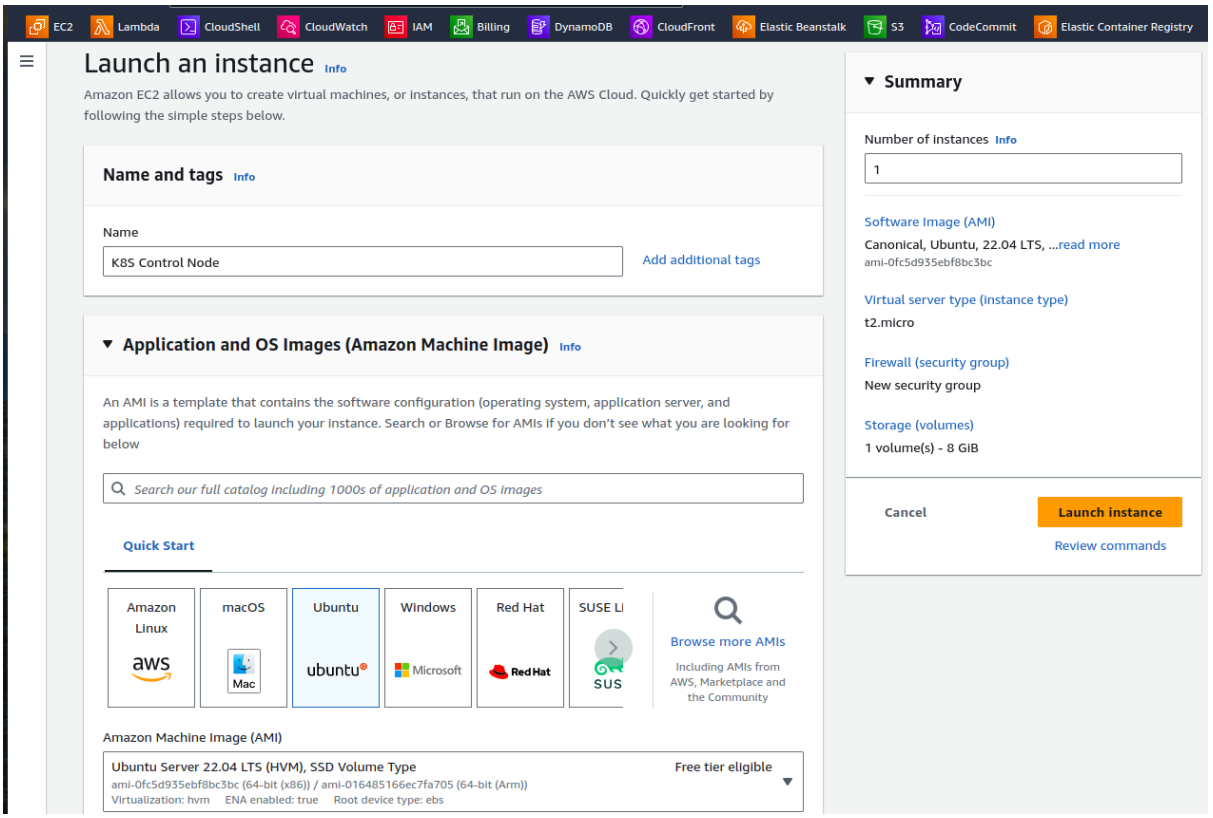
Create a Kubernetes Cluster And Deploy Jenkins, Using Elastic Kubernetes Service(EKS) :

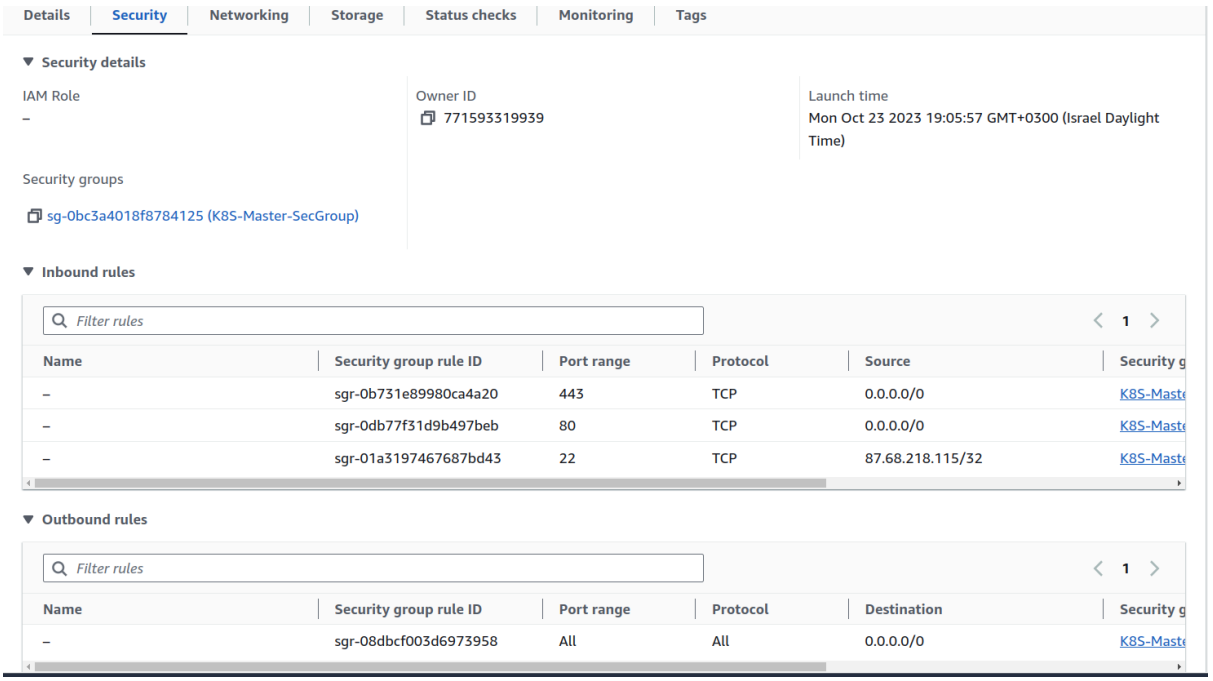
EC2 Control Node Instance Type:

t2.micro
AMI: Ubuntu.

Security Group Settings:

- Ports to Open:
  - 22 for SSH
  - 80 for HTTP
  - 443 for HTTPS





SSH into EC2 Instance

ssh -i "your-key.pem" ubuntu@<EC2_PUBLIC_IP>

Change Host Name(Optional for comfort)

sudo hostnamectl set-hostname k8s-master
echo "127.0.0.1 k8s-master" | sudo tee -a /etc/hosts
(close the current SSH session and open a new one.
The changes should take effect)

Update Packages

sudo apt-get update

Install AWS CLI

sudo apt install awscli -y

Configure AWS CLI

Run the “aws configure” command(Here you'll be prompted to enter details such as):

- AWS Access Key ID: Your AWS Access Key.
- AWS Secret Access Key: Your AWS Secret Access Key.
- Default region name: The region where you want to create the EKS cluster.
- Default output format: You can leave this as ‘None’

Install EKSCTL

curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname
-s)_amd64.tar.gz" | tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin

*verify the installation
eksctl version

Install Kubectl

sudo apt-get update
sudo apt-get install -y apt-transport-https
sudo curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a
/etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl

Create an EKS Cluster

eksctl create cluster \
--name “cluster-name” (You'll use this name to refer to the cluster) \
--region “your-region” \
--nodes 1 \
--nodegroup-name “group-name” \
--node-type “instance-type” (e..g t2/t3.medium) \
--managed(simplifies node management.)

```
2023-10-23 18:18:57 [ℹ]  creating EKS cluster "devops-cluster" in "us-east-1" region with managed nodes
2023-10-23 18:18:57 [ℹ]  will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup
2023-10-23 18:18:57 [ℹ]  if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=us-east-1 --cluster=devo
ps-cluster'
2023-10-23 18:18:57 [ℹ]  Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "devops-cluster" in "us-
east-1"
2023-10-23 18:18:57 [ℹ]  CloudWatch logging will not be enabled for cluster "devops-cluster" in "us-east-1"
2023-10-23 18:18:57 [ℹ]  you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=us-
east-1 --cluster=devops-cluster'
2023-10-23 18:18:57 [ℹ]
2 sequential tasks: { create cluster control plane "devops-cluster",
    2 sequential sub-tasks: {
        wait for control plane to become ready,
        create managed nodegroup "devops-nodes",
    }
}
2023-10-23 18:18:57 [ℹ]  building cluster stack "eksctl-devops-cluster-cluster"
2023-10-23 18:18:57 [ℹ]  deploying stack "eksctl-devops-cluster-cluster"
2023-10-23 18:19:27 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:19:57 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:20:58 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:21:58 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:22:58 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:23:58 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:24:58 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:25:58 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:26:58 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:27:58 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:28:58 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-cluster"
2023-10-23 18:30:59 [ℹ]  building managed nodegroup stack "eksctl-devops-cluster-nodegroup-devops-nodes"
2023-10-23 18:30:59 [ℹ]  deploying stack "eksctl-devops-cluster-nodegroup-devops-nodes"
2023-10-23 18:30:59 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-nodegroup-devops-nodes"
2023-10-23 18:31:29 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-nodegroup-devops-nodes"
2023-10-23 18:32:25 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-nodegroup-devops-nodes"
2023-10-23 18:33:15 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-nodegroup-devops-nodes"
2023-10-23 18:34:17 [ℹ]  waiting for CloudFormation stack "eksctl-devops-cluster-nodegroup-devops-nodes"
2023-10-23 18:34:17 [ℹ]  waiting for the control plane to become ready
2023-10-23 18:34:17 [✓]  EKS cluster "devops-cluster" in "us-east-1" region is ready
```

the output should be EKS cluster "cluster-name" in "region name" region is ready.

Confirm the cluster creation

eksctl get cluster --name ‘cluster-name --region “your region”

```
ubuntu@k8s-master:~$ eksctl get cluster --name devops-cluster --region us-east-1
NAME            VERSION STATUS  CREATED                 VPC                     SUBNETS
ECURITYGROUPS           PROVIDER
devops-cluster  1.27    ACTIVE  2023-10-23T19:50:30Z    vpc-025f7501f93fcbb6f   subnet-
2,subnet-0ac2d4118f68f1212      sg-09292cd9ec15b3fe6    EKS
```

Verify that kubectl is properly communicating with your new cluster

kubectl version

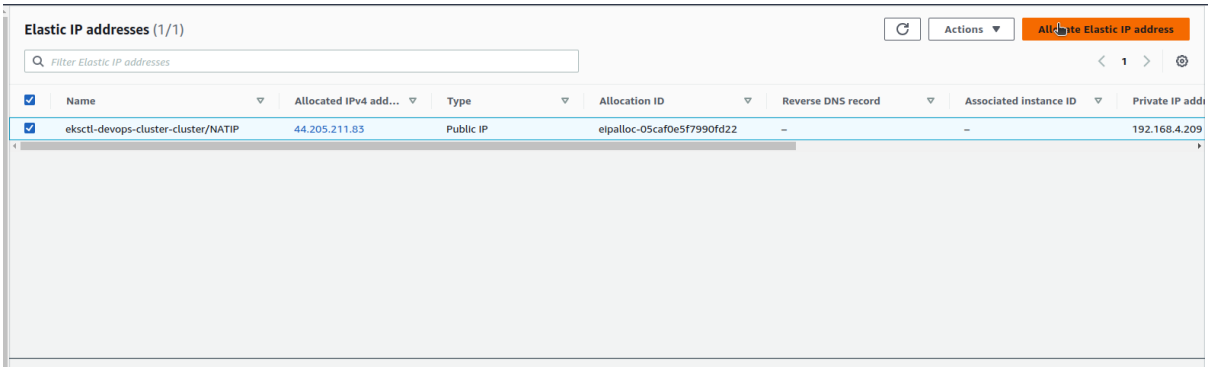
```
ubuntu@k8s-master:~$ kubectl version
Client Version: v1.28.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
Server Version: v1.27.6-eks-f8587cb
```

