Jenkins

Jenkins offers a simple way to set up a continuous integration and continuous delivery environment for almost any combination of languages and source code repositories using pipelines. It gives us a faster and more robust way to integrate our entire chain of build, test, and deployment tools than we can easily build ourselves.

Generally Jenkins is installed in 3 ways : Either as a container in Docker, either in a dedicated namespace in Kubernetes cluster, or directly on our local system.

Note : By default it operates on port 8080 as jenkins runs on Tomcat which by default starts at this port, and port 50000 is used by jenkins for master-slave communicate on this port, so this will enable jenkins to bind slave with master in case we had some. One more thing to add, for data persistence we need to bind a volume.

Installing Jenkins : Docker

“docker run -d -p 8080:8080 -p 50000:50000 -v docker_home2:/var/jenkins_home jenkins/jenkins:latest”

- This command will start a jenkins container using its official image “jenkins/jenkins:latest” with respective versions.
- In this we have exposed port 8080 and 50000 as mentioned above.
- For data persistence we will bind a volume “-v docker_home2:/var/jenkins_home”, “docker_hwill be automaome” repo tically created at “/var/lib/docker/docker_home” and it has been bind to our “jenkins_home” repo inside our jenkins container.

Installing Jenkins : Ubuntu

Prerequisites:-

Jenkins requires Java in order to run, yet certain distributions don’t include this by default and some Java versions are incompatible with Jenkins.

Update the Debian apt repositories, install OpenJDK 11

- sudo apt update -y
- sudo apt install default-jdk

Note : we need to install openjdk-11 or above, as some Java versions are incompatible with Jenkins.

First, add the repository key to your system:

- “wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key |sudo gpg --dearmor -o /usr/share/keyrings/jenkins.gpg”

The gpg --dearmor command is used to convert the key into a format that apt recognizes. (gpg is called with the --dearmor flag to convert the PGP key into a GPG file format, with -o used to indicate the file output. On Ubuntu, the /usr/share/keyrings directory is the recommended location for your converted GPG files, as it is the default location where Ubuntu stores its keyrings.
)

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

- sudo sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

Now, we will run apt update so that apt will use the new repository.

- “sudo apt update”

Finally, install Jenkins and its dependencies:

- “sudo apt install jenkins”

We are done with the installation part, now we need to start jenkins (if required) and Opening the Firewall (if required)

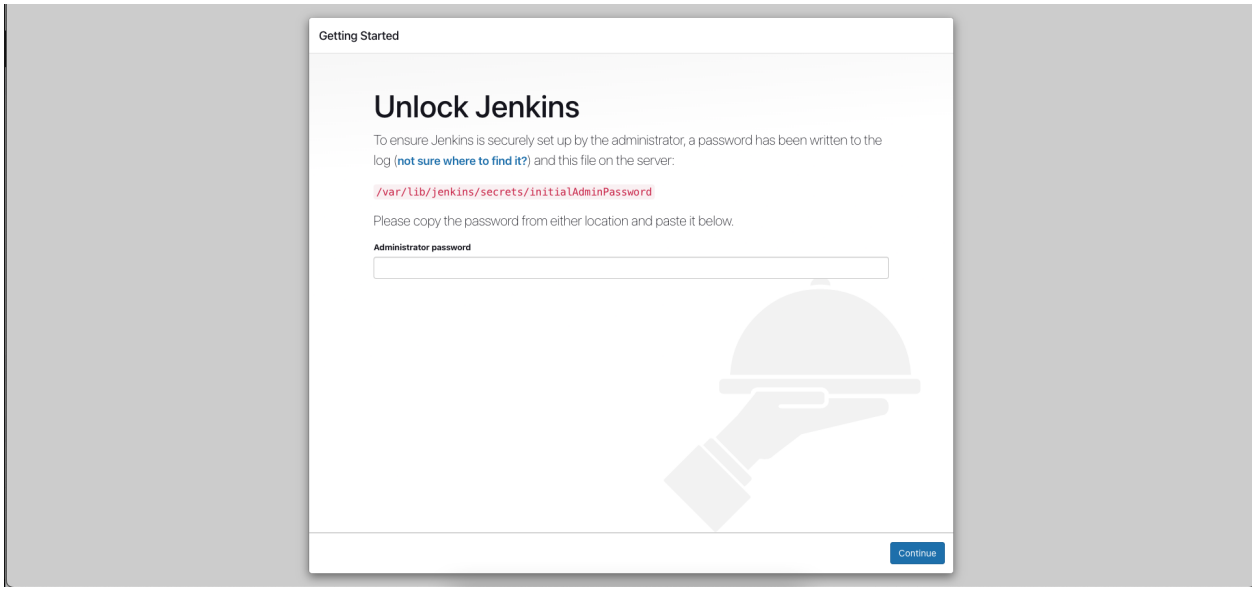
- sudo systemctl status jenkins
- sudo systemctl start jenkins.service

