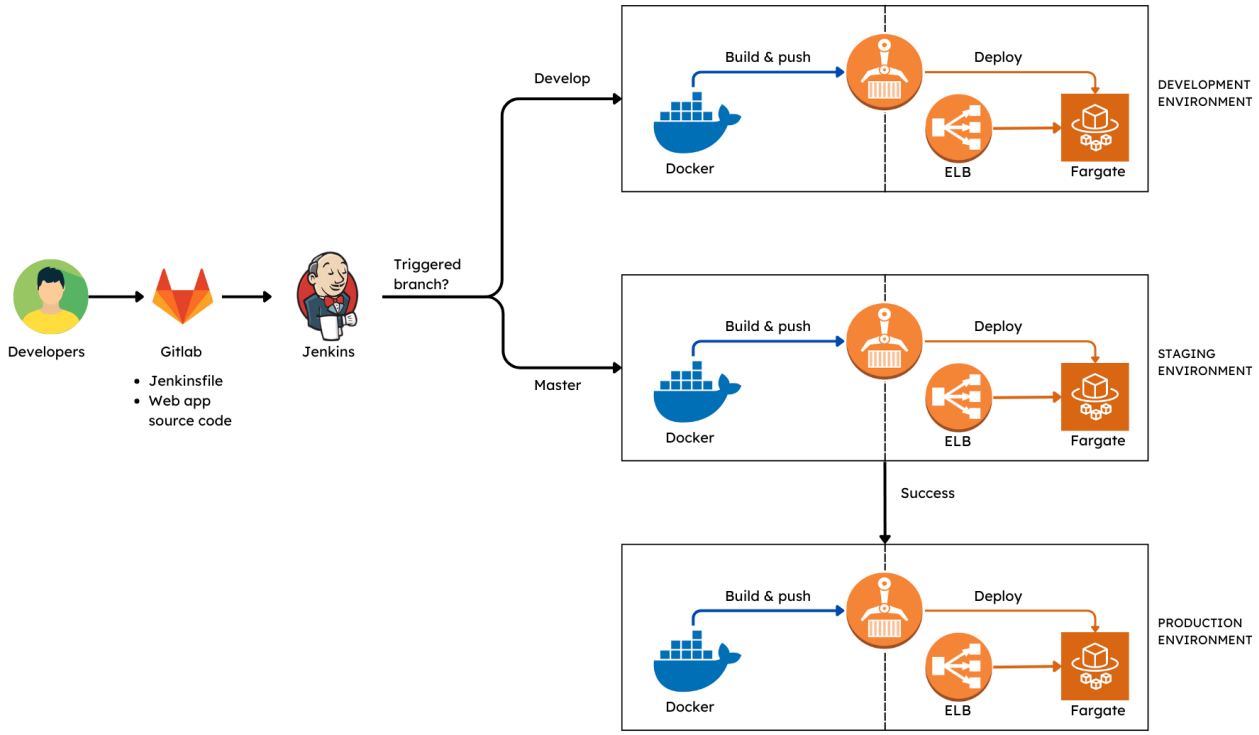
AWS CI/CD for deploying a FastAPI application

Goal

This is a guideline on how to setup a CI-CD pipeline using Jenkins to deploy a FastAPI application on develop/staging/production environments on AWS. The source code is stored at jenkins-aws-for-web-app



Set up the CI/CD pipeline

1. Gitlab:

- The repo has 2 branches: `dev` and `master`
- Directory structure:

```
.
├── app
│   └── main.py
├── jenkins_setup
│   ├── docker-compose.yml
│   └── Dockerfile
├── Jenkinsfile
├── Dockerfile
├── docker-compose.yml
├── load-balancer.yaml
├── .flake8
├── .gitignore
├── images
├── README.md
└── requirements.txt
```

- The repo includes:
  - FastAPI application source code.

```python
# In ./app/main.py
from fastapi import FastAPI

app = FastAPI()


@app.get('/')
async def root():
    return {'greeting': 'Hello from root function'}


@app.get('/{name}')
async def hello(name: str):
    return {'greeting': f'Hello {name}!'}
```