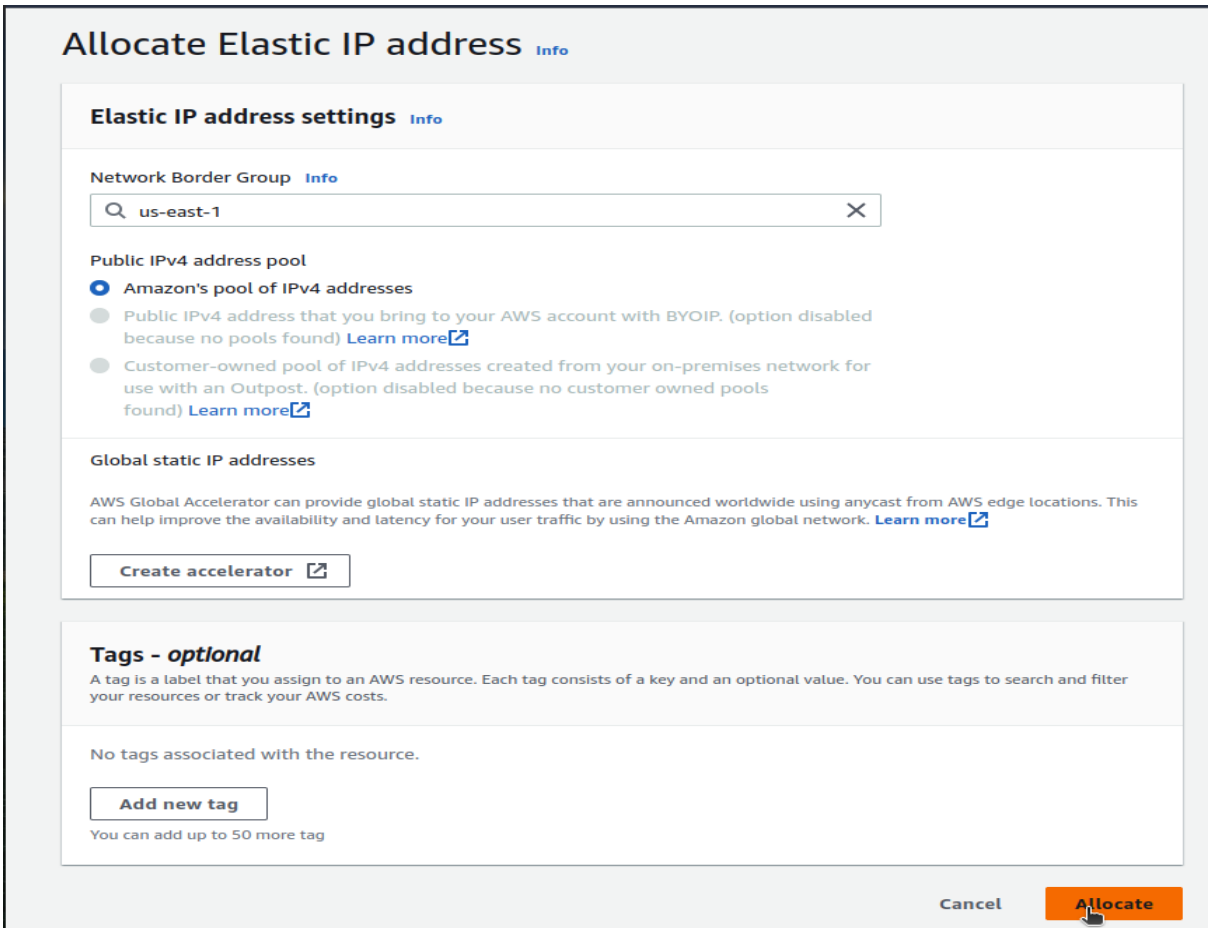
Create ‘devops’ namespace

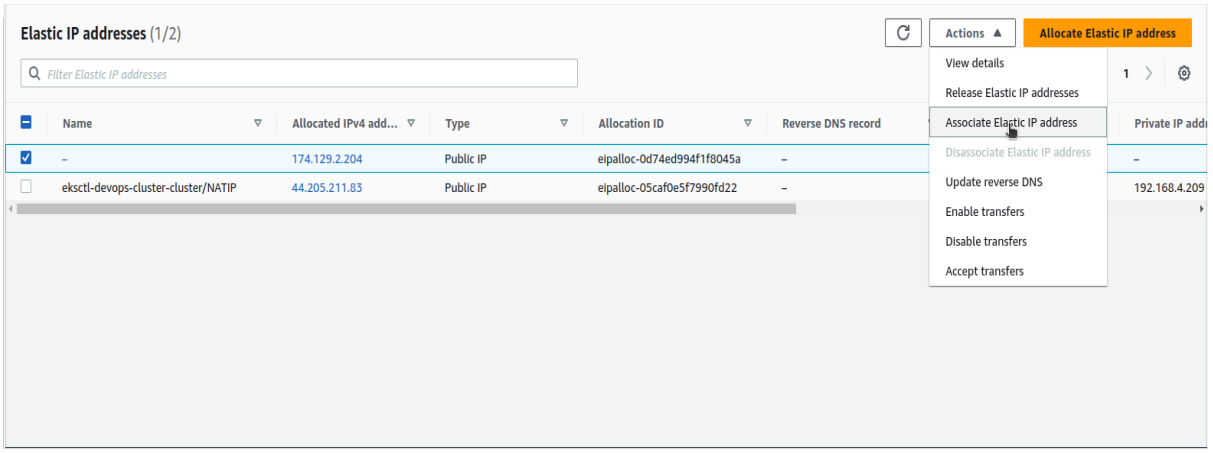
kubectl create namespace devops

Allocating An Elastic IP

Navigate to EC2 -> Elastic IP’s->Allocate Elastic IP Address -> Allocate -> Choose the New IP, Actions, Associate elastic ip address ->Choose “Network Interface”, The Instance Network Interface, His Private IP-> Associate


Elastic IP addresses (1/1)
Actions
Allocate Elastic IP address
Filter Elastic IP addresses
Name
Allocated IPv4 add...
Type
Allocation ID
Reverse DNS record
Associated instance ID
Private IP addr
eksctl-devops-cluster-cluster/NATIP
44.205.211.83
Public IP
eipalloc-05caf0e5f7990fd22
–
192.168.4.209


Allocate Elastic IP address Info
Elastic IP address settings Info
Network Border Group Info
us-east-1
Public IPv4 address pool
Amazon's pool of IPv4 addresses
Public IPv4 address that you bring to your AWS account with BYOIP. (option disabled because no pools found) Learn more
Customer-owned pool of IPv4 addresses created from your on-premises network for use with an Outpost. (option disabled because no customer owned pools found) Learn more
Global static IP addresses
AWS Global Accelerator can provide global static IP addresses that are announced worldwide using anycast from AWS edge locations. This can help improve the availability and latency for your user traffic by using the Amazon global network. Learn more
Create accelerator
Tags - optional
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.
No tags associated with the resource.
Add new tag
You can add up to 50 more tag
Cancel
Allocate


Elastic IP addresses (1/2)
Actions
Allocate Elastic IP address
Filter Elastic IP addresses
View details
Release Elastic IP addresses
Associate Elastic IP address
Disassociate Elastic IP address
Update reverse DNS
Enable transfers
Disable transfers
Accept transfers
Name
Allocated IPv4 add...
Type
Allocation ID
Reverse DNS record
Private IP addr
–
174.129.2.204
Public IP
eipalloc-0d74ed994f1f8045a
eksctl-devops-cluster-cluster/NATIP
44.205.211.83
Public IP
eipalloc-05caf0e5f7990fd22
192.168.4.209

EC2 > Elastic IP addresses > Associate Elastic IP address

Associate Elastic IP address Info

Choose the instance or network interface to associate to this Elastic IP address (174.129.2.204)

Elastic IP address: 174.129.2.204

Resource type
Choose the type of resource with which to associate the Elastic IP address.

- ○ Instance
- ◉ Network interface

⚠ If you associate an Elastic IP address with an instance that already has an Elastic IP address associated, the previously associated Elastic IP address will be disassociated, but the address will still be allocated to your account. Learn more

If no private IP address is specified, the Elastic IP address will be associated with the primary private IP address.

Network interface

eni-054f6ee95e0e7dab6

Private IP address
The private IP address with which to associate the Elastic IP address.

192.168.1.33

Reassociation
Specify whether the Elastic IP address can be reassociated with a different resource if it already associated with a resource.

☐ Allow this Elastic IP address to be reassociated

Cancel Associate

Create a Jenkins Dockerfile With necessary installations

Dockerfile

home > ori > Desktop > Dockerfile

```
# Switch to root to install Docker and sudo
USER root

# Update the package index and install sudo
RUN apt-get update && \
    apt-get install -y sudo && \
    echo 'jenkins ALL=(ALL) NOPASSWD: ALL' >> /etc/sudoers

# Update the package index and install Docker
RUN apt-get update && \
    apt-get install -y apt-transport-https ca-certificates curl software-properties-common && \
    curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add - && \
    add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable" && \
    apt-get update && \
    apt-get install -y docker-ce

# ...
# Add Jenkins user to the docker group
RUN if getent group docker > /dev/null; then \
    echo "The docker group already exists."; \
  else \
    groupadd docker; \
  fi && \
  usermod -aG docker jenkins
# ...

# Switch back to the Jenkins user
USER jenkins
```

You have Docker installed on you
install the recommended extensi

Build the Image: Navigate to the directory where the Dockerfile is located, then build the Docker image:

docker build -t <jenkins-image> .
docker tag <jenkins-image> <username>/<repo-name:<tag>
docker push <username>/<repo-name:<tag>

```
ori@OriElias:~/Desktop$ docker build -t jenkins-image
ERROR: "docker buildx build" requires exactly 1 argument.
See 'docker buildx build --help'.

Usage:  docker buildx build [OPTIONS] PATH | URL | -

Start a build
ori@OriElias:~/Desktop$ docker build -t jenkins-image .
[+] Building 7.4s (7/7) FINISHED                                docker:default
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 131B
 => [internal] load metadata for docker.io/jenkins/jenkins:lts
 => [auth] jenkins/jenkins:pull token for registry-1.docker.io
 => CACHED [1/2] FROM docker.io/jenkins/jenkins:lts@sha256:80587de2dac2
 => [2/2] RUN apt-get update && apt-get install -y sudo
 => exporting to image
 => => exporting layers
 => => writing image sha256:1c13a55d0e309d18efde7331f88c4446aa4d7ffcde2
 => => naming to docker.io/library/jenkins-image
ori@OriElias:~/Desktop$ docker tag jenkins-image orielias/jenkins-image:latest
ori@OriElias:~/Desktop$ docker push orielias/jenkins-image:latest
The push refers to repository [docker.io/orielias/jenkins-image]
f0d828a9dc8f: Pushed
f20a0c613bdc: Mounted from jenkins/jenkins
599f5f8d0ab0: Mounted from jenkins/jenkins
4c8b96ce47c2: Mounted from jenkins/jenkins
44fc60778258: Mounted from jenkins/jenkins
5f7c5aca7eca: Mounted from jenkins/jenkins
c91efd33f121: Mounted from jenkins/jenkins
e68a7d6d6b63: Mounted from jenkins/jenkins
9cbc357f747e: Mounted from jenkins/jenkins
a9a7c0db4243: Mounted from jenkins/jenkins
965c6f166415: Mounted from jenkins/jenkins
78880e985c37: Mounted from jenkins/jenkins
2fa37f2ee66e: Mounted from jenkins/jenkins
latest: digest: sha256:55149de1332c5e6899969cceb059b8eb27bdf5604b0d526ff47b52d8d46f551f size: 3045
ori@OriElias:~/Desktop$
```

Create a Jenkins persistent volume YAML File

change values : to your “node name”

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-pv
spec:
  capacity:
    storage: 10Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: /var/jenkins-data
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
            - ip-192-168-0-18.ec2.internal
```

kubectl apply -f persistent-volume.yaml

Create a Jenkins persistent volume claim YAML File

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: local-pvc
  namespace: devops
spec:
  storageClassName: local-storage
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

kubectl apply -f persistent-volume-claim.yaml

Create a Jenkins persistent local-storage YAML File

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
```

kubectl apply -f local-storage.yaml

Expose the Jenkins Service

A NodePort service will open a specific port on all the cluster nodes, allowing external traffic to reach the service. (we specified 30000)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: jenkins
  namespace: devops
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30000 # Optional
  selector:
    app: jenkins
```

Apply the Service

kubectl apply -f jenkins-service.yaml

the output should be : “service/jenkins created”

Create a Jenkins Deployment YAML File

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
  namespace: devops
spec:
  replicas: 1
  selector:
    matchLabels:
      app: jenkins
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      containers:
      - name: jenkins
        image: orielias/jenkins-image:latest
        ports:
        - containerPort: 8080
        volumeMounts:
          - mountPath: "/var/jenkins_home"
            name: jenkins-home
      volumes:
        - name: jenkins-home
          persistentVolumeClaim:
            claimName: local-pvc
```

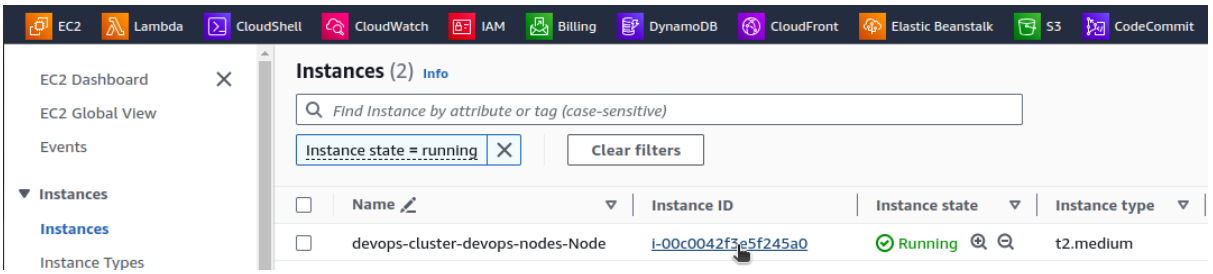
Apply the Deployment

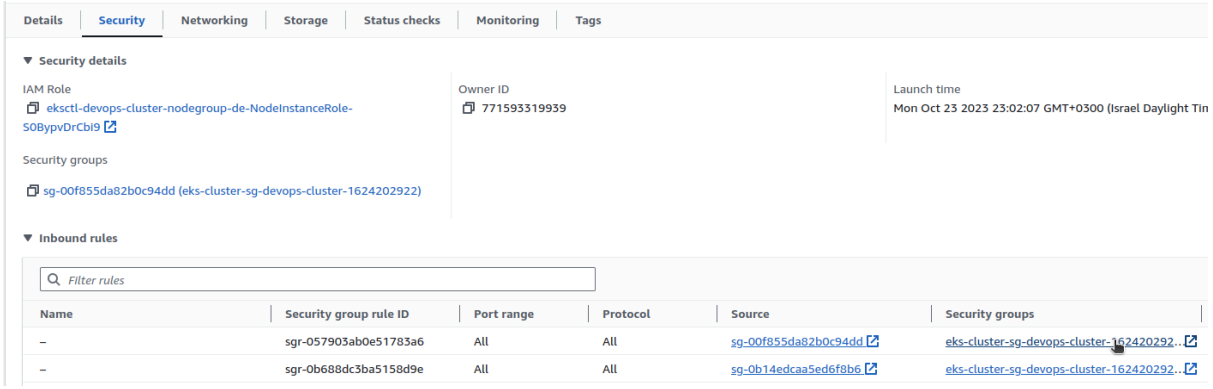
kubectl apply -f jenkins-deployment.yaml

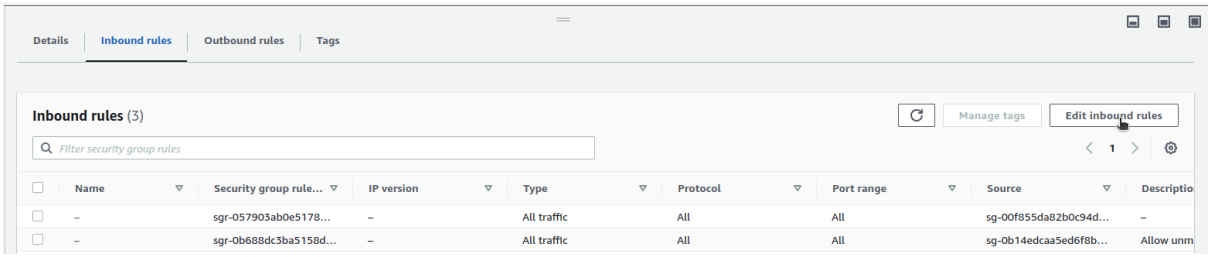
the output should be : “deployment.apps/jenkins created”