- sudo ufw status
- sudo ufw allow 8080
- sudo ufw allow OpenSSH
- sudo ufw enable

- sudo ufw status

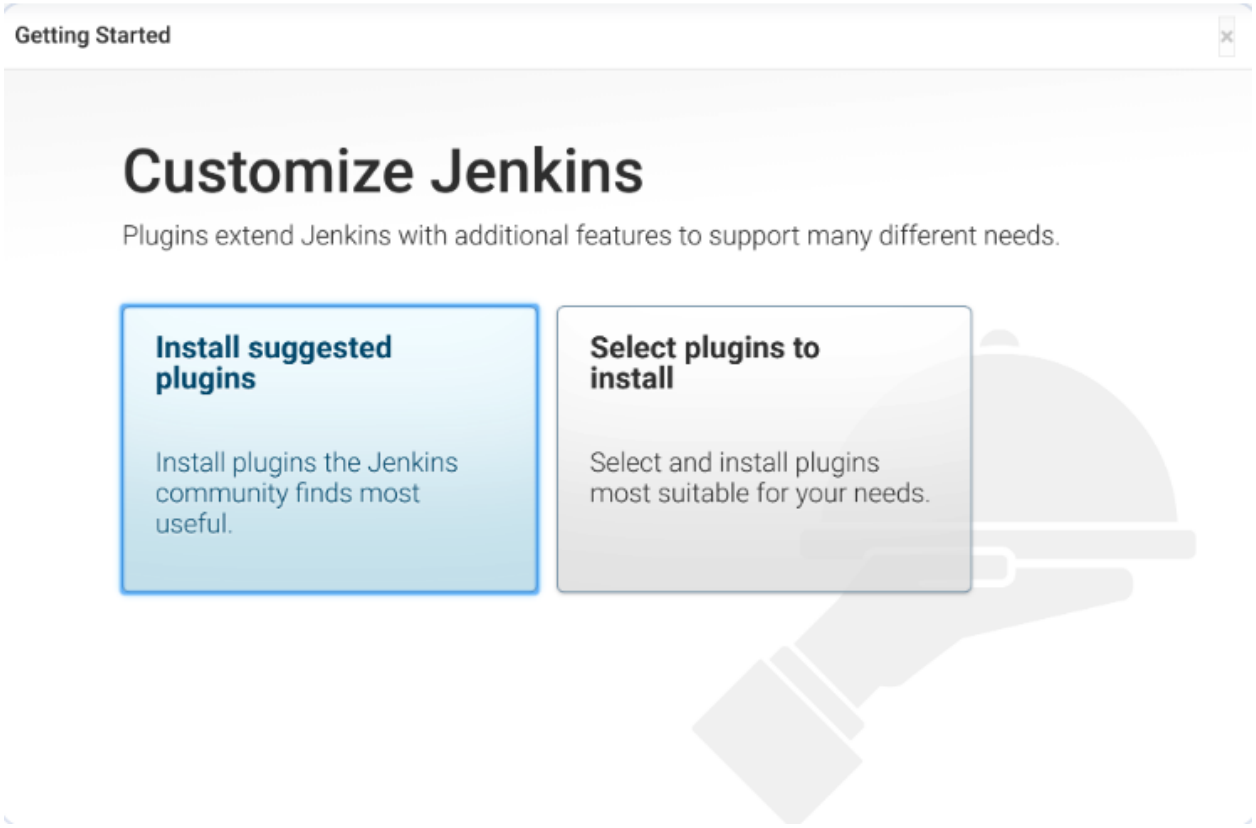
Note : Now our jenkins server is up and running at port 8080.