  - Jenskinfile: located in the root directory of the git repository `./Jenkinsfile`
  - Dockerfile and requirements.txt for the FastAPI application.

```dockerfile
FROM python:3.9-slim

WORKDIR /code

RUN apt-get update \
&& apt-get install -y --no-install-recommends \
        curl\
&& apt-get autoremove -yqq --purge \
&& apt-get clean \
&& rm -rf /var/lib/apt/lists/*

COPY ./requirements.txt /code/requirements.txt

RUN pip install --no-cache-dir --upgrade -r
/code/requirements.txt

COPY ./app /code/app

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port",
"80"]
```

  - jenkins_setup: Dockerfile and docker-compose.yml for running jenkins server using containers:

- Dockerfile:

```
FROM jenkins/jenkins:2.401.1
USER root
RUN apt-get update && apt-get install -y lsb-release
RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc \
https://download.docker.com/linux/debian/gpg
RUN echo "deb [arch=$(dpkg --print-architecture) \
signed-by=/usr/share/keyrings/docker-archive-keyring.asc] \
https://download.docker.com/linux/debian \
$(lsb_release -cs) stable" > /etc/apt/sources.list.d/docker.list
RUN apt-get update && apt-get install -y docker-ce-cli
RUN apt-get install -y --no-install-recommends \
        python3-pip python3-venv\
&& apt-get autoremove -yqq --purge \
&& apt-get clean \
&& rm -rf /var/lib/apt/lists/*

RUN pip install virtualenv
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-
x86_64.zip" -o "awscliv2.zip" && unzip awscliv2.zip &&
./aws/install
USER jenkins
RUN jenkins-plugin-cli --plugins "blueocean docker-workflow"
```

- docker-compose.yml:

```
networks:
jenkins:
    driver: bridge
services:
jenkins-docker:
    image: docker:dind
    container_name: jenkins-docker
    privileged: true
    restart: always
    networks:
    jenkins:
        aliases:
        - docker
    environment:
    - DOCKER_TLS_CERTDIR=/certs
    volumes:
    - ./jenkins-docker-certs:/certs/client
    - ./jenkins-data:/var/jenkins_home
    ports:
    - 2376:2376
    command: --storage-driver=overlay2

jenkins-blueocean:
```

```
    image: jenkins-blueocean:2.401.1-1
    build: .
    container_name: jenkins-blueocean
    networks:
    - jenkins
    environment:
    - DOCKER_HOST=tcp://docker:2376
    - DOCKER_CERT_PATH=/certs/client
    - DOCKER_TLS_VERIFY=1
    volumes:
    - ./jenkins-docker-certs:/certs/client:ro
    - ./jenkins-data:/var/jenkins_home
    ports:
    - 8080:8080
    - 50000:50000
    restart: always
```

2. Setup AWS services:

2.1 AWS ECR: Create 3 ECR repositories, one for each environment dev/staging/prod

2.2 AWS ECS:

- Create a task definition for each environment: Below is the example for creating a task definition for dev environment. Create task definitions for other environments similarly and names, image uri should be changed. Configuration not mentioned ought to be left default.

Task definition family Info
Specify a unique task definition family name.
dev-web-app-definition

Revision
Source revision
1

Container - 1 Info

Essential container | Remove

Container details
Specify a name, container image, and whether the container should be marked as essential. Each task definition must have at least one essential container.

Name	Image URI	Essential container
dev-web-app	666243375423.dkr.ecr.us-east-2.amazonaws.com/dev-\	Yes

Private registry Info
Store credentials in Secrets Manager, and then use the credentials to reference images in private registries.
Private registry authentication

Port mappings Info
Add port mappings to allow the container to access ports on the host to send or receive traffic. Any changes to port mappings configuration impacts the associated service connect settings.

Container port	Protocol	Port name	App protocol	
80	TCP	dev-web-app-80-tcp	HTTP	Remove
8080	TCP	dev-web-app-8080-tcp	HTTP	Remove

Add more port mappings

Environment variables - *optional* Info

Add individually
Add a key-value pair to specify an environment variable.

Environment
Specify the infrastructure requirements for the task definition.