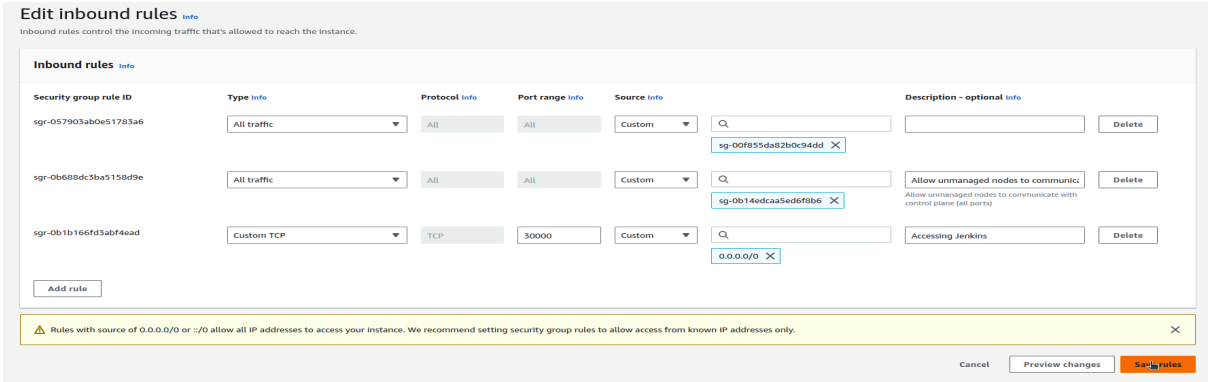
Allow Traffic on Port 30000

Navigate to: EC2 -> Instances -> Click on Instance ID (worker node) -> Security -> Security groups -> Inbound rules -> -> Edit inbound rules -> Add rule (custom TCP, Port Range - 30000, CIDR - 0.0.0.0/0) -> Save rules





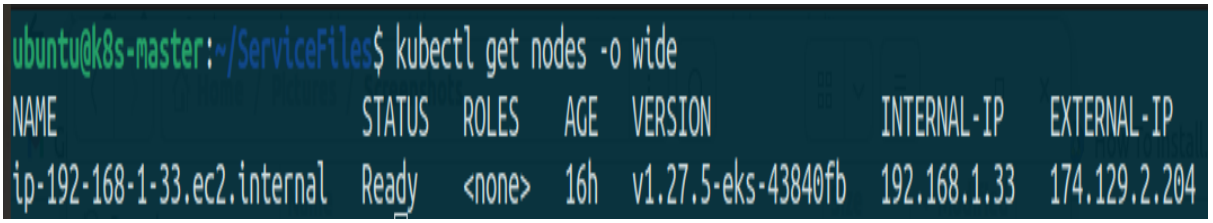




Access the Jenkins Website (nodeIP:NodePort)

to find the nodeIP use : kubectl get nodes -o wide

copy the EXTERNAL-IP



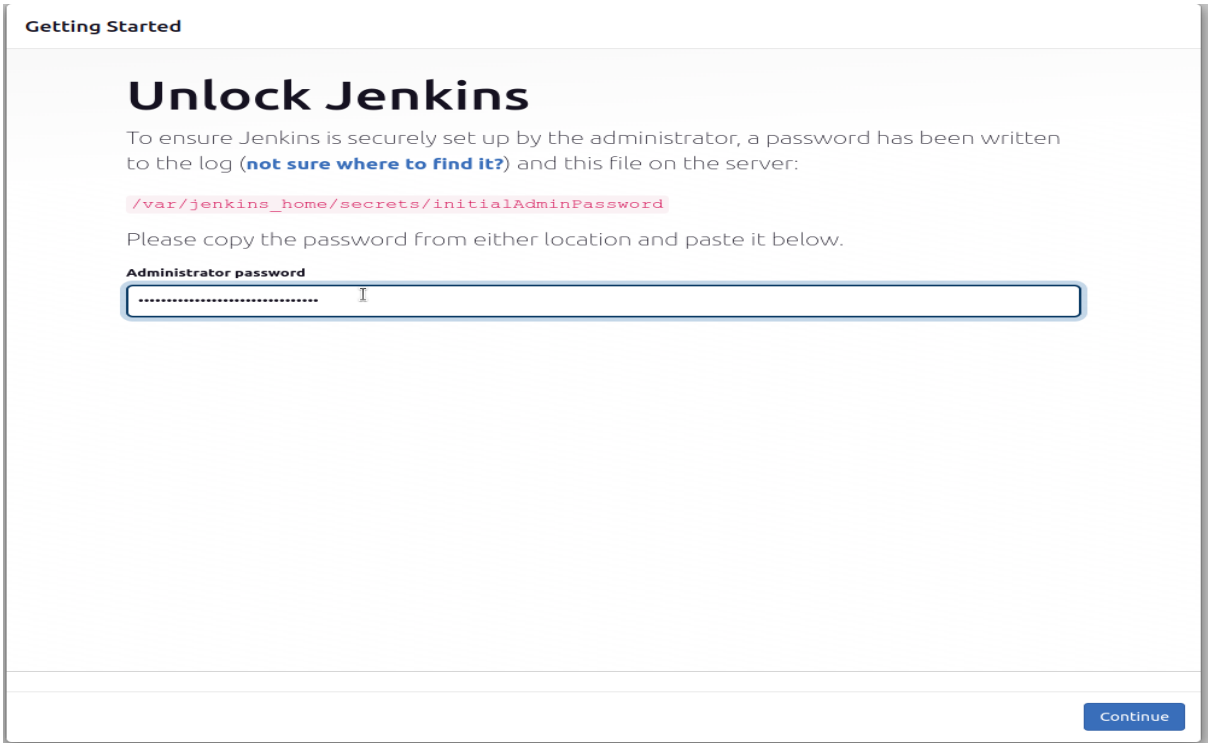
```
ubuntu@k8s-master:~/ServiceFiles$ kubectl get nodes -o wide
NAME                          STATUS   ROLES    AGE   VERSION          INTERNAL-IP    EXTERNAL-IP
ip-192-168-1-33.ec2.internal  Ready    <none>   16h   v1.27.5-eks-43840fb  192.168.1.33   174.129.2.204
```

now reach the jenkins website through : EXTERNAL-IP:30000

Retrieve Jenkins Unlock Key

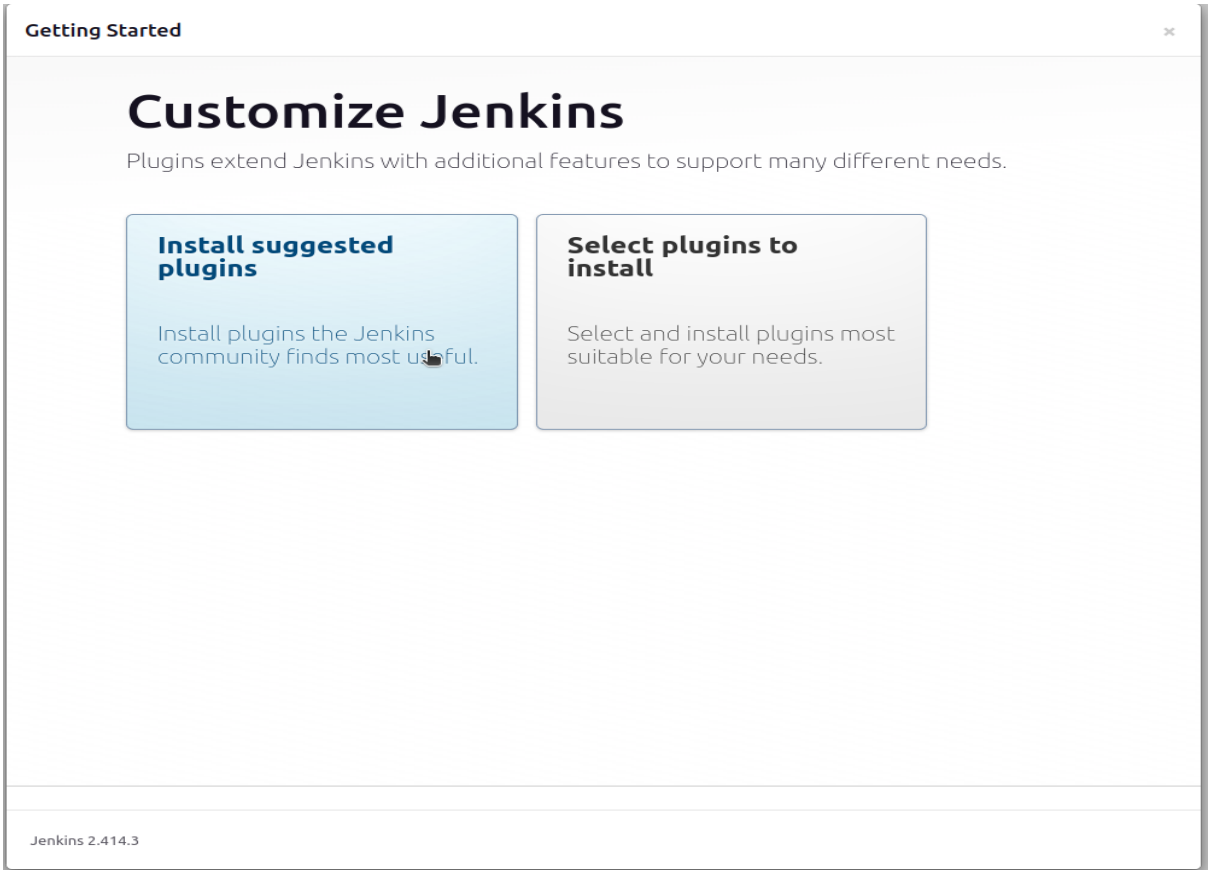
kubectl exec -it $(kubectl get pod -n devops -l app=jenkins -o jsonpath="{.items[0].metadata.name}") -n devops -- cat /var/jenkins_home/secrets/initialAdminPassword

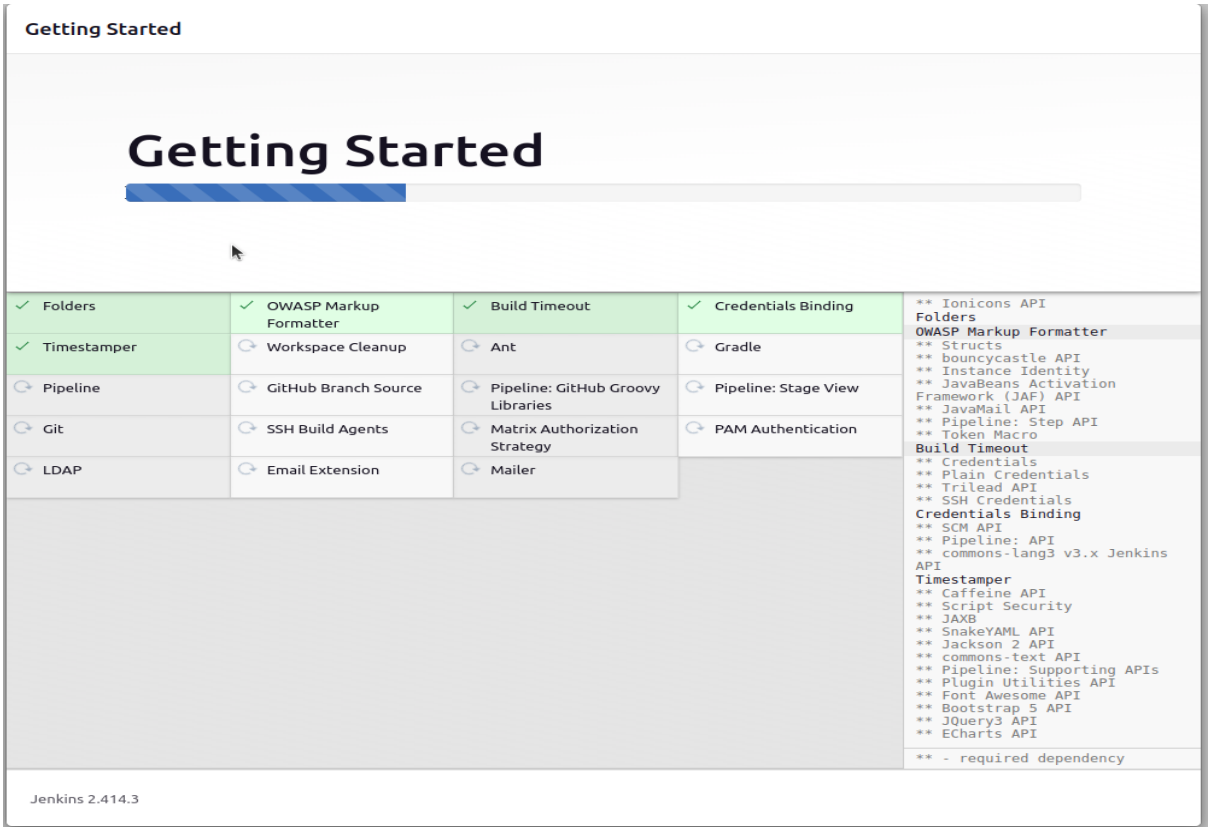
copy the output and press 'continue'



Install plugins

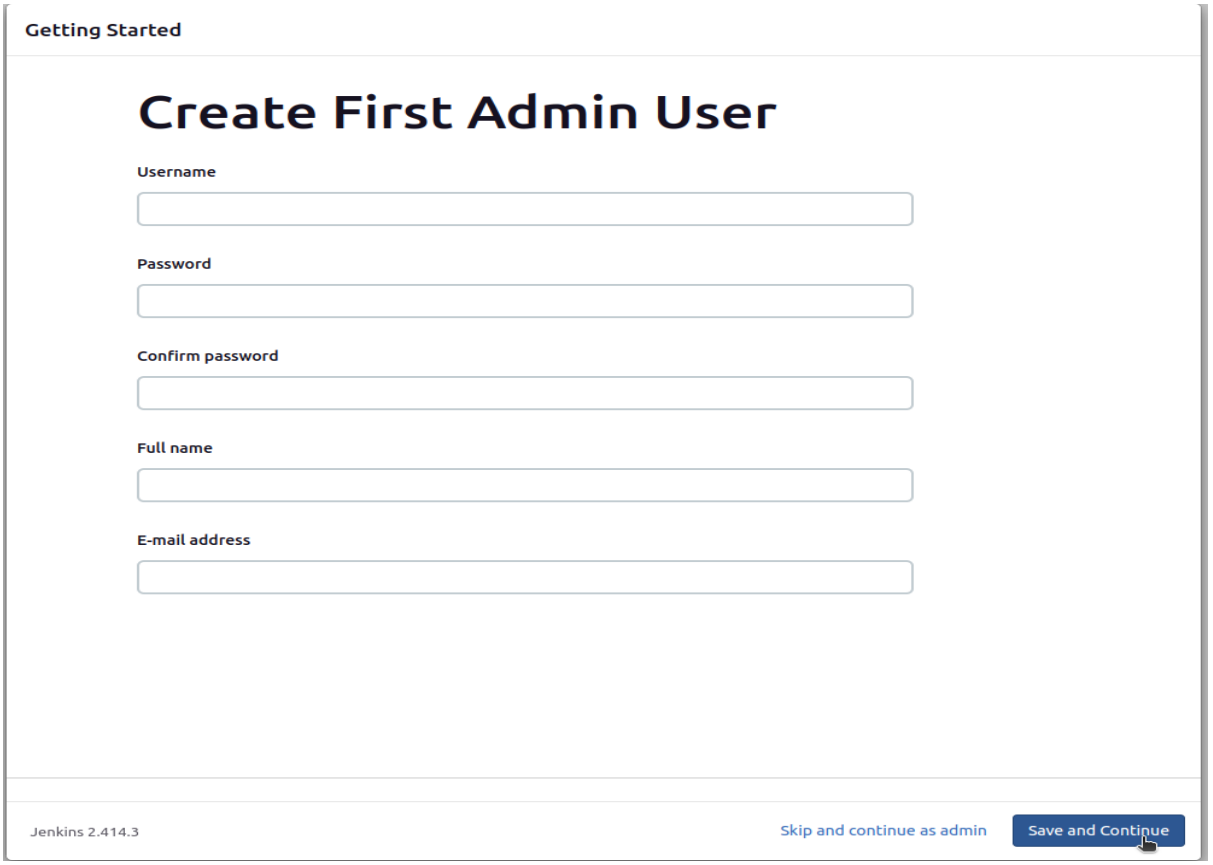
you can always install additional plugins later.