Setting Up Jenkins :-

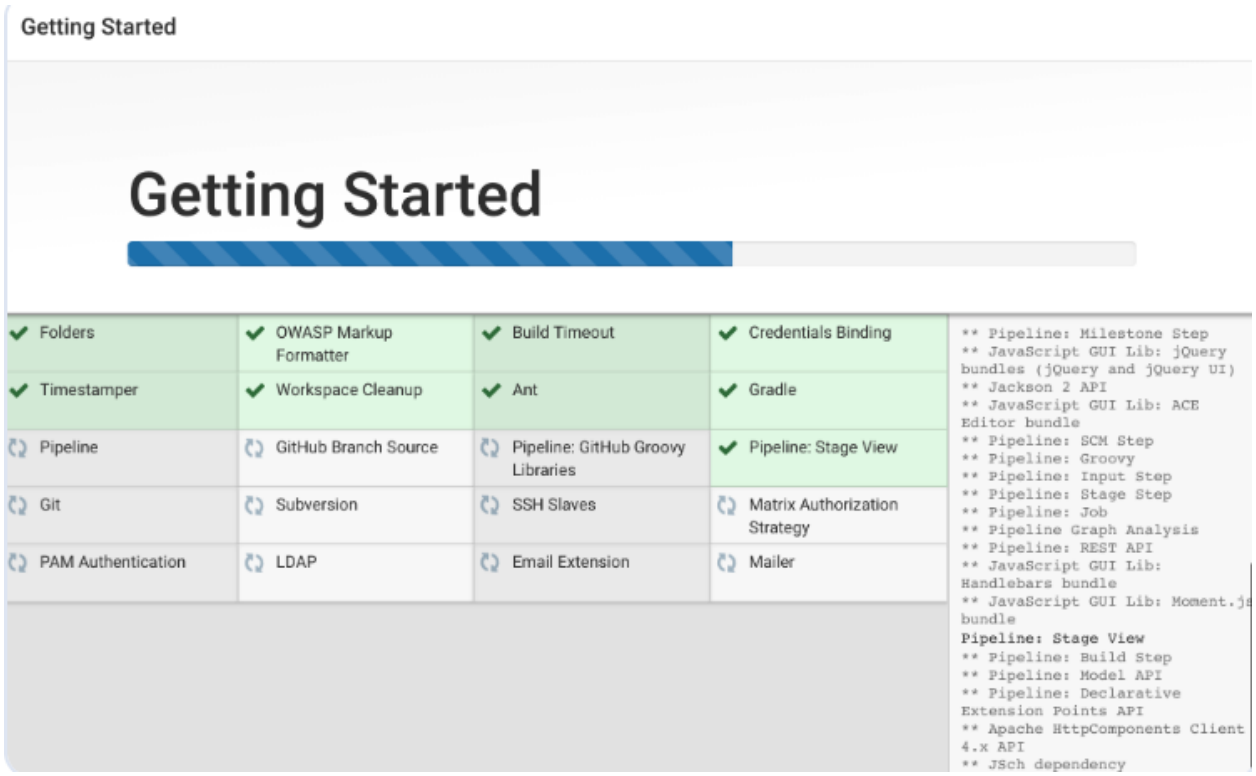


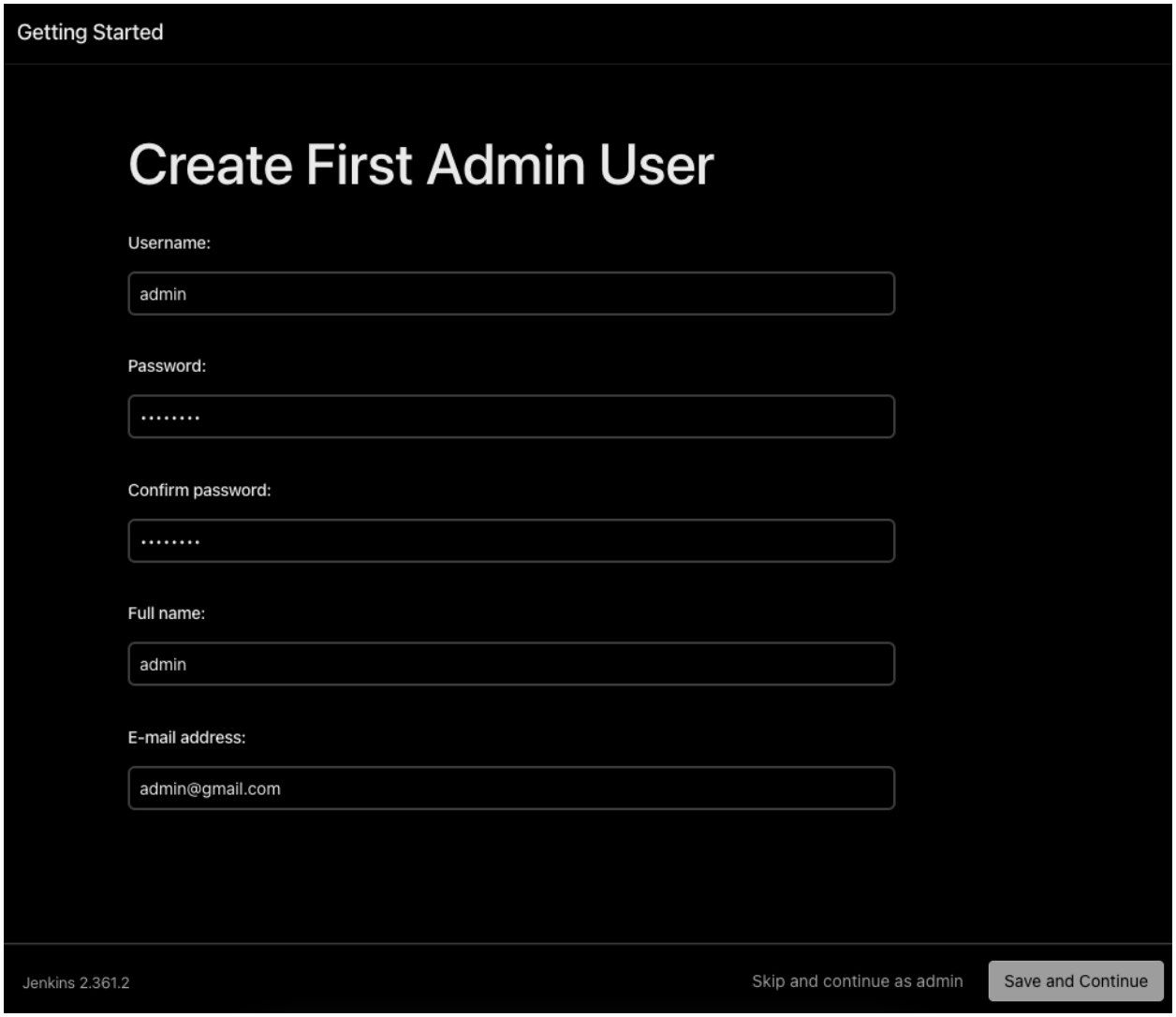
We need to fetch our password from the given path.

- sudo cat `/var/lib/jenkins/secrets/initialAdminPassword`



It will start installing all suggested plugins on its own.

