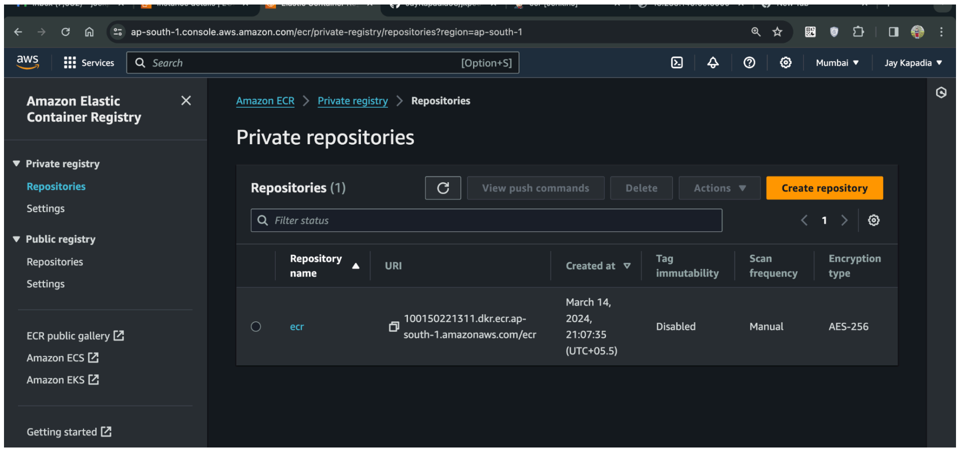
Install AWS CLI and Terraform.
Create Access key and secret key from AWS Console.
Pass AWS Access Key, Secret Key, and Region as **Environment Variables** to avoid printing them as **plain text** in terraform state files.

Go to Aws console and create a key pair.
Download the .pem file. (Without this you won't be able to ssh into ec2)

Write Terraform Config File to create EC2 and ECR.

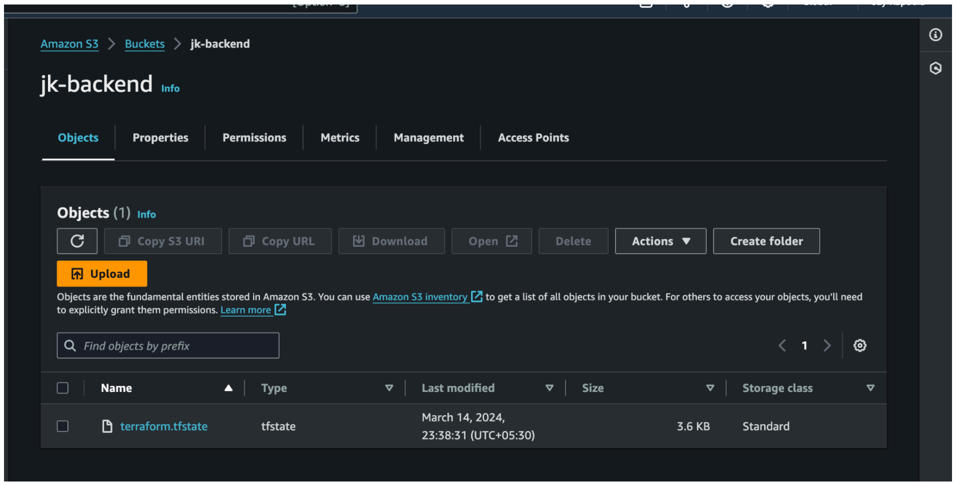


Private repositories

Repositories (1)

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
ecr	100150221311.dkr.ecr.ap-south-1.amazonaws.com/ecr	March 14, 2024, 21:07:35 (UTC+05.5)	Disabled	Manual	AES-256

Create an S3 bucket and configure it as a **remote backend** to avoid **.tfstate** files being corrupted or **compromised** while pushing it to any **public repo.**
Make sure **versioning** is **enabled** and the bucket is **private**.