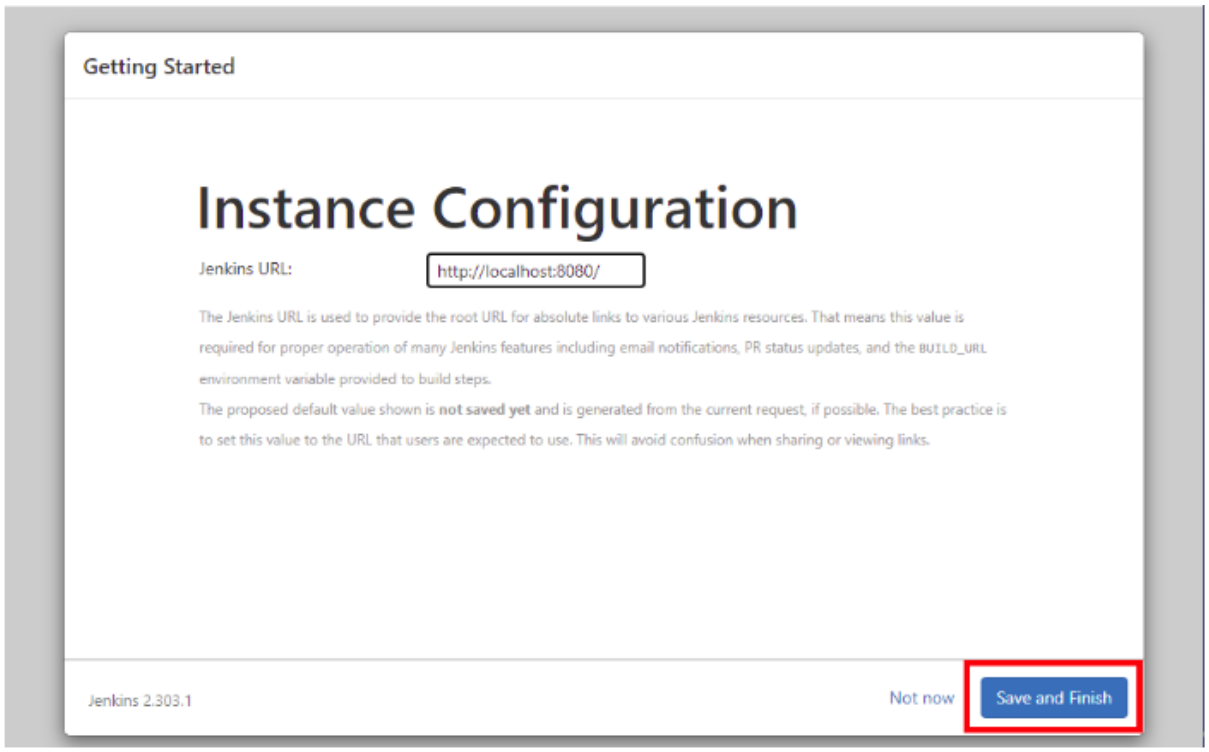


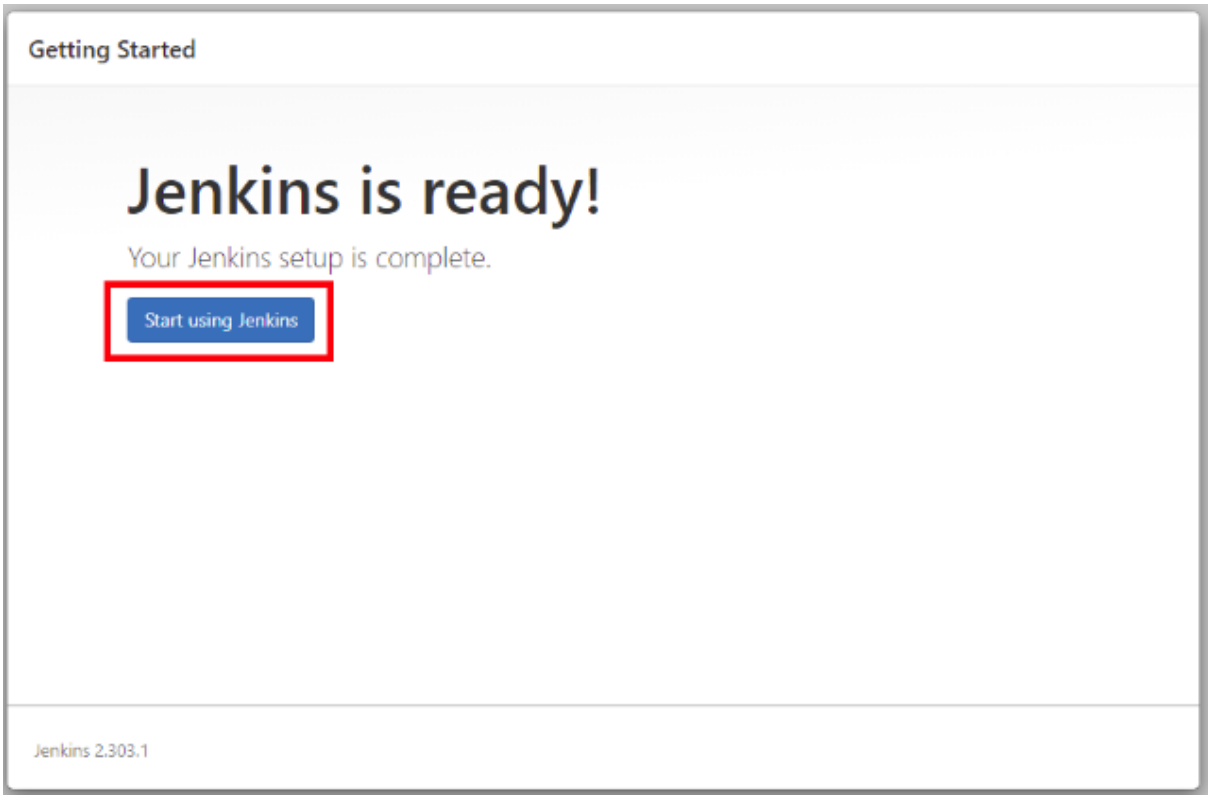
Create Admin user

fill the fields :



enter the public ip here `http:///"public-ip":30000/`





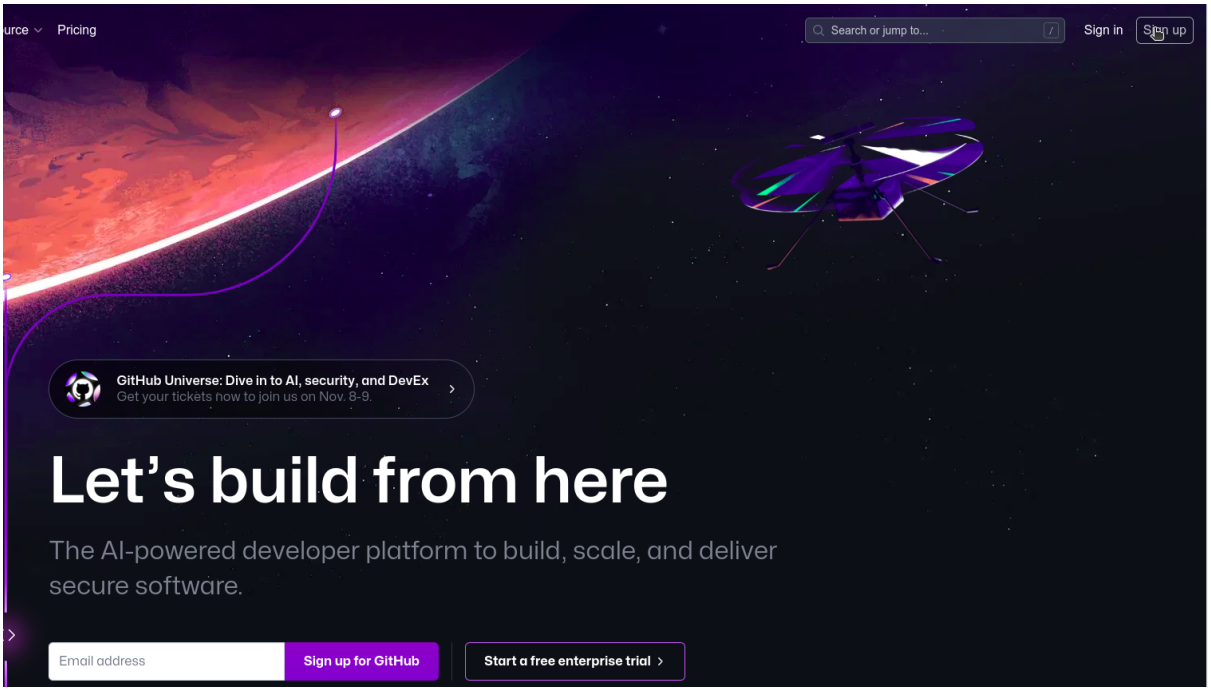
That's it! Jenkins is now ready to use

Configure jenkins to work with GitHub :

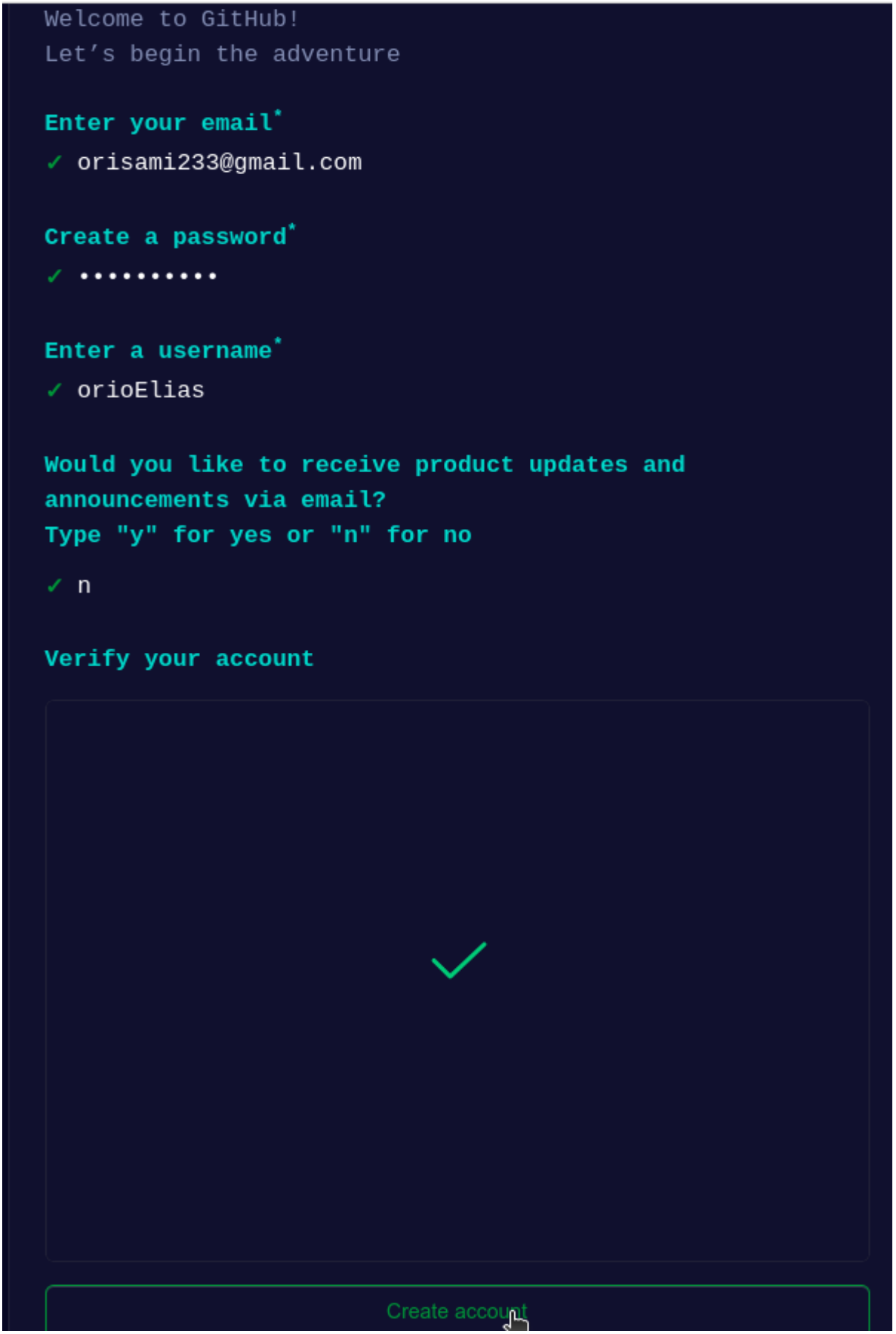
Create GitHub Account

go to : https://github.com

Click on “Sign Up”