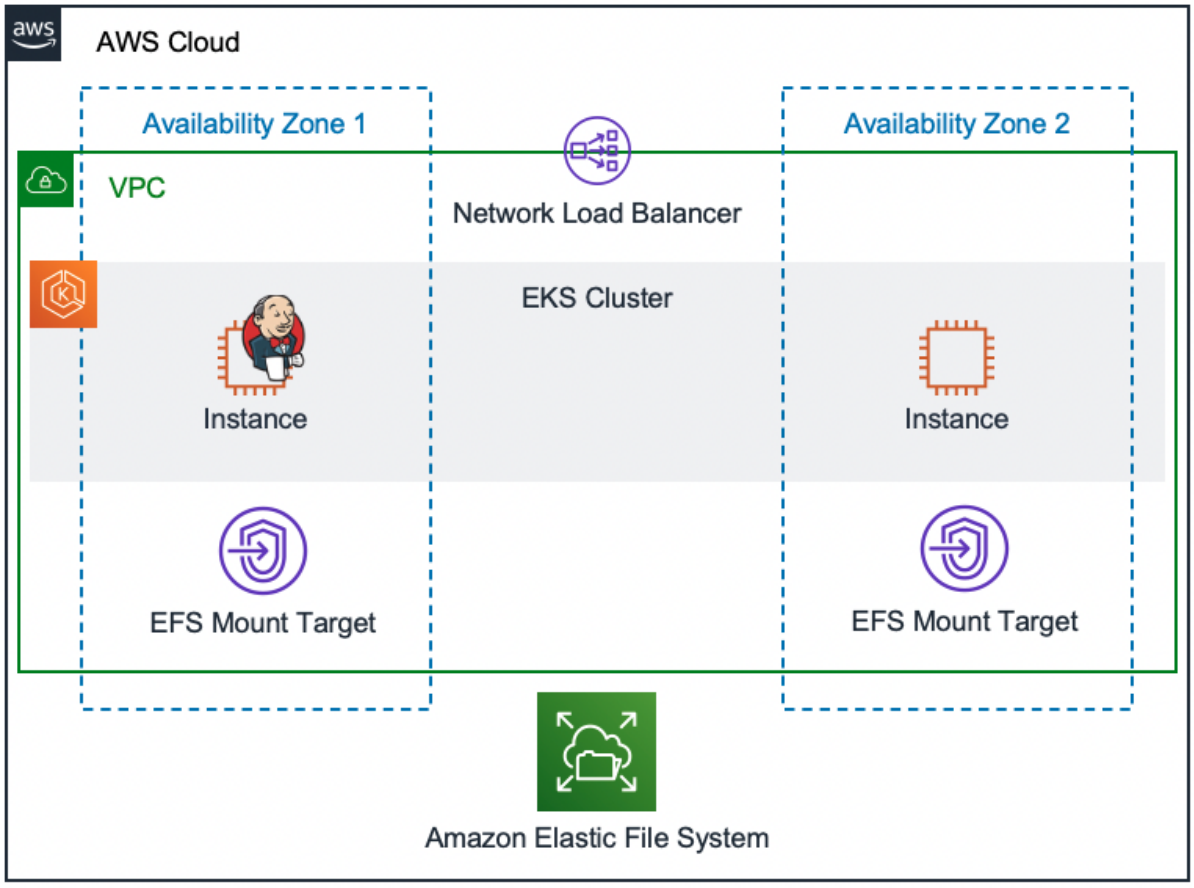




Enter username, password and fill up admin details.

Installing Jenkins : Kubernetes Cluster

We will deploy Jenkins on Amazon EKS with Amazon EFS, to provide elastic, durable, and persistent shared storage to containerized applications running in Amazon EKS.



By default, Jenkins stores application configuration, job configurations, change logs, and past build records in the $JENKINS_HOME directory. Configuring $JENKINS_HOME to point to a shared Amazon EFS file system enables us to quickly create a new Jenkins pod in another AWS Availability Zone in the event of node failure or pod termination. Additionally, having a shared file system enables us to further architect our Jenkins deployment for scale.

For this we need to follow these steps :-

- Create an Amazon EKS cluster
- Create an Amazon EFS file system
- Deploy the Amazon EFS CSI Driver to your Amazon EKS cluster

- Deploy Jenkins to Amazon EKS
- Simulate a Kubernetes node failure
- Check for persisted state thanks to Amazon EFS!