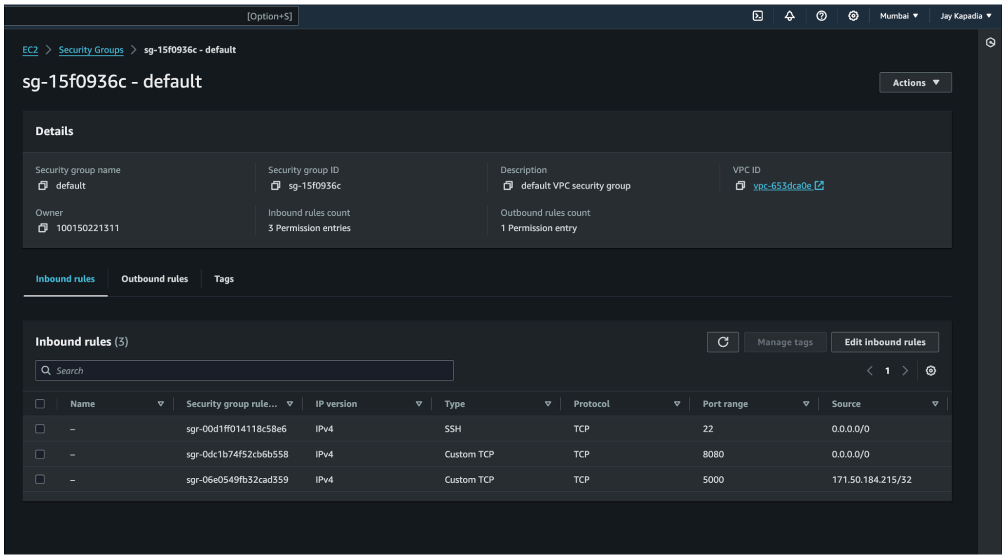
jk-backend

Objects (1)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
terraform.tfstate	tfstate	March 14, 2024, 23:38:31 (UTC+05:30)	3.6 KB	Standard

Edit Inbound rules for the Security group.
Allow Inbound traffic for port 8080(Jenkins), 22(SSH) and 5000(Flask App)
Allow all outbound traffic to install Jenkins, Docker, and dependencies.



sg-15f0936c - default

Details

Security group name default	Security group ID sg-15f0936c	Description default VPC security group	VPC ID vpc-653dca0e
Owner 100150221311	Inbound rules count 3 Permission entries	Outbound rules count 1 Permission entry	

Inbound rules (3)

Name	Security group rule...	IP version	Type	Protocol	Port range	Source
–	sgr-00d1ff014118c58e6	IPv4	SSH	TCP	22	0.0.0.0/0
–	sgr-0dc1b74f52cb6b558	IPv4	Custom TCP	TCP	8080	0.0.0.0/0
–	sgr-06e0549fb32cad359	IPv4	Custom TCP	TCP	5000	171.50.184.215/32

Login to the EC2 instance with ssh -i ‘path to .pem file’ ubuntu@publicIP.
(We can use Terraform **remote-exec provisioner** to install packages once EC2 is created)

To keep things simple we will do it manually

Install JDK (Jenkins Depedency)

```
sudo apt update
sudo apt install openjdk-11-jre
```

Install Jenkins

```
$curl -fsSL https://pkg.jenkins.io/debian/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 binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
$sudo apt-get update
$sudo apt-get install jenkins
```

Add jenkins user to docker user group

```
sudo su -
usermod -aG docker jenkins
```

Write a Hello World application using Flask and Python.
Create a Docker file, pull the base image for Python, and expose port 5000.
Writeue requirements.txt for dependencies to be installed on the ec2 instance.

```
# Use the official Python image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container at /app
COPY requirements.txt .

# Install any needed dependencies specified in requirements.txt
RUN pip install  -r requirements.txt

# Copy the current directory contents into the container at /app
COPY . .

# Expose port 5000 to the outside world
EXPOSE 5000

# Run app.py when the container launches
CMD ["python", "app.py"]
```

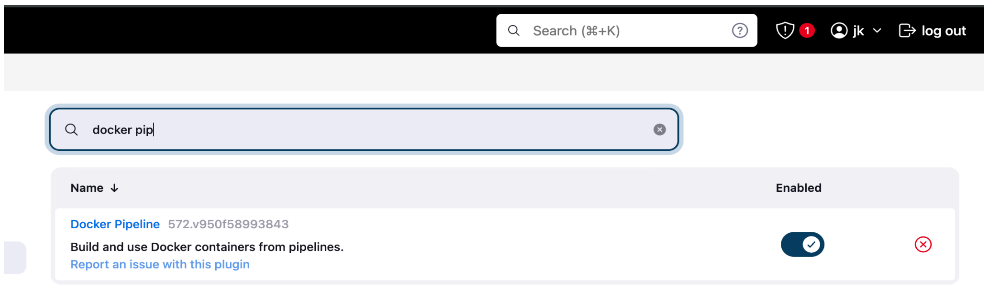
Build Image from the docker file we created
docker build -t tag .