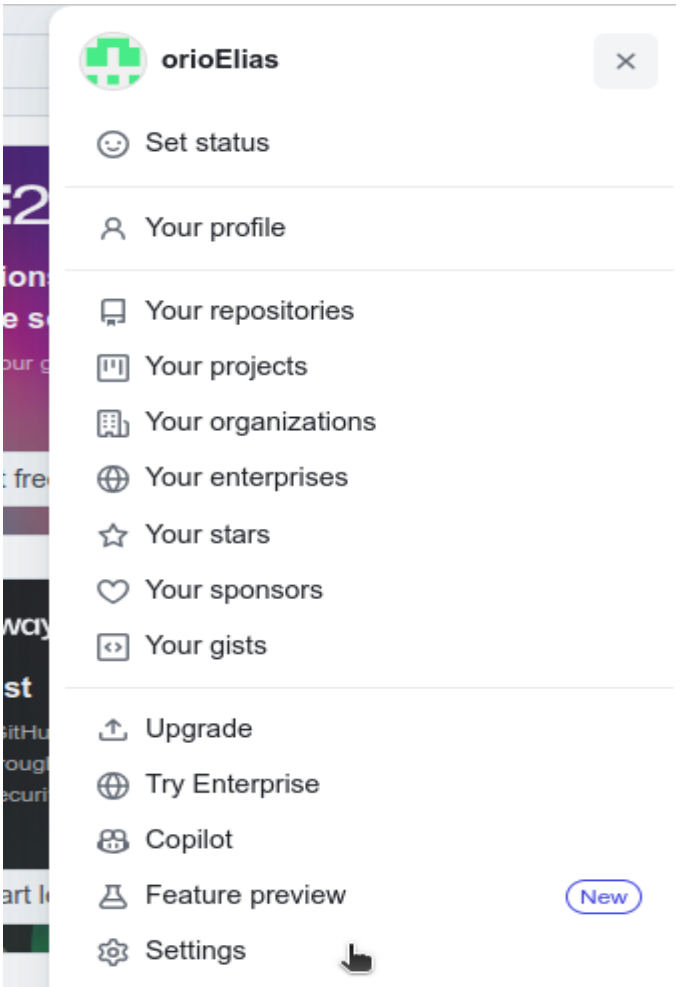
Fill the Fields:



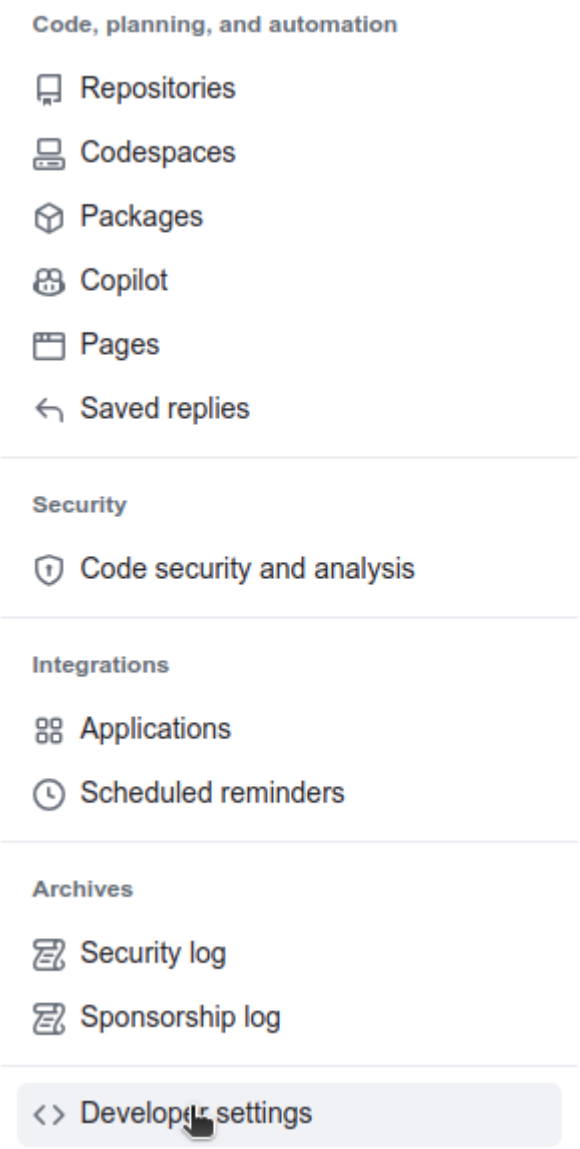
Copy The Password that sent to your email

Create Personal Access Token (PAT) on GitHub:

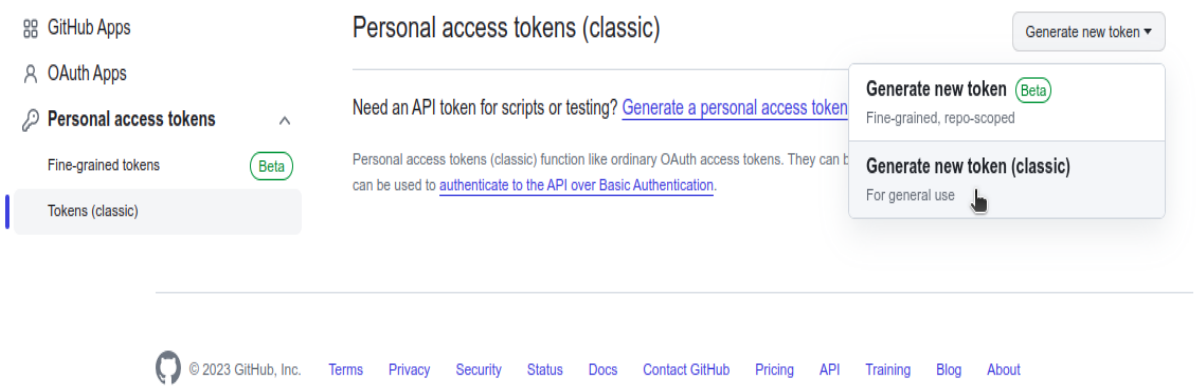
Click on profile picture at top-right and choose “Settings”.



Scroll Down And choose “Developer Settings”



Click on "Personal access tokens" from the left sidebar, then click on "Generate new token.



Name the token and give it the necessary scopes (permissions)

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note

Jenkins Token

What's this token for?

Expiration *

90 days — The token will expire on Mon, Jan 22 2024

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

☑ **repo**	Full control of private repositories
☑ repo:status	Access commit status
☑ repo_deployment	Access deployment status
☑ public_repo	Access public repositories
☑ repo:invite	Access repository invitations
☑ security_events	Read and write security events
☐ **workflow**	Update GitHub Action workflows
☐ **write:packages**	Upload packages to GitHub Package Registry
☐ read:packages	Download packages from GitHub Package Registry
☐ **delete:packages**	Delete packages from GitHub Package Registry
☐ **admin:org**	Full control of orgs and teams, read and write org projects
☐ write:org	Read and write org and team membership, read and write org projects
☐ read:org	Read org and team membership, read org projects
☐ manage_runners:org	Manage org runners and runner groups
☐ **admin:public_key**	Full control of user public keys
☐ write:public_key	Write user public keys
☐ read:public_key	Read user public keys
☑ **admin:repo_hook**	Full control of repository hooks
☑ write:repo_hook	Write repository hooks
☑ read:repo_hook	Read repository hooks
☑ **admin:org_hook**	Full control of organization hooks

Click "Generate token" at the bottom

Generate token Cancel

Copy the generated token somewhere safe; you won't be able to see it again.
(its cut for safety reasons)

Personal access tokens (classic)

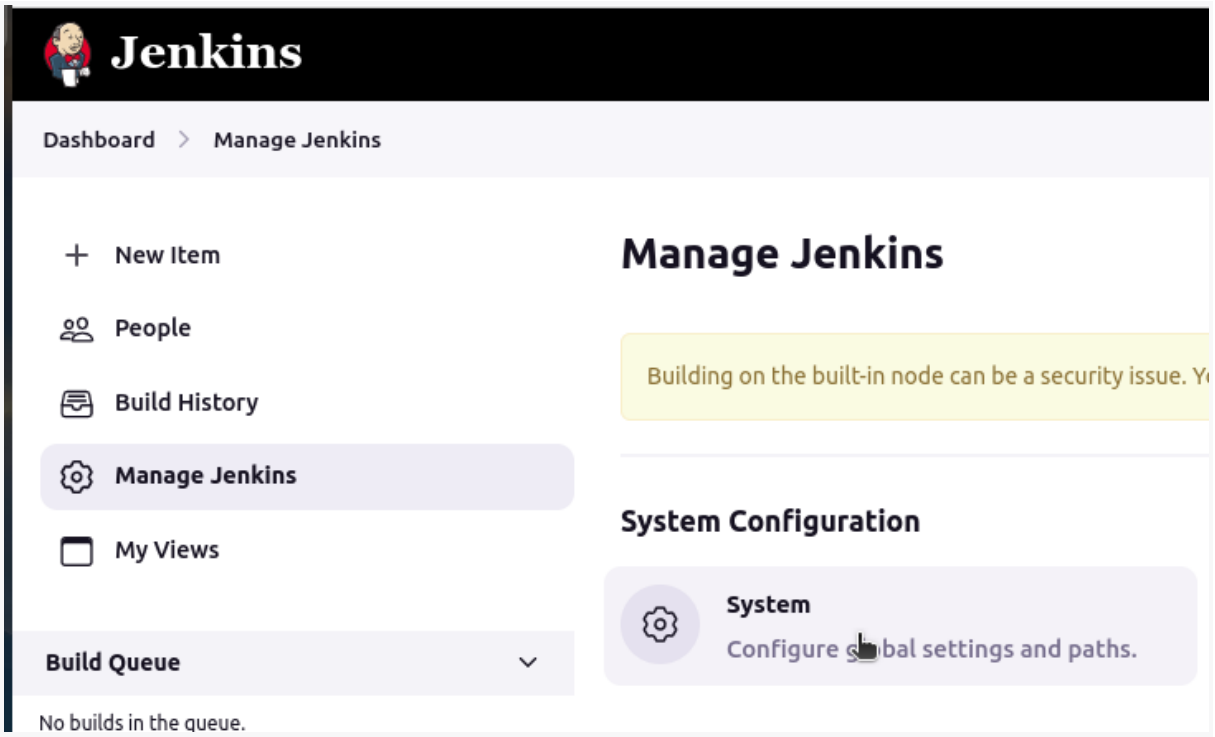
Generate new token Revoke all

Tokens you have generated that can be used to access the GitHub API.

Make sure to copy your personal access token now. You won't be able to see it again!

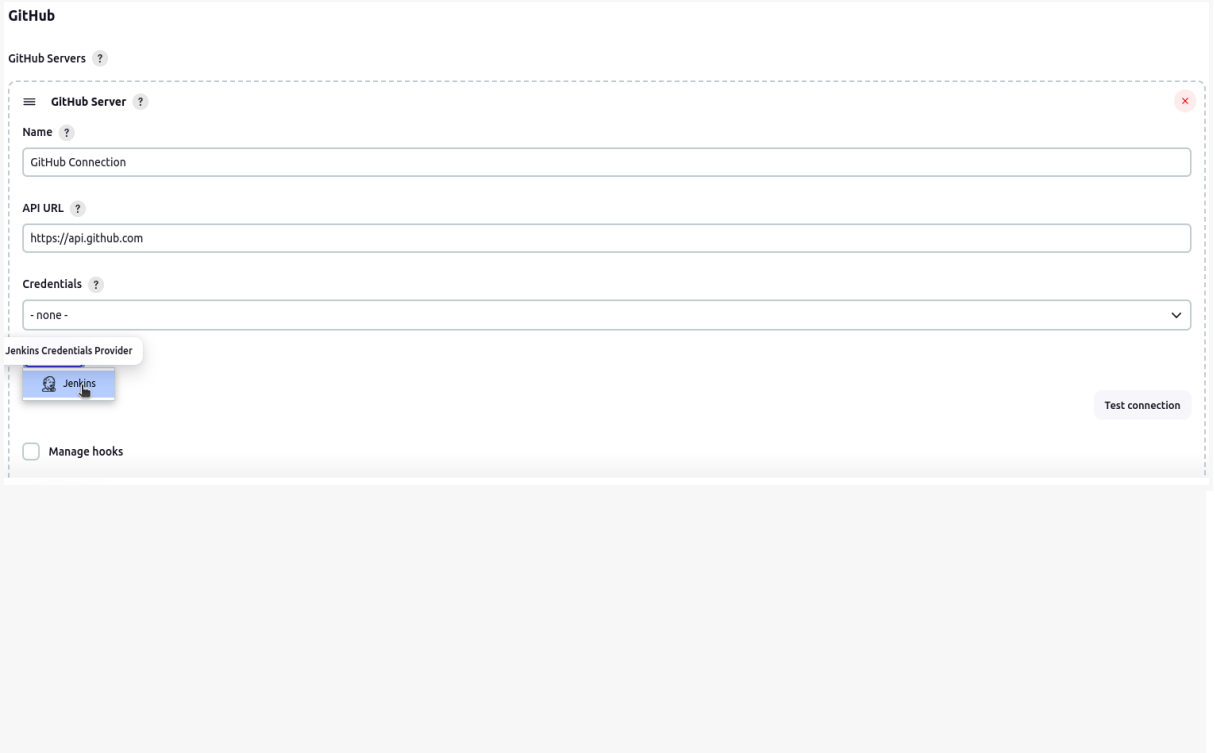
Add GitHub Configuration In Jenkins :

Open Jenkins in your browser, then navigate to "Manage Jenkins" -> " System."



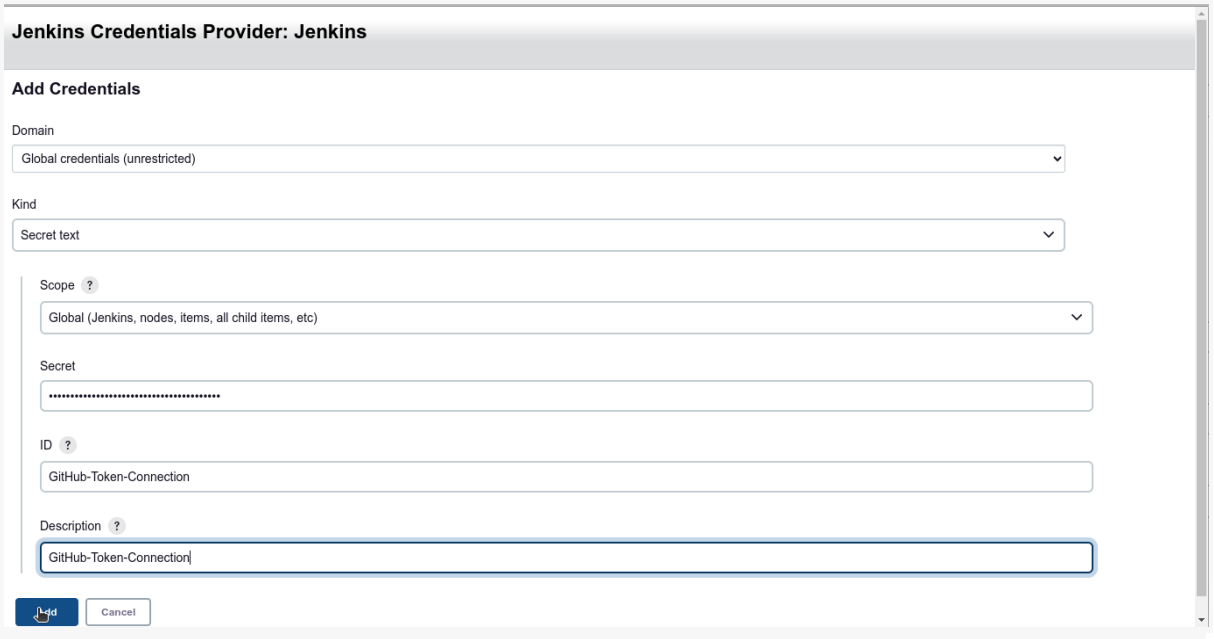
Scroll down until you find the GitHub section
Click on "Add GitHub Server" and fill in the details.

