.cfg"`

#REPLACING THE ALCHEMY DATABASE
echo "$airflow_initial_db" | sed 's/sqlalchemy_conn = //g'
database_value='echo "$airflow_initial_db" | sed 's/sqlalchemy_conn = //g'' 

sed -i '$database_value#postgresql+psycopg2://admin123:admin123a@airflow-db.cyn1tccpnjs5.us-east-1.rds.amazonaws.com/airflowdata#g' airflow.cfg
```

```
sed -i 's#sqlite:///home/ubuntu/airflow/airflow.db#postgresql+psycopg2://admin_123a:admin_123a@ airflow-
db.cj0ajqgnaw0.us-east-1.rds.amazonaws.com/airflowdata#g' airflow.cfg

#sed -i 's#SequentialExecutor#LocalExecutor#g' airflow.cfg

airflow db init

airflow users create -u airflow-1234 -f airflow -l airflow -r Admin -p airflow -e airflowi@gmail.com

airflow webserver &

airflow scheduler
```

RC LOCAL

Monday, June 26, 2023 11:34 PM

The rc.local script in some Linux distributions and Unix systems is a superuser startup script, usually located under the directory /etc/etc/rc.d. The file name rc refers to Run Control.

For UBUBTU, to run in RC LOCAL, you need to add this or configure this in your machine

Create a file with name : "rc-local.service"

Add this below code and save it

```
[Unit]
Description=/etc/rc.local Compatibility
ConditionPathExists=/etc/rc.local
[Service]
Type=forking
ExecStart=/etc/rc.local start
TimeoutSec=0
StandardOutput=tty
RemainAfterExit=yes
SysVStartPriority=99
[Install]
WantedBy=multi-user.target
```

MANUAL COMMANDS TO EXECUTE OR SETUP THE RC.LOCAL :

\$ ls
\$ sudo systemctl enable rc-local
<i>O rc-local.service - /etc/rc.local Compatibility Loaded: loaded (/lib/systemd/system/rc-local.service; static) Drop-In: /usr/lib/systemd/system/rc-local.service.d └─debian.conf Active: inactive (dead) Docs: man:systemd-rc-local-generator(8)</i> root@ip-172-31-94-84:/home/ubuntu# sudo systemctl enable rc-local <i>The unit files have no installation config (WantedBy=, RequiredBy=, Also=, Alias= settings in the [Install] section, and DefaultInstance= for template units). This means they are not meant to be enabled using systemctl.</i>

\$ sudo vim /etc/systemd/system/rc-local.service
--

<i>[Unit]</i> <i>Description=/etc/rc.local Compatibility</i> <i>ConditionPathExists=/etc/rc.local</i> <i>[Service]</i> <i>Type=forking</i> <i>ExecStart=/etc/rc.local start</i> <i>TimeoutSec=0</i> <i>StandardOutput=tty</i> <i>RemainAfterExit=yes</i>
--

```
SysVStartPriority=99
```

```
[Install]
```

```
WantedBy=multi-user.target
```

```
$ printf '%s\n' '#!/bin/bash' 'exit 0' | sudo tee -a /etc/rc.local
```

```
$ sudo chmod +x /etc/rc.local
```

```
$ sudo systemctl enable rc-local
```

Created symlink /etc/systemd/system/multi-user.target.wants/rc-local.service → /etc/systemd/system/rc-local.service.

```
$ sudo systemctl start rc-local.service
```

```
$ sudo systemctl status rc-local.service
```

● **rc-local.service - /etc/rc.local Compatibility**

Loaded: loaded (/etc/systemd/system/rc-local.service; enabled; vendor pres>

Drop-In: /usr/lib/systemd/system/rc-local.service.d

└──debian.conf

Active: active (exited) since Sun 2023-06-11 15:48:16 UTC; 14s ago

Process: 1416 ExecStart=/etc/rc.local start (code=exited, status=0/SUCCESS)

CPU: 2ms

Jun 11 15:48:16 ip-172-31-94-84 systemd[1]: Starting /etc/rc.local Compatibility>

Jun 11 15:48:16 ip-172-31-94-84 systemd[1]: Started /etc/rc.local Compatibility.

lines 1-10/10 (END)