Run a container from the image we built and port forward 5000
docker run -p 5000:5000 tag

Hello world will be visible curl to localhost:5000 or 127.0.0.0:5000

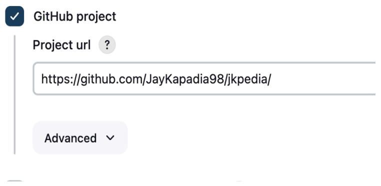
Go to browser and hit public ip of ec2 at port 8080
http://ip:8080

Configure Jenkins and login
Install Docker Pipeline plugin and Github plugin



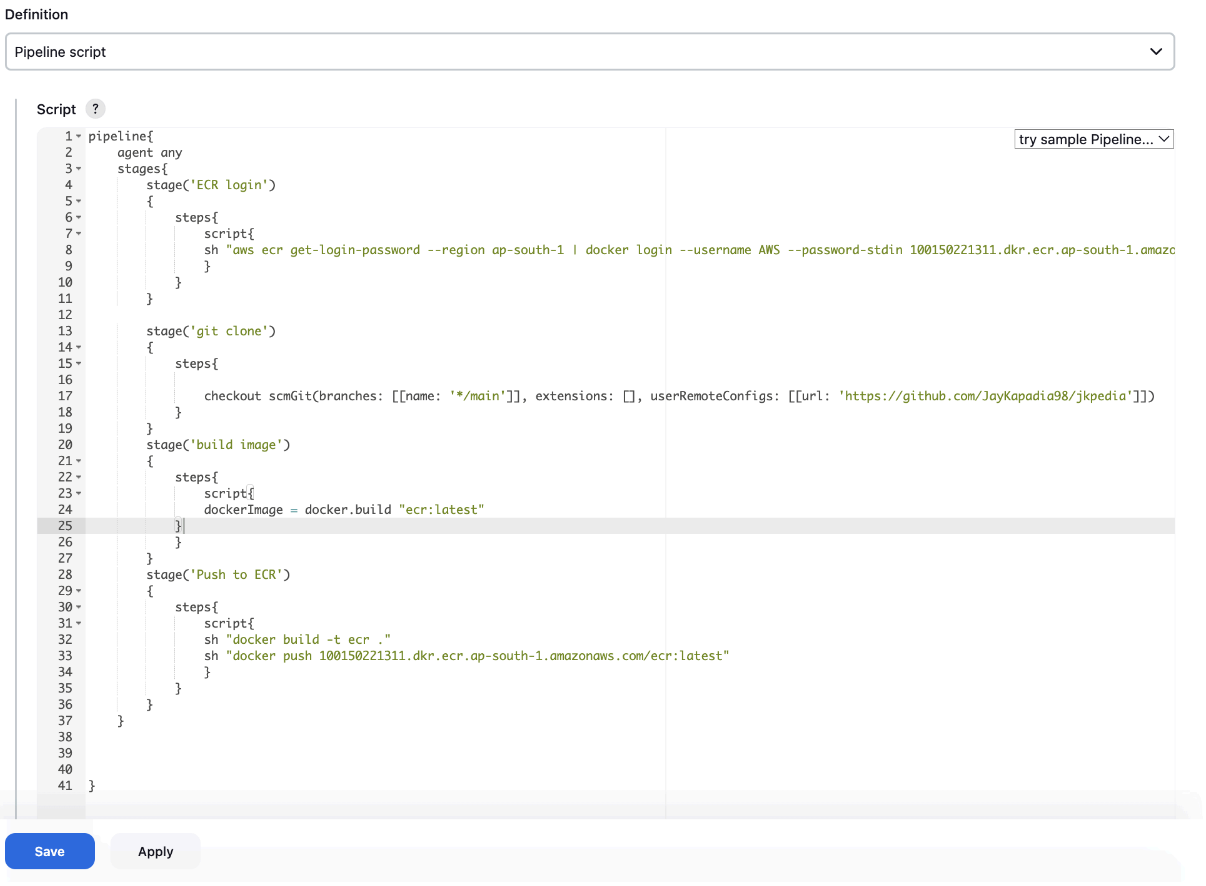
Docker Pipeline 572.v950f58993843
Build and use Docker containers from pipelines.
Report an issue with this plugin