- Name: Can be any name to identify this GitHub configuration.
- API URL: Usually this is `https://api.github.com` for GitHub.com



For "Credentials," click the "Add" button next to the dropdown.

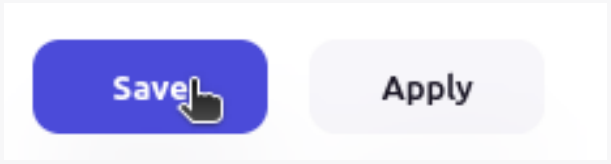
- Kind: Choose "Secret text."
- Secret: Paste your GitHub Personal Access Token here.
- ID: Can be any name to identify this secret.
- Description: Optional, but helps you remember what this secret is for.



choose the new created credentials and press “Test connection”

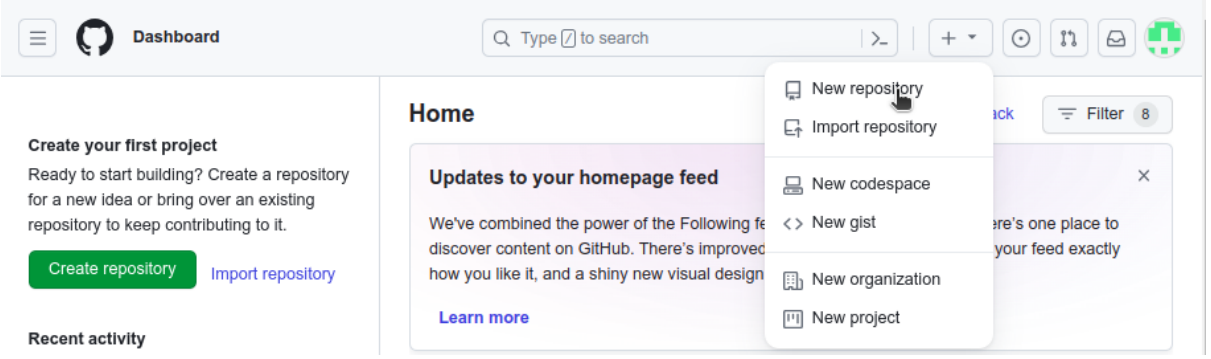
the output should be something like “Credentials verified for user orioElias, rate limit:4999”

press “**Save**”

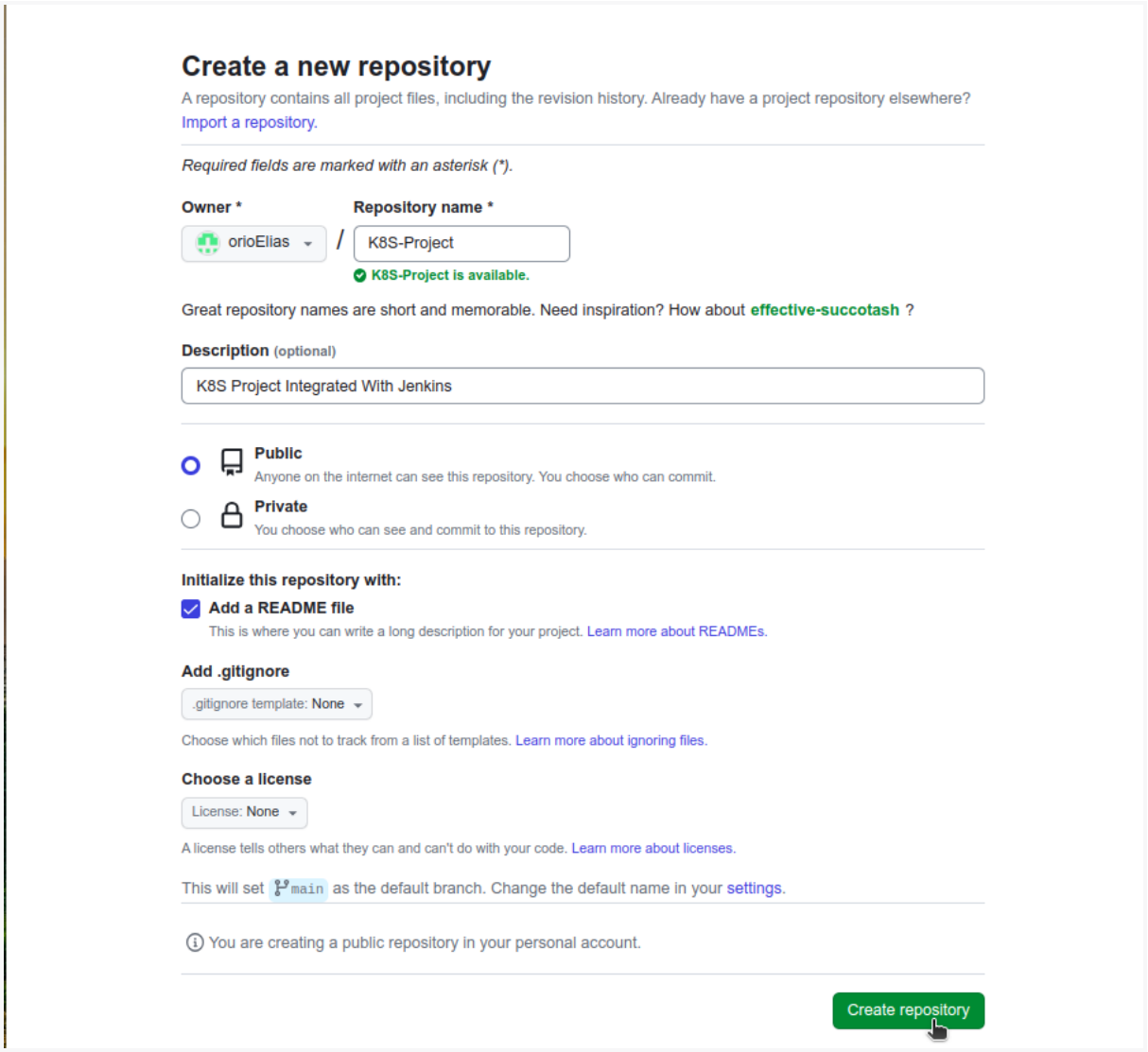


Create a GitHub Repository

log in GitHub, you'll see a "+" icon in the upper-right corner next to your profile picture. Click on it and select "New repository."

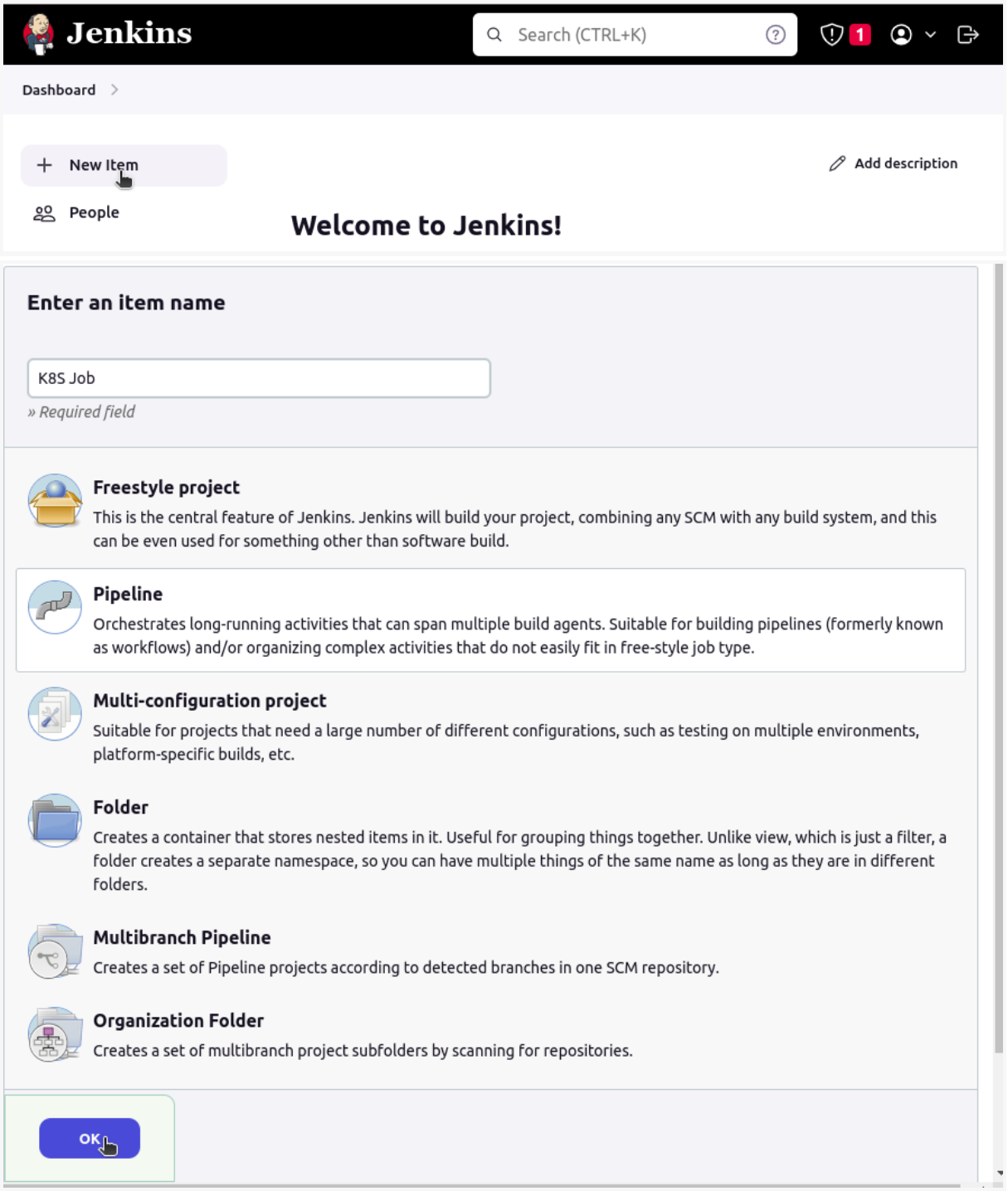


Fill your repository details, then Click “Create repository”



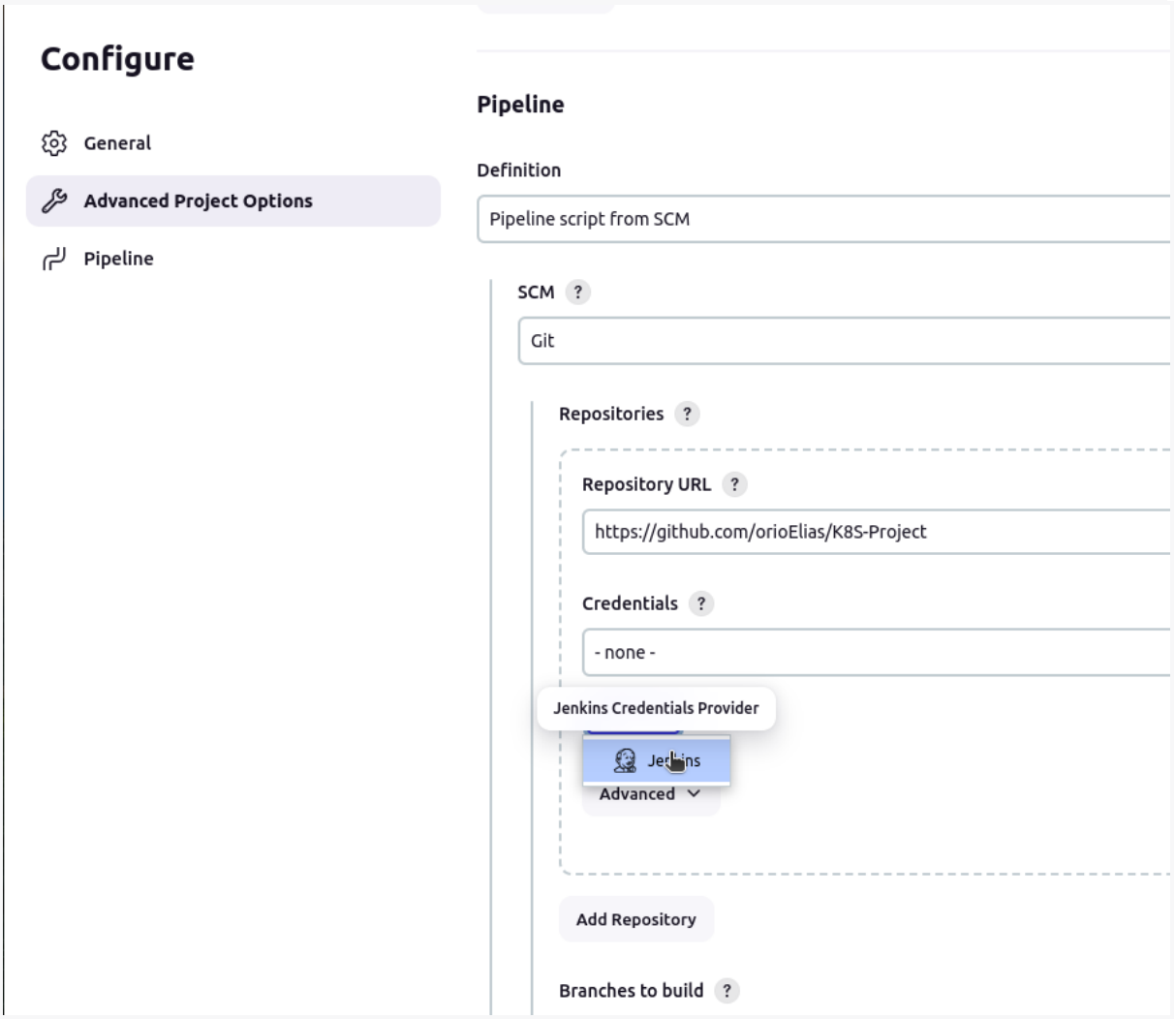
Connect the New GitHub Repository To Jenkins:

Open your Jenkins interface, Click on "New Item" on the Jenkins dashboard, select "Pipeline," and then proceed.