In our case we already had our kubernetes cluster created, but if it is not then we can use eksctl to create one.

- eksctl create cluster --name <cluster name> --region <region name> --zones us-east-1a,us-east-1b --managed --nodegroup-name mynodegroup

Create an Amazon EFS file system

Now, we will create a security group, Amazon EFS file system, two mount targets, and an EFS Access Point. The access point allows the Amazon EKS cluster to use the EFS file system.

Firstly we need to find the vpc id

- aws ec2 describe-vpcs

Create a security group for your Amazon EFS mount target

- aws ec2 create-security-group \
  --region < region name > \
  --group-name efs-mount-sg \
  --description "Amazon EFS for EKS, SG for mount target" \
  --vpc-id < vpc id >

Add rules to the security group to authorize inbound/outbound access

- aws ec2 authorize-security-group-ingress \
  --group-id < security group id > \
  --region < region name > \
  --protocol tcp \
  --port 2049 \
  --cidr 192.168.0.0/16

Authorize inbound access to the security group for the Amazon EFS mount target (efs-mount-sg) to allow inbound traffic to the NFS port (2049) from the VPC CIDR block

Note: Network File Sharing (NFS) is a protocol that allows you to share directories and files with other Linux clients over a network.

create an encrypted Amazon EFS file system

- aws efs create-file-system \
  --creation-token creation-token \
  --performance-mode generalPurpose \
  --throughput-mode bursting \
  --region < region name > \
  --tags Key=Name,Value=MyEFSFileSystem \
  --encrypted

Capture your VPC subnet IDs

To create mount targets, you must have subnet IDs for my node group. Use the following command to find and record the subnets IDs

- aws ec2 describe-instances --filters Name=vpc-id,Values= identifier for our VPC (i.e. vpc-1234567ab12a345cd) --query 'Reservations[*].Instances[].SubnetId'

Create two Amazon EFS mount targets
Now that we have the security group, file system ID, and subnets, we can create mount targets in each of the Availability Zones using the following command

- aws efs create-mount-target \
  --file-system-id <file system id> (i.e. fs-123b45fa) \
  --subnet-id <subnet id> \
  --security-group <sg id> \
  --region < region name >

Note: Be sure to create a mount target in each of the Availability Zones!

Create an Amazon EFS access point

We can create access points using aws console as well, which is a more suitable way of creating.

Amazon EFS access points are application-specific entry points into an EFS file system that make it easier to manage application access to shared datasets or, in our case, configuration.

- aws efs create-access-point --file-system-id identifier for our file system (i.e. fs-123b45fa) \
  --posix-user Uid=1000,Gid=1000 \
  --root-directory
  "Path=/jenkins,CreationInfo={OwnerUid=1000,OwnerGid=1000,Permissions=77}"

Next step is to Deploy the Amazon EFS CSI driver to your Amazon EKS cluster

- kubectl apply -k
  "github.com/kubernetes-sigs/aws-efs-csi-driver/deploy/kubernetes/overlays/stable/?ref=master"

Note : Create efs-sc storage class YAML file, efs-pv persistent volume YAML file, efs-claim persistent volume claim YAML file

storage-class.yaml

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
```

persistent-volume.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: jenkins-efs-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: efs-sc
  csi:
    driver: efs.csi.aws.com
    volumeHandle: fs-058707c5e54ab0bbc
```

persistent-volume-claim.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: jenkins-efs-pvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 5Gi
```

Now, Deploy the efs-sc storage class, efs-pv persistent volume, and efs-claim persistent volume claim

- kubectl apply -f storageclass.yaml, persistentvolume.yaml, persistentvolumeclaim.yaml

Final step is to Deploy Jenkins to Amazon EKS, we will deploy Jenkins using helm.

- helm repo add jenkins https://charts.jenkins.io
- helm repo update
- helm install <RELEASE_NAME> jenkins/jenkins

With this final step our Jenkins is up and running on localhost:8080.