Create a pipeline and check GitHub project and paste URL of repository



☑ GitHub project
Project url ?
https://github.com/JayKapadia98/jkpedia/

Tick ‘GitHub hook trigger for GITScm polling’ box for auto build via GitHub commits
Create pipeline and add stages.

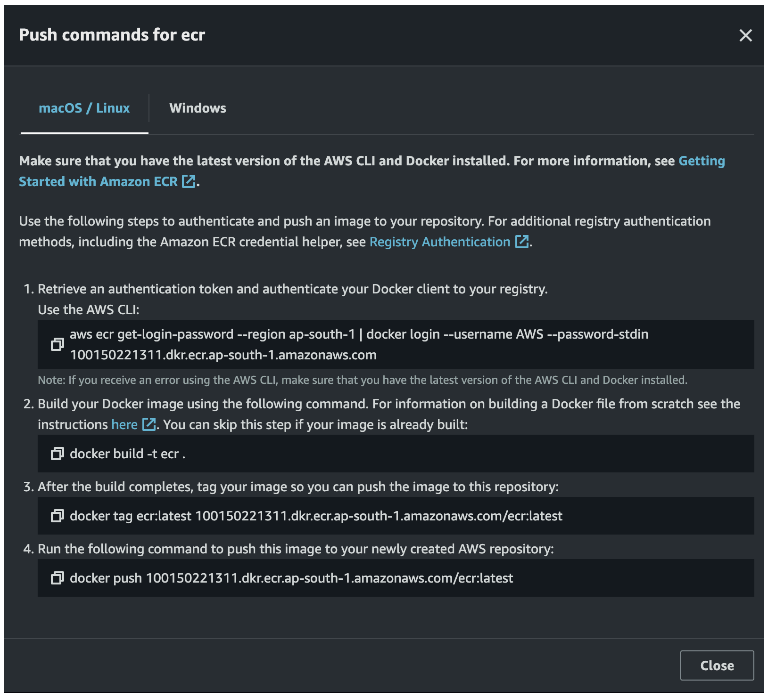


Definition
Pipeline script

Script ?

```
pipeline{
    agent any
    stages{
        stage('ECR login')
        {
            steps{
                script{
                sh "aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 100150221311.dkr.ecr.ap-south-1.amazc
                }
            }
        }

        stage('git clone')
        {
            steps{

                checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/JayKapadia98/jkpedia']])
            }
        }
        stage('build image')
        {
            steps{
                script{
                dockerImage = docker.build "ecr:latest"
                }
            }
        }
        stage('Push to ECR')
        {
            steps{
                script{
                sh "docker build -t ecr ."
                sh "docker push 100150221311.dkr.ecr.ap-south-1.amazonaws.com/ecr:latest"
                }
            }
        }
    }



}
```

Open ECR, check for push commands and you will find these.