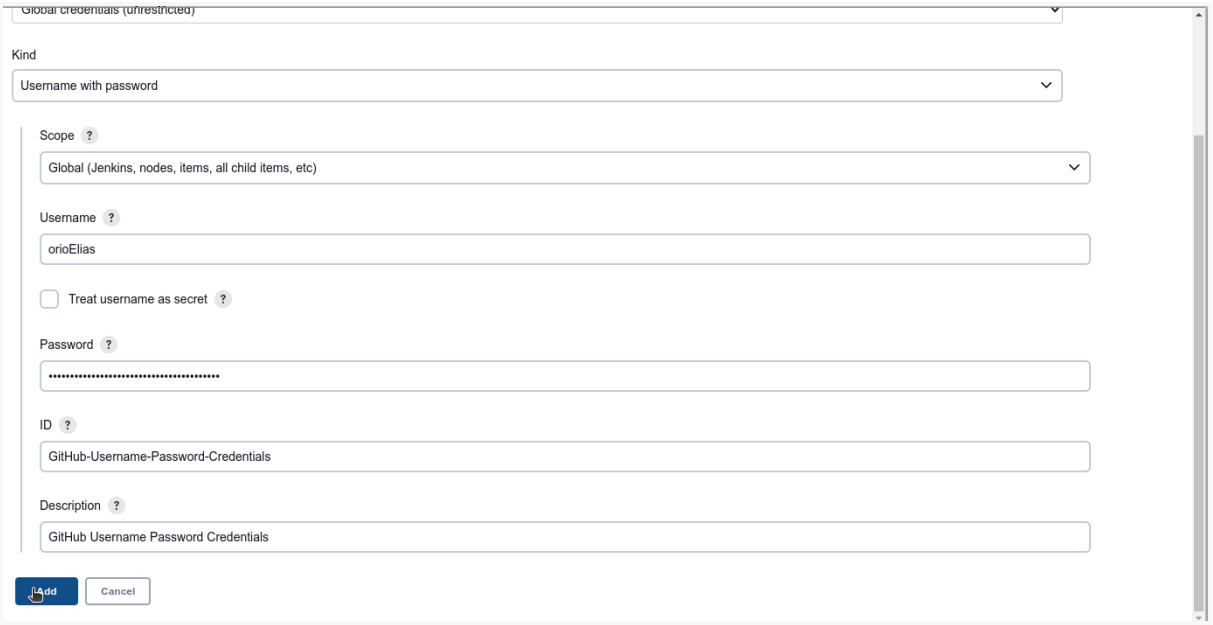
In the pipeline configuration, Check "GitHub hook trigger for GITScm polling". Additionally you'll need to specify the source as your GitHub repository.

- Source: Git
- Repository URL: The URL of the GitHub repository you just created
- Credentials: Create a new set of credentials in Jenkins using the same GitHub Personal Access Token (PAT) you generated earlier.

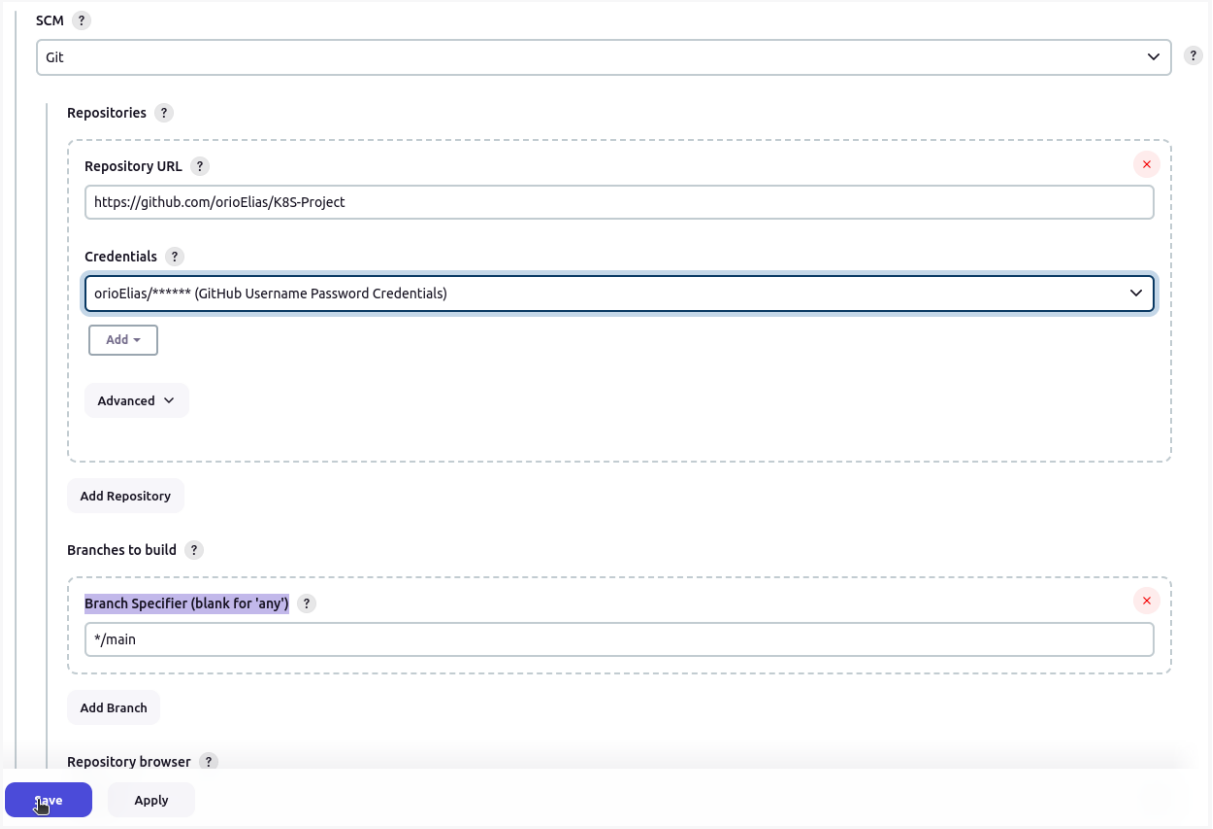


In the "Credentials" dropdown, click on the "Add" button and Jenkins.

- For the “Kind” field, choose “Username with password.”
  Enter your GitHub username in the “Username” field.
- Paste your GitHub Personal Access Token in the “Password” field.
- Add an ID and Description to identify these credentials.

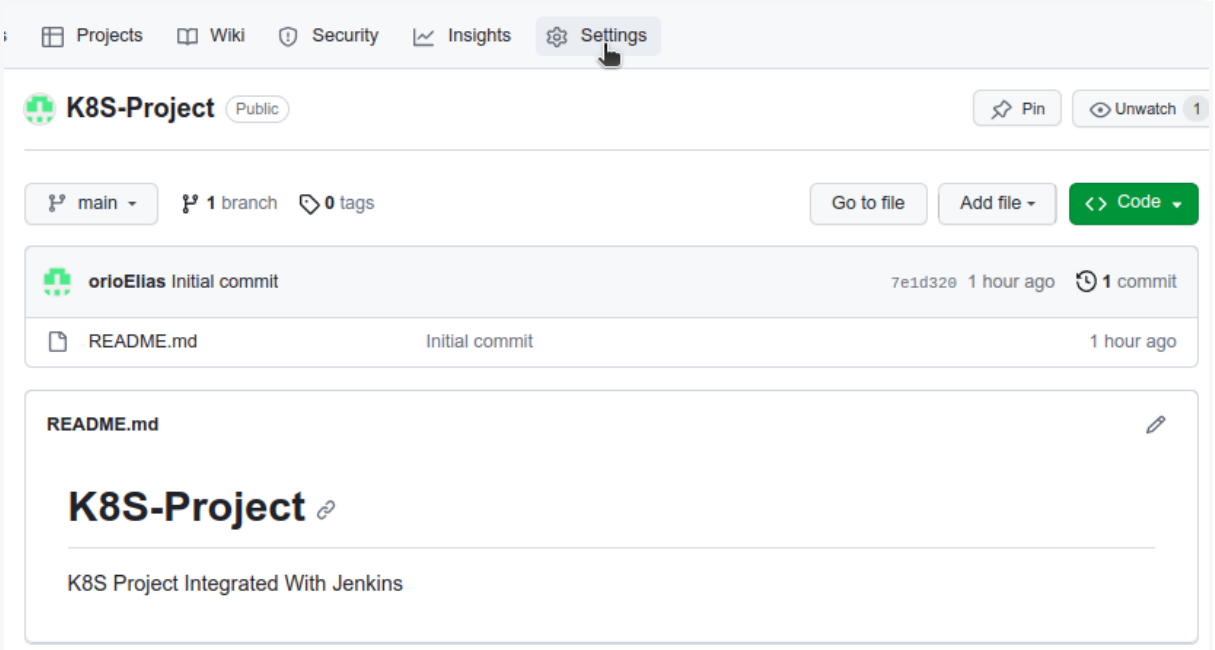


Now, choose the New Credentials, Change “Branch Specifier (blank for 'any')” From: “*/master” To “*/main” And Click “SAVE”

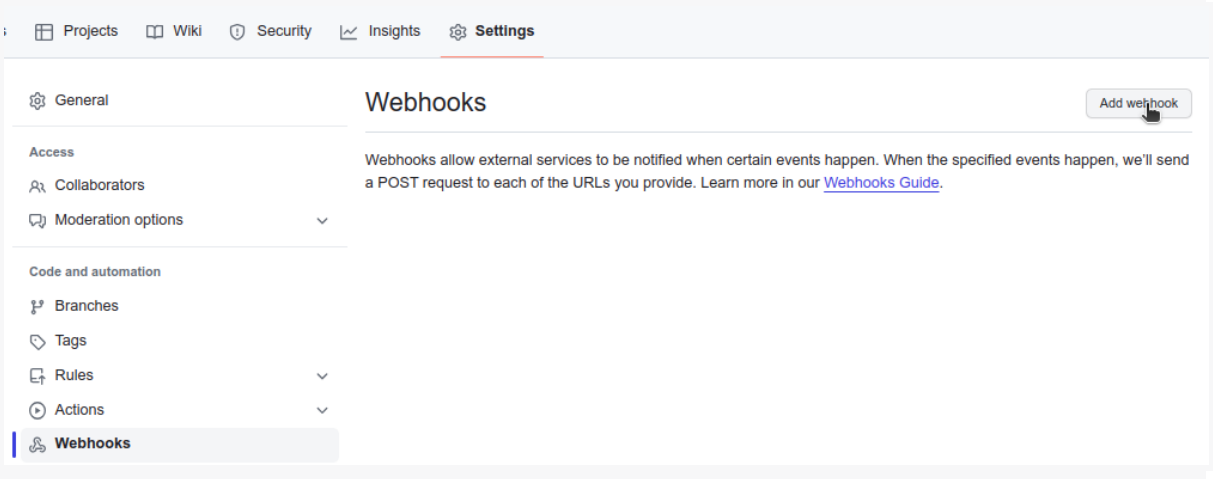


Add GitHub Webhook to Trigger Jenkins Pipeline

GitHub Repository Settings: Navigate to "Settings" on your GitHub repository.



Webhooks: Go to the "Webhooks" tab and click on "Add webhook."



Use your Jenkins URL followed by /github-webhook/. For example, http://’your-public-ip:30000/github-webhook/

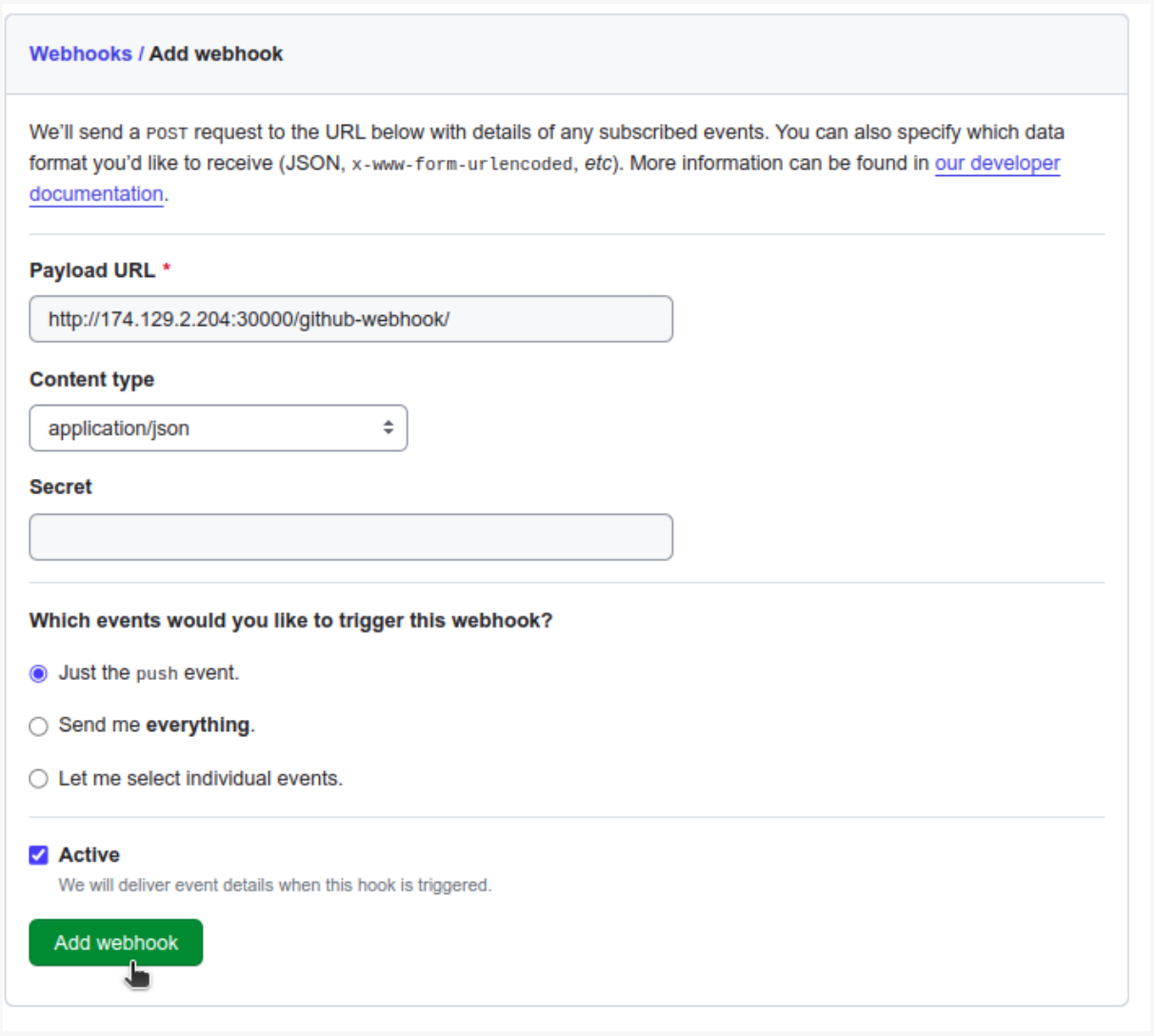
Content type: Choose application/json

Secret: Leave this blank if you haven’t set up a secret in Jenkins for webhook validation.