```
$ Reboot
```

REFERENCE :

<https://linuxhint.com/use-rc-local-on-ubuntu/>

<https://www.linuxbabe.com/linux-server/how-to-enable-etcrc-local-with-systemd>

[How To Run A Command On StartUp in Linux | How to run a startup script](#)

SUBMODULES IN GIT

Monday, June 26, 2023 11:34 PM

The screenshot shows the GitHub repository 'terraform-aws' by 'kishandev1999'. The repository is private and has 148 commits. The commit history includes:

- kishandev1999 Update Jenkinsfile (Jun 18)
- ansible-role-airflow @ b07452f adding the modules (Jun 18)
- env Update main.tf (Jun 18)
- files Rename ec2-keys to ec2keys (Jun 18)
- modules Update sg.tf (Jun 18)
- scripts Delete cleanup.shg (Jun 18)
- .gitmodules adding the modules (Jun 18)
- Jenkinsfile Update Jenkinsfile (Jun 18)
- README.md Initial commit (Jun 18)

The repository has 1 unwatched, 0 forks, and 0 stars. It also has 1 watching and 0 forks.

To create a submodule, follow the below commands :

Consider 2 repositories :

1. First Repo is

The screenshot shows the GitHub repository 'terraform-aws' by 'kishandev1999'. The repository is private and has 148 commits. The repository has 1 unwatched, 0 forks, and 0 stars. It also has 1 watching and 0 forks.

2. Second Repo is

The screenshot shows the GitHub repository 'ansible-role-airflow' by 'kishandev1999'. The repository is private and has 0 commits. The repository has 1 unwatched, 0 forks, and 0 stars. It also has 1 watching and 0 forks.

Now we need to link 2nd repo (ansible-role-airflow) to the first repo (terraform-aws)

1. Create a folder in your local machine :
 2. Git clone the 1st repo (terraform-aws)
- asus@KishanDev MINGW64 /y/terraform aws latest

```
$ git clone https://github.com/kishandev1999/terraform-aws.git
```

Cloning into 'terraform-aws'...

remote: Enumerating objects: 276, done.

remote: Counting objects: 100% (155/155), done.

remote: Compressing objects: 100% (155/155), done.

remote: Total 276 (delta 88), reused 0 (delta 0), pack-reused 121

Receiving objects: 100% (276/276), 65.71 KiB | 1.88 MiB/s, done.

Resolving deltas: 100% (115/115), done.

hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead

```
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git submodule add https://github.com/kishandev1999/ansible-role-airflow.git
hint: core.useBuiltInFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
```

```
Cloning into 'Y:/terraform aws latest/terraform-aws/ansible-role-airflow'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
warning: LF will be replaced by CRLF in .gitmodules.
The file will have its original line endings in your working directory
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git add .
hint: core.useBuiltInFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git commit -m "adding the submodule ansible-role-airflow"
hint: core.useBuiltInFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
[master a5cf2ba] adding the submodule ansible-role-airflow
 2 files changed, 4 insertions(+)
 create mode 100644 .gitmodules
 create mode 160000 ansible-role-airflow
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 432 bytes | 432.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/kishandev1999/terraform-aws.git
 ff7d01b..a5cf2ba master -> master
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
```

Check the above commands :
The repo will automatically link to your git

REMOVING THE SUBMODULE :

```
$ git clone https://github.com/kishandev1999/terraform-aws.git
Cloning into 'terraform-aws'...
```

```
remote: Enumerating objects: 449, done.  
remote: Counting objects: 100% (3/3), done.  
remote: Compressing objects: 100% (3/3), done.  
remote: Total 449 (delta 0), reused 0 (delta 0), pack-reused 446  
s: 90% (405/449)  
Receiving objects: 100% (449/449), 101.97 KiB | 1.32 MiB/s, done.  
Resolving deltas: 100% (213/213), done.  
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead  
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
```

asus@KishanDev MINGW64 /y/terraform aws latest

```
$ ls  
terraform-aws/
```

asus@KishanDev MINGW64 /y/terraform aws latest
\$ cd terraform-aws/

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)

```
$ ls  
Jenkinsfile README.md ansible-role-airflow/ env/ files/ modules/ scripts/
```

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)

```
$ git submodule deinit -f ansible-role-airflow/
```

```
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead  
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
```

```
error: could not lock config file .git/modules/ansible-role-airflow/config: No such file or directory  
warning: Could not unset core.worktree setting in submodule 'ansible-role-airflow'  
Cleared directory 'ansible-role-airflow'
```

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)

```
$ git rm -rf ansible-role-airflow/
```

```
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead  
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"  
rm 'ansible-role-airflow'
```

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)

```
$ ls  
Jenkinsfile README.md env/ files/ modules/ scripts/
```

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)

```
$ rm -rf .git/modules/ansible-role-airflow/
```

asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)

```
$ git status
```

```
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead  
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"  
On branch master  
Your branch is up to date with 'origin/master'.
```

Changes to be committed:

```
(use "git restore --staged <file>..." to unstage)  
  modified: .gitmodules  
  deleted: ansible-role-airflow
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git add .
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git commit -m "removing the modules"
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
[master b325842] removing the modules
 2 files changed, 4 deletions(-)
 delete mode 160000 ansible-role-airflow
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 284 bytes | 284.00 KiB/s, done.
Total 3 (delta 1), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/kishandev1999/terraform-aws.git
 6bd3486..b325842 master -> master
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git submodule add https://github.com/kishandev1999/ansible-role-airflow.git
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"

Cloning into 'Y:/terraform aws latest/terraform-aws/ansible-role-airflow'...
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (38/38), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 38 (delta 10), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (38/38), 8.21 KiB | 4.10 MiB/s, done.
Resolving deltas: 100% (10/10), done.
warning: LF will be replaced by CRLF in .gitmodules.
The file will have its original line endings in your working directory
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ ls
Jenkinsfile README.md ansible-role-airflow/ env/ files/ modules/ scripts/
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git add .
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git commit -m "Adding the modules"
hint: core.useBuiltinFSMonitor will be deprecated soon; use core.fsmonitor instead
hint: Disable this message with "git config advice.useCoreFSMonitorConfig false"
[master 1bf6890] Adding the modules
 2 files changed, 4 insertions(+)
 create mode 160000 ansible-role-airflow
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git push origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.
```

Please make sure you have the correct access rights
and the repository exists.

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 417 bytes | 417.00 KiB/s, done.
Total 3 (delta 1), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/kishandev1999/terraform-aws.git
 b325842..1bf6890 master -> master
```

```
asus@KishanDev MINGW64 /y/terraform aws latest/terraform-aws (master)
```

JENKINS PIPELINE SCRIPT

Monday, June 26, 2023 11:34 PM

```
pipeline{
    agent any
    tools {
        terraform 'terraform-11'
    }
    parameters {
        choice(
            name : 'environment',
            choices : [
                'dev',
                'testing',
                'prod'
            ],
            description : 'Choose the environment you want to deploy: ')
        choice (
            name : 'application',
            choices : [
                'airflow',
                'tom-cat',
                'jenkins'
            ],
            description : 'Choose the application')
    }
}

//environment variables
environment {
    ENVIRONMENT = "${params.environment}"
    APP = "${params.application}"
    WORKSPACE = "${env.WORKSPACE}"
}
stages {

    // stage: 0
    stage("Clean up") {
        steps {
            withCredentials([
                $class: 'AmazonWebServicesCredentialsBinding',
                credentialsId: 'aws-jenkins-creds',
                ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
                SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
            ]) {
                dir("${WORKSPACE}/scripts") {
                    sh label: "", script: 'chmod +x cleanup.sh'
                    sh label: "", script: './cleanup.sh'
                }
            }
        }
    }

    // stage: 0.1
}
```

```

stage("Updating the submodule") {
    steps {
        withCredentials([gitUsernamePassword(credentialsId: 'git-cred-id',
            gitToolName: 'git-tool')]) {
            sh 'git submodule update --init --recursive'
        }
        //sh 'git config --global user.name "kishandev1999"'
        //sh 'git config --global user.email "kishandev005@gmail.com"'
        //sh 'git config -l'

    }
}

//stage: 1
stage("Terraform Init") {
    steps {
        withCredentials([
            $class: 'AmazonWebServicesCredentialsBinding',
            credentialsId: 'aws-jenkins-creds',
            ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
            SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
        ]) {
            dir("${WORKSPACE}/env/${ENVIRONMENT}") {
                sh "terraform init"
            }
        }
    }
}

//stage: 2
stage("Terraform apply") {
    steps {
        withCredentials([
            $class: 'AmazonWebServicesCredentialsBinding',
            credentialsId: 'aws-jenkins-creds',
            ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
            SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
        ]) {
            dir("${WORKSPACE}/env/${ENVIRONMENT}") {
                sh "terraform destroy --auto-approve"
            }
            //input(message: "Deploy to ${env.ENVIRONMENT} with application ${env.APP}?", ok :
'Deploy')
        }
    }
}

//stage : 3
stage("Configuring Selected Application") {
    steps {
        withCredentials([
            $class: 'AmazonWebServicesCredentialsBinding',
            credentialsId: 'aws-jenkins-creds',
            ACCESS_KEY: 'AWS_ACCESS_KEY_ID',
            SECRET_KEY: 'AWS_SECRET_ACCESS_KEY'
        ]) {
            script {
                if ("${APP}" == "airflow") {
                    dir("${WORKSPACE}/scripts")
                }
            }
        }
    }
}

```