Push commands for ecr

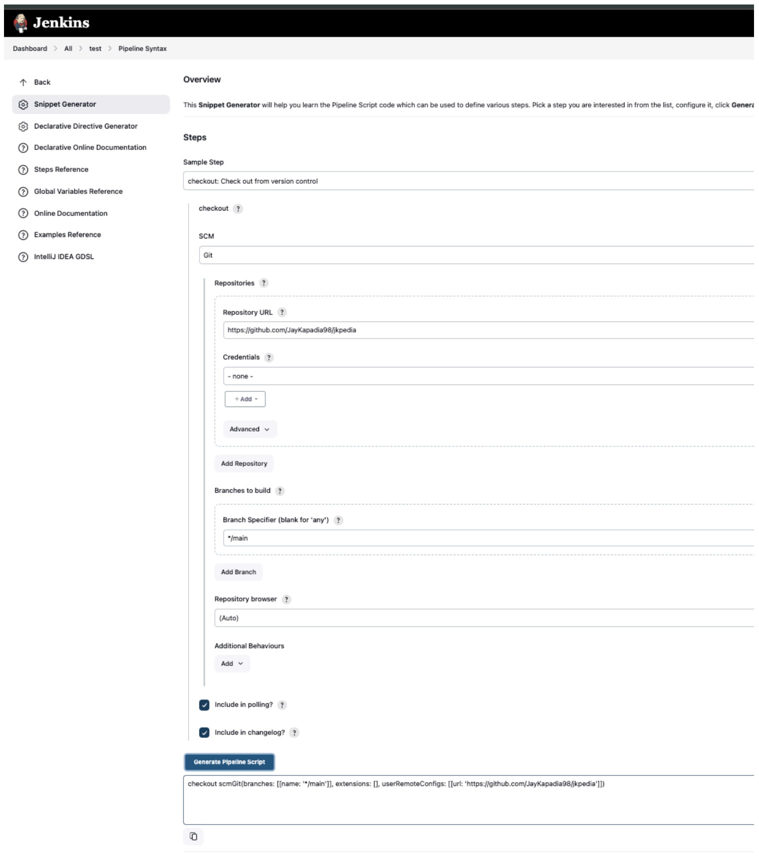
Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see Getting Started with Amazon ECR.

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see Registry Authentication.

1. Retrieve an authentication token and authenticate your Docker client to your registry.
   Use the AWS CLI:
   ```
   aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin
   100150221311.dkr.ecr.ap-south-1.amazonaws.com
   ```
   Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.
2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here. You can skip this step if your image is already built:
   ```
   docker build -t ecr .
   ```
3. After the build completes, tag your image so you can push the image to this repository:
   ```
   docker tag ecr:latest 100150221311.dkr.ecr.ap-south-1.amazonaws.com/ecr:latest
   ```
4. Run the following command to push this image to your newly created AWS repository:
   ```
   docker push 100150221311.dkr.ecr.ap-south-1.amazonaws.com/ecr:latest
   ```

Create IAM role for EC2 instance to push image to ECR.

Click on pipeline syntax, and click checkout from version control.
Add the repo link, branch name and click Generate pipeline script.


Jenkins
Dashboard > All > test > Pipeline Syntax
Back
Snippet Generator
Declarative Directive Generator
Declarative Online Documentation
Steps Reference
Global Variables Reference
Online Documentation
Examples Reference
IntelliJ IDEA GDSL

Overview

This **Snippet Generator** 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 **Genera**

Steps

Sample Step: checkout: Check out from version control

checkout

SCM: Git

Repositories

Repository URL: https://github.com/JayKapadia98/jkpedia

Credentials: - none -

+ Add
Advanced
Add Repository

Branches to build

Branch Specifier (blank for 'any'): */main

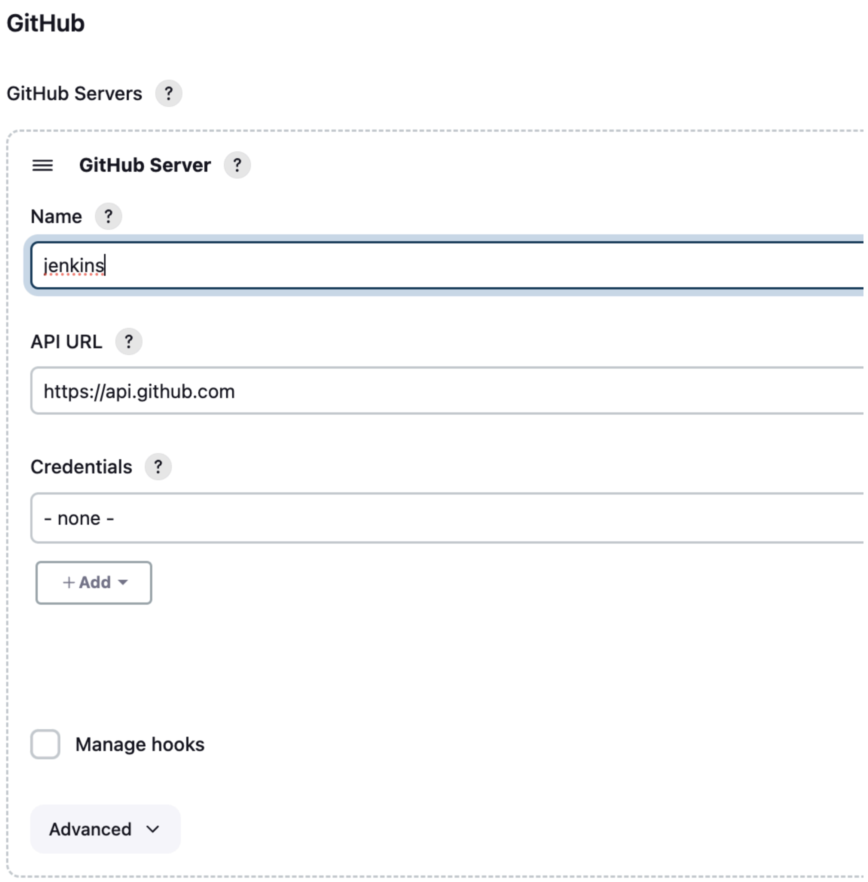
Repository browser: (Auto)