Choose "Just the push event" if you want the pipeline to run only when new commits are pushed.

Active: Make sure this checkbox is checked.

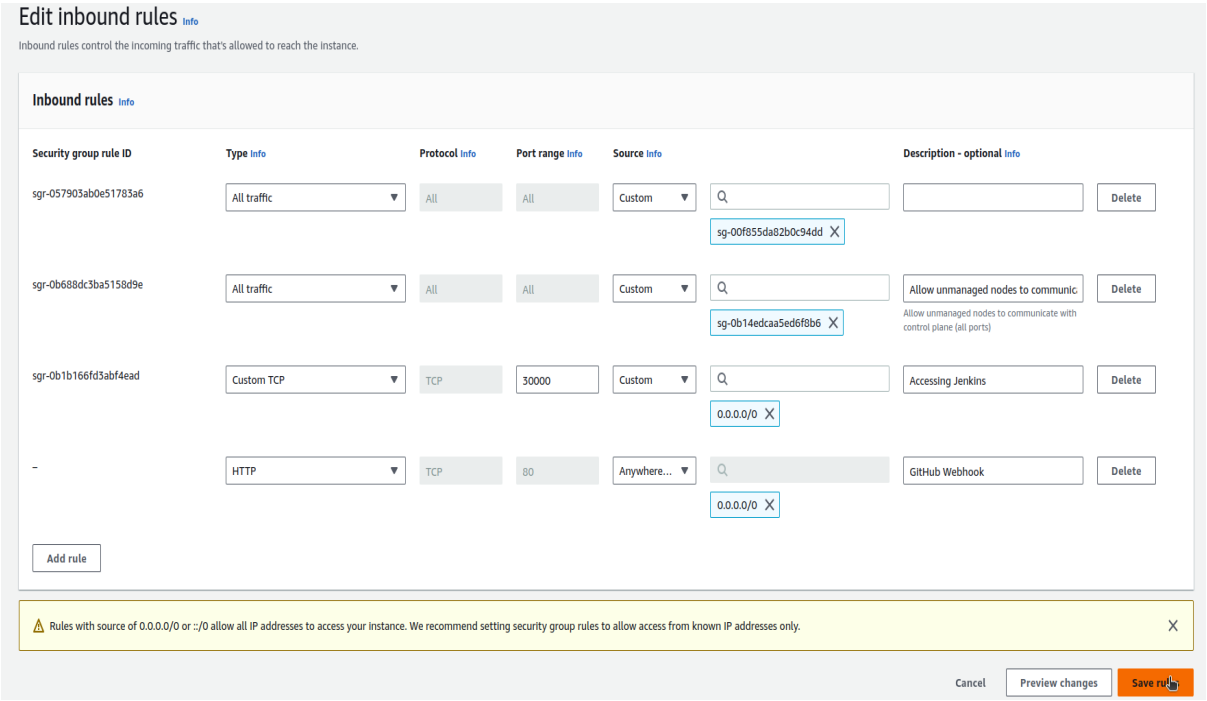
Click: “Add webhook”



Allow Traffic on Port 80 (For the Webhook, Same as we did Before)

Here's how to add a rule for port 80 in your AWS Security Group:

Navigate to: EC2 -> Instances -> Click on Instance ID (worker node) -> Security -> Security groups -> Inbound rules -> -> Edit inbound rules -> Add rule (HTTP, CIDR - 0.0.0.0/0) -> Save rules



Clone the repository to your local machine && Checking the Webhook

run in the terminal (in the directory that you want to clone your repo) :
git clone https://github.com/<your-username>/<directory-name>.git

this will create a new directory that contains your GitHub repository.

```
ori@OriElias:~/Desktop/k8s project$ git clone https://github.com/orioElias/K8S-Project.git
Cloning into 'K8S-Project'...
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.
```

cd <directory-name>
Make some changes to the existing files or add new files to the directory.
git add <filename>
git commit -m "Your commit message here"
git config --global credential.helper 'cache --timeout=3600' (optional)

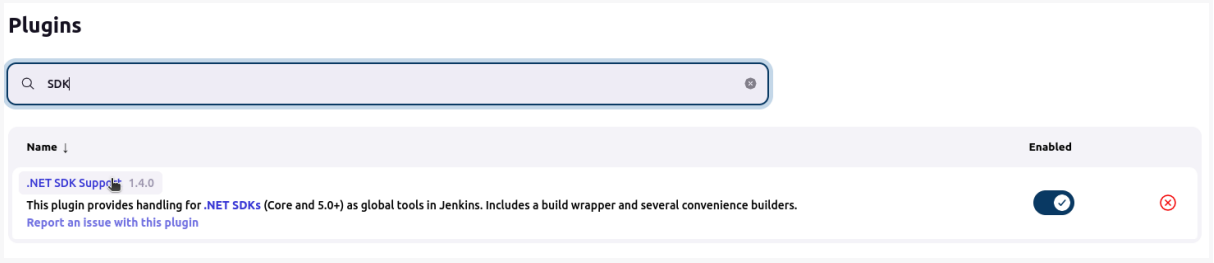
git push origin main
username: <your-username>
password: <Personal Access Token>

After you push the commit, your GitHub webhook should trigger your Jenkins pipeline if it's configured correctly.

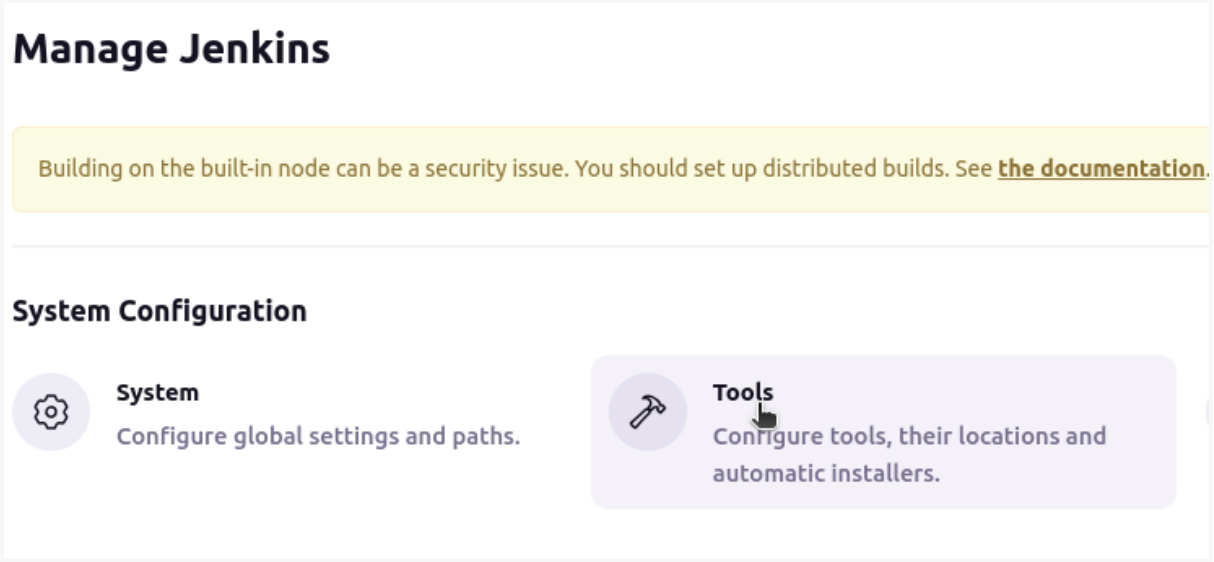
Download an ASP.NET Core web application.

Download from Official Samples: Microsoft offers various sample projects that you can download directly. You can find these on the official ASP.NET Core samples page.

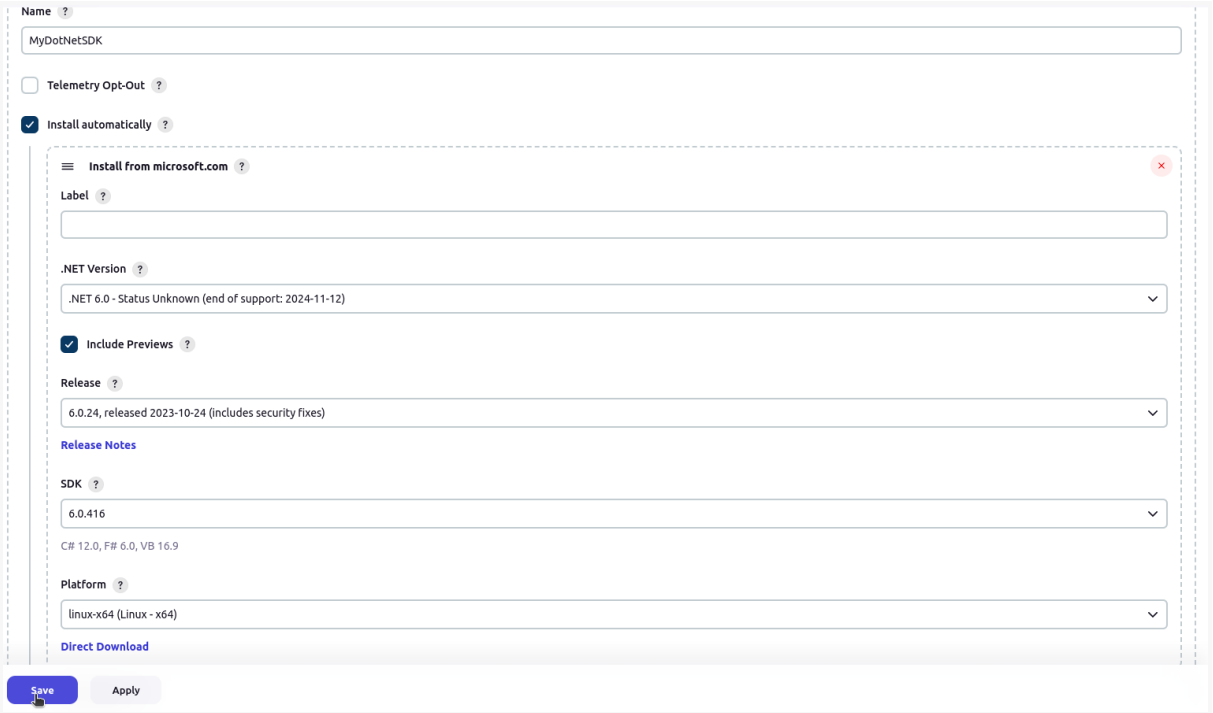
Install .NET SDK plugin



- **Click on "Manage Jenkins" on the left sidebar.**
- **Then click on "Tools"**



Configure as your needed



Create Role Files

deployment-manager-role.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: deploy
  name: deployment-manager
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "create", "update", "delete"]
```

deployment-manager-rolebinding.yaml

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: deployment-manager-binding
  namespace: deploy
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: deployment-manager
subjects:
- kind: ServiceAccount
  name: default
  namespace: devops
```

Create a Jenkinsfile

```jenkinsfile
pipeline {
    agent any

    tools {
        // Define the .NET SDK tool
        dotnetsdk 'MyDotNetSDK'
    }

    environment {
        // Add this line to disable globalization
        SYSTEM_GLOBALIZATION_INVARIANT = 'true'
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }

        stage('Docker Build') {
            steps {
                sh '''
                    docker build -t my-dotnet-app:latest ./MySimpleWebApp
                '''
            }
        }

        stage('Docker Push') {
            steps {
                sh '''
                    docker tag my-dotnet-app:latest orielias/my-dotnet-app:latest
```

```
        stage('Deploy to Deploy Namespace') {
            steps {
                sh '''
                    if ! kubectl get namespaces | grep -q 'deploy'; then
                        kubectl create namespace deploy
                    fi
                    kubectl apply -f deployment-deploy.yaml --namespace=deploy
                '''
            }
        }
    }
}
```

Create Dockerfile

Dockerfile ×

home > ori > Desktop > k8s project > K8S-Project > Dockerfile

```
# Use the SDK image to build the app
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS build
WORKDIR /source

# Restore and publish the app
COPY MySimpleWebApp/MySimpleWebApp.csproj .
RUN dotnet restore
COPY MySimpleWebApp/. .
RUN dotnet publish -c release -o /app

# Use the runtime image to run the app
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS runtime
WORKDIR /app
COPY --from=build /app .
ENTRYPOINT ["dotnet", "MySimpleWebApp.dll"]
```

Create deployment-deploy.yaml file

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dotnetcore-deploy
  namespace: deploy
spec:
  replicas: 2
  selector:
    matchLabels:
      app: dotnetcore-deploy
  template:
    metadata:
      labels:
        app: dotnetcore-deploy
    spec:
      containers:
      - name: dotnetcore-deploy
        image: orielias/my-dotnet-app:latest
        ports:
        - containerPort: 80
```

Create a service To Access this WEB APP

```yaml
apiVersion: v1
kind: Service
metadata:
  name: dotnetcore-service
  namespace: deploy
spec:
  type: NodePort
  selector:
    app: dotnetcore-deploy
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080
```

Your file structure suppose to be similar to this :

```
├── deployment-deploy.yaml
├── Dockerfile
├── dotnetcore-deploy.yaml
├── Jenkinsfile
├── MySimpleWebApp
│   ├── mySimpleWebApp.csproj
│   ├── program.cs
│   └── startup.cs
```

Now you can push the new Files to the repository and See the pipeline runs :

Declarative: Checkout SCM	Declarative: Tool Install	Checkout	Docker Build	Docker Push	Deploy to Devops Namespace	Deploy to Deploy Namespace
637ms	220ms	636ms	742ms	127ms	83ms	77ms