Additional Behaviours

[✓] Include in polling?

[✓] Include in changelog?

```
checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/JayKapadia98/jkpedia']])
```



GitHub

GitHub Servers

GitHub Server

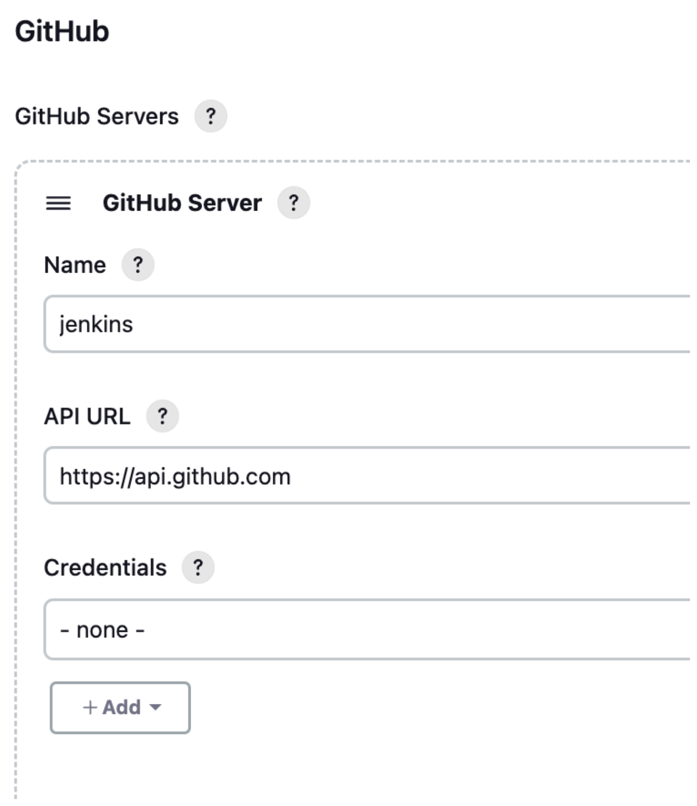
Name: jenkins

API URL: https://api.github.com

Credentials: - none -

Go to Dashboard >Manage Jenkins > System

Add Git Hub server and Keep API URL as it is.



GitHub

GitHub Servers

GitHub Server

Name

jenkins

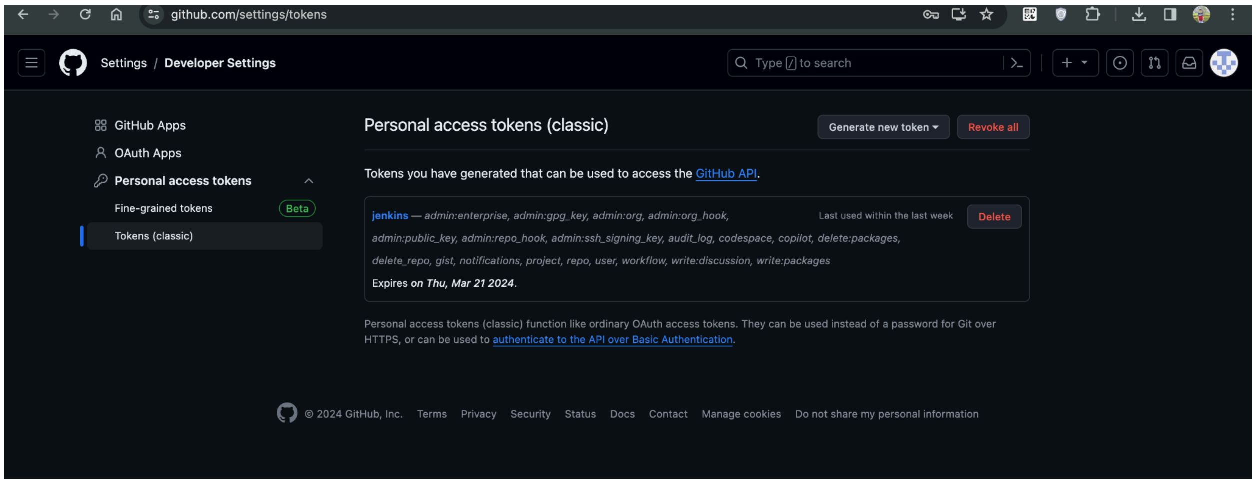
API URL

https://api.github.com

Credentials

- none -

Before adding credentials, go to developer settings in GitHub and create a personal token.



Personal access tokens (classic)

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

jenkins — *admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, copilot, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:packages*

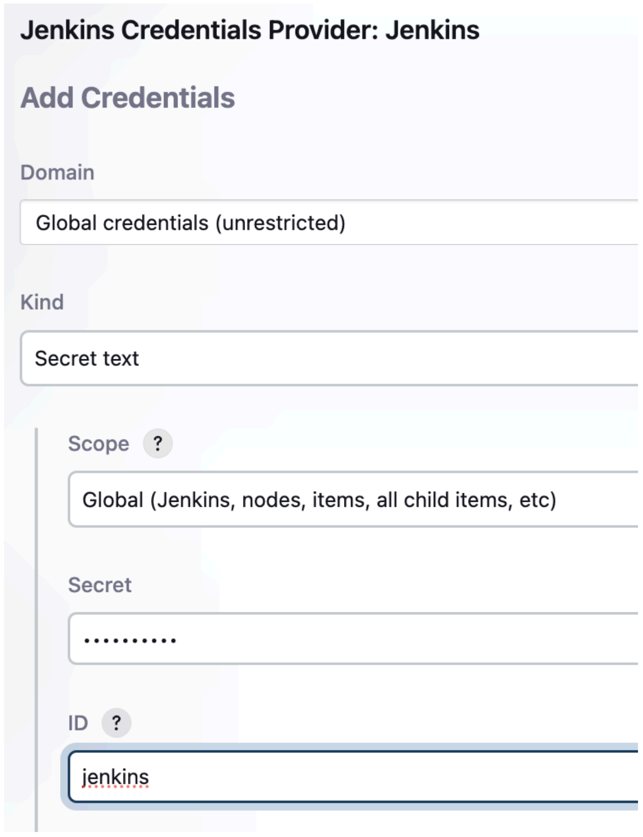
Last used within the last week

Expires *on Thu, Mar 21 2024*.

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.

© 2024 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

Copy token, come to jenkins, select **secret text** as kind, global as scope and paste the token



Jenkins Credentials Provider: Jenkins

Add Credentials

Domain: Global credentials (unrestricted)

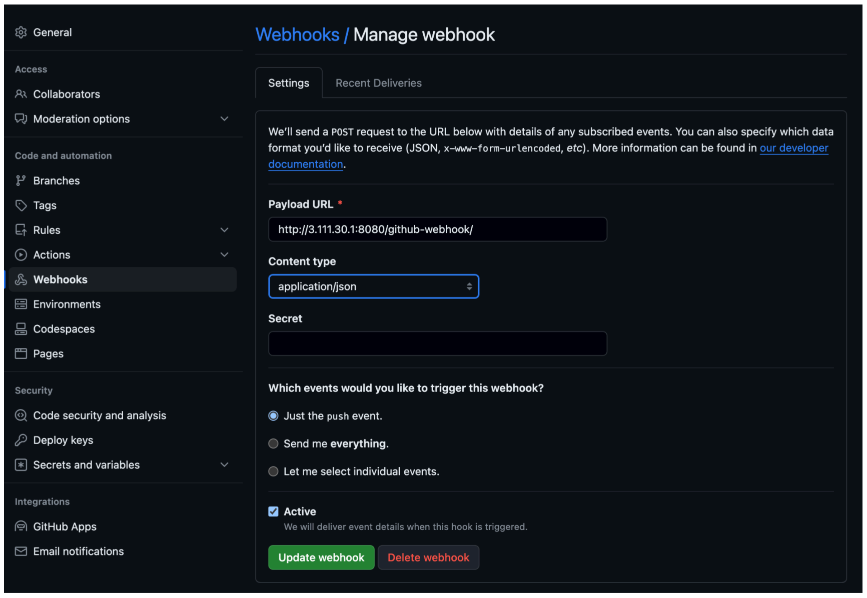
Kind: Secret text

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

Secret: ••••••••••

ID: jenkins

Go to setting of your repo, click webhook
Enter Payload url as jenkins url along with /github-webhook/
Example: http://3.111.30.1:8080/github-webhook/
Select Content type as application/json



Webhooks / Manage webhook

We'll send a `POST` request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, `x-www-form-urlencoded`, *etc*). More information can be found in our developer documentation.

Payload URL *: http://3.111.30.1:8080/github-webhook/

Content type: application/json

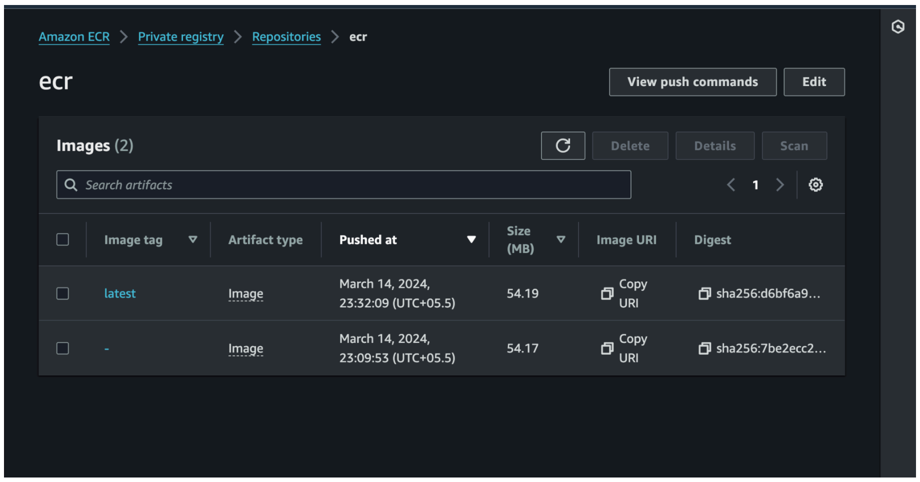
Secret:

Which events would you like to trigger this webhook?

- [✓ Just the `push` event.]
- Send me **everything**.
- Let me select individual events.

[✓ Active]
We will deliver event details when this hook is triggered.

Push code to repo and Build will trigger automatically.
New Image will be pushed to ECR.


Amazon ECR > Private registry > Repositories > ecr

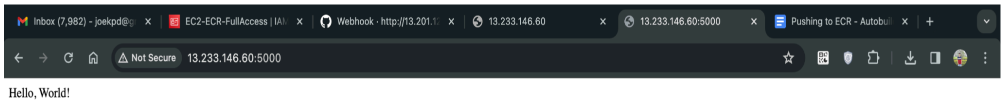
ecr

Images (2)

	Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
	latest	Image	March 14, 2024, 23:32:09 (UTC+05.5)	54.19	Copy URI	sha256:d6bf6a9...
	-	Image	March 14, 2024, 23:09:53 (UTC+05.5)	54.17	Copy URI	sha256:7be2ecc2...

Paste the public IP of EC2 with port 5000 and we can see our Hello World Application running on the internet.

Example: http://13.233.146.60:5000



Hello, World!