App environment Info
Specify the infrastructure for the task definition.
Add an option
AWS Fargate (serverless)

Operating system/Architecture Info
Linux/X86_64

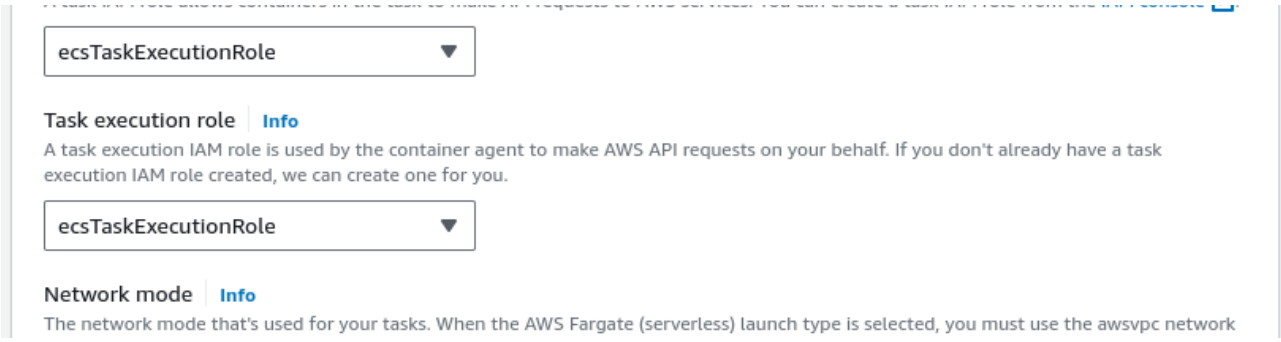
Task size Info
Specify the amount of CPU and memory to reserve for your task.

CPU	Memory
1 vCPU	3 GB

Container size - *optional* Info

Task roles, network mode - *conditional*

Task role Info
A task IAM role allows containers in the task to make API requests to AWS services. You can create a task IAM role from the IAM console



- Create a cluster for each environment: Below is the example for creating a cluster for dev environment. Create clusters for other environments similarly with other names.

Cluster configuration

Cluster name

DevCluster

There can be a maximum of 255 characters. The valid characters are letters (uppercase and lowercase), numbers, hyphens, and underscores.

Networking Info

By default tasks and services run in the default subnets for your default VPC. To use the non-default VPC, specify the VPC and subnets.

VPC

Use a VPC with public and private subnets. By default, VPCs are created for your AWS account. To create a new VPC, go to the VPC Console.

vpc-046ad6577b5ca5cfe
default | default

Subnets

Select the subnets where your tasks run. We recommend that you use three subnets for production.

Choose subnets

subnet-0344d44130e11b79c
us-east-2a 172.31.0.0/20

subnet-0de56d1e9c2d97060
us-east-2c 172.31.32.0/20

subnet-02bbfe46ba29ce20a
us-east-2b 172.31.16.0/20

Default namespace - *optional*

Select the namespace to specify a group of services that make up your application. You can overwrite this value at the service level.

DevCluster

Infrastructure Info

Serverless

Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers.
Add Amazon EC2 instances, or external instances using ECS Anywhere.

- [x] AWS Fargate (serverless)
  Pay as you go. Use if you have tiny, batch, or burst workloads or for zero maintenance overhead.
  The cluster has Fargate and Fargate Spot capacity providers by default.
- [ ] Amazon EC2 instances
  Manual configurations. Use for large workloads with consistent resource demands.
- [ ] External instances using ECS Anywhere
  Manual configurations. Use to add data center compute.

Monitoring - *optional* Info

Container Insights is off by default. When you use Container Insights, there is a cost associated with it.

Tags - *optional* Info

Tags help you to identify and organize your clusters.

Cancel Create

- Modify Cloudformation template for elastic load balancer: `TargetGroup/VpcId`, `LoadBalancer/Subnets`, `LoadBalancer/SecurityGroups` should be changed to meet your requirements.

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Create a public load balancer, listener and target group.

Resources:
TargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
    HealthCheckIntervalSeconds: 10
    HealthCheckPath: /
    HealthCheckProtocol: HTTP
    HealthCheckTimeoutSeconds: 5
    HealthyThresholdCount: 2
    TargetType: ip
    Name: !Join [ "-", [!Ref AWS::StackName, 'tg'] ]
    Port: 80
    Protocol: HTTP
    UnhealthyThresholdCount: 5
    VpcId: vpc-046ad6577b5ca5cfe

Listener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    DependsOn:
    - TargetGroup
    - LoadBalancer
    Properties:
    DefaultActions:
        - Type: forward
        TargetGroupArn: !Ref TargetGroup
    LoadBalancerArn: !Ref LoadBalancer
    Port: 80
    Protocol: HTTP

LoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
    IpAddressType: ipv4
    Name: !Join [ "-", [!Ref AWS::StackName, 'bl'] ]
    Scheme: internet-facing
    SecurityGroups:
        - sg-04741693a6494256c
    Subnets:
        - subnet-0344d44130e11b79c
        - subnet-0de56d1e9c2d97060
        - subnet-02bbfe46ba29ce20a

Outputs:
LoadBalancerDNS:
    Description: The DNS of the Elastic Load Balancer of the web app
```

```
    Value: !GetAtt LoadBalancer.DNSName
LoadBalancerFullName:
    Description: The LoadBalancerFullName of the Elastic Load Balancer
of the web app
    Value: !GetAtt LoadBalancer.LoadBalancerFullName
TargetGroupArn:
    Description: The TargetGroupArn of the Target group
    Value: !GetAtt TargetGroup.TargetGroupArn
```

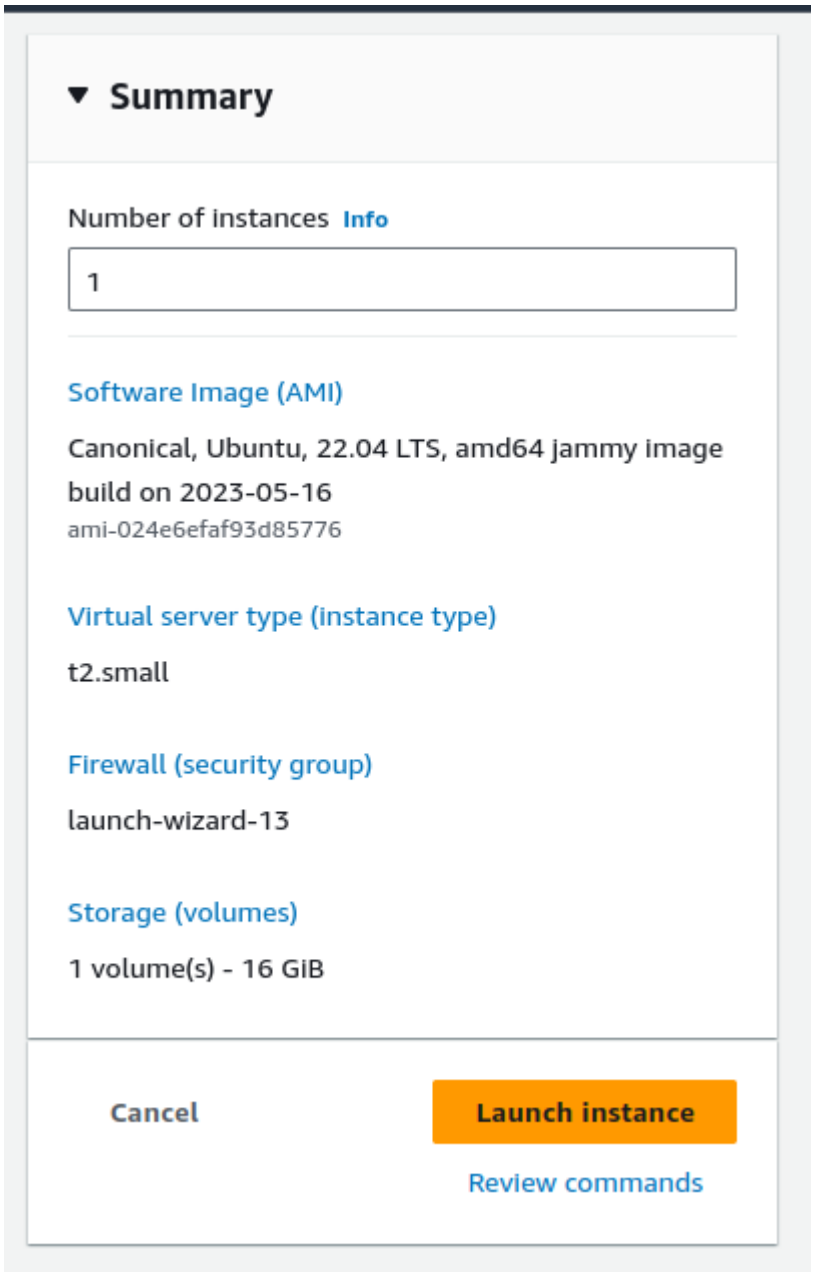
- Modify `Jenkinsfile` in source repository: Look for all lines that contain the `--network-configuration` term and modify its to meet your requirements like below:

```
--network-configuration "awsvpcConfiguration={subnets=
[{YOUR_SUBNET_ID_1},{YOUR_SUBNET_ID_2},{YOUR_SUBNET_ID_3},
{...}],securityGroups=[{YOUR_SECURITY_GROUP_ID_1},
{...}],assignPublicIp=ENABLED}" \
```

- Take a look at `parameters` section in the `Jenkinsfile` and modify parameters' default values so that it meets your needs:
    - AWS account id: AWS_ACCOUNT_ID
    - AWS default region: AWS_REGION
    - Created ECR repo names: *_ECR_REPO
    - Created ECS cluster names: *_CLUSTER
    - Created ECS task definition names: *_TASK_DEFINITION
    - Defined container names in the created task definitions: *_CONTAINER
    - Container port in the created task definitions: CONTAINER_PORT
    - ECS service names for deployment: *_SERVICE
    - Elastic load balancer stack names which will be created during the deployment process: *_LOAD_BALANCER_STACK

3. Setup Jenkins server using AWS EC2:

- Launch an EC2 instance: Remember to expose port 8080 for accessing Jenkins web server



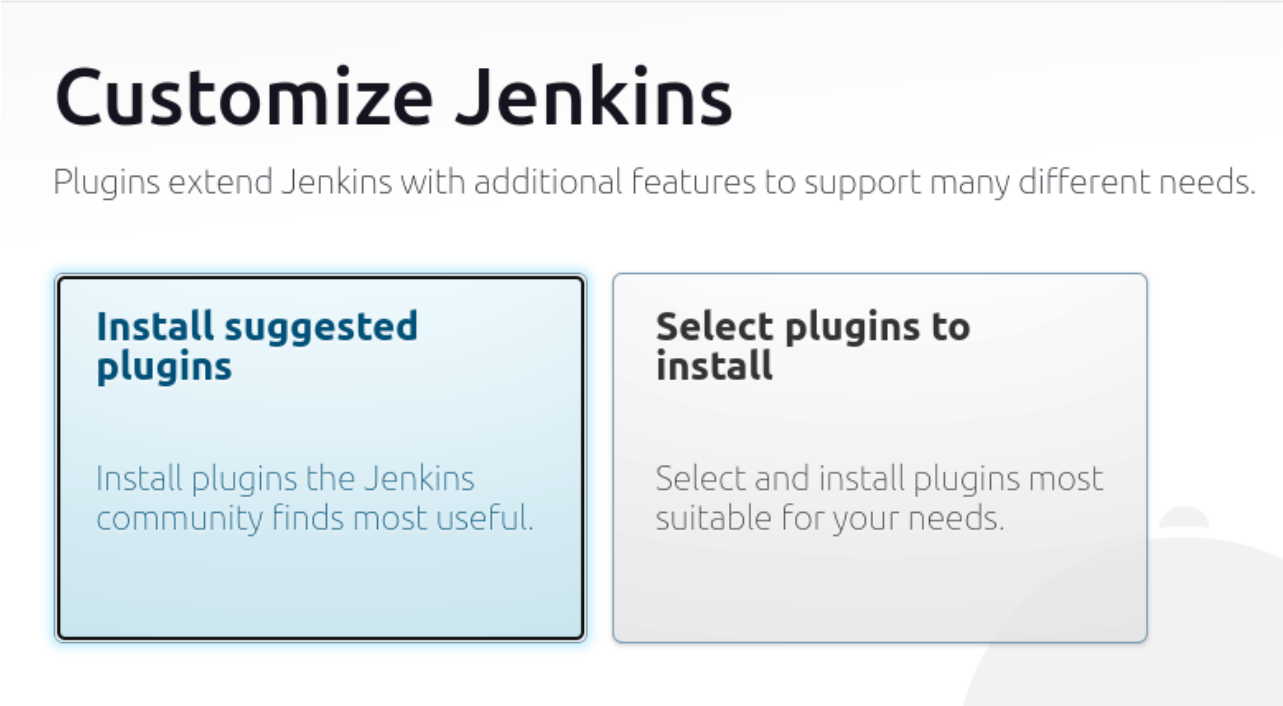
- Copy `./jenkins_setup` folder to the instance. SSH to the EC2 instance, change the working directory to it and run this command:

```
# Inside jenkins_setup folder
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
mkdir jenkins-data jenkins-docker-certs
sudo docker compose up
```

- Access Jenkins web server using the instance IP with port 8080 using the initial password shown in the output of the above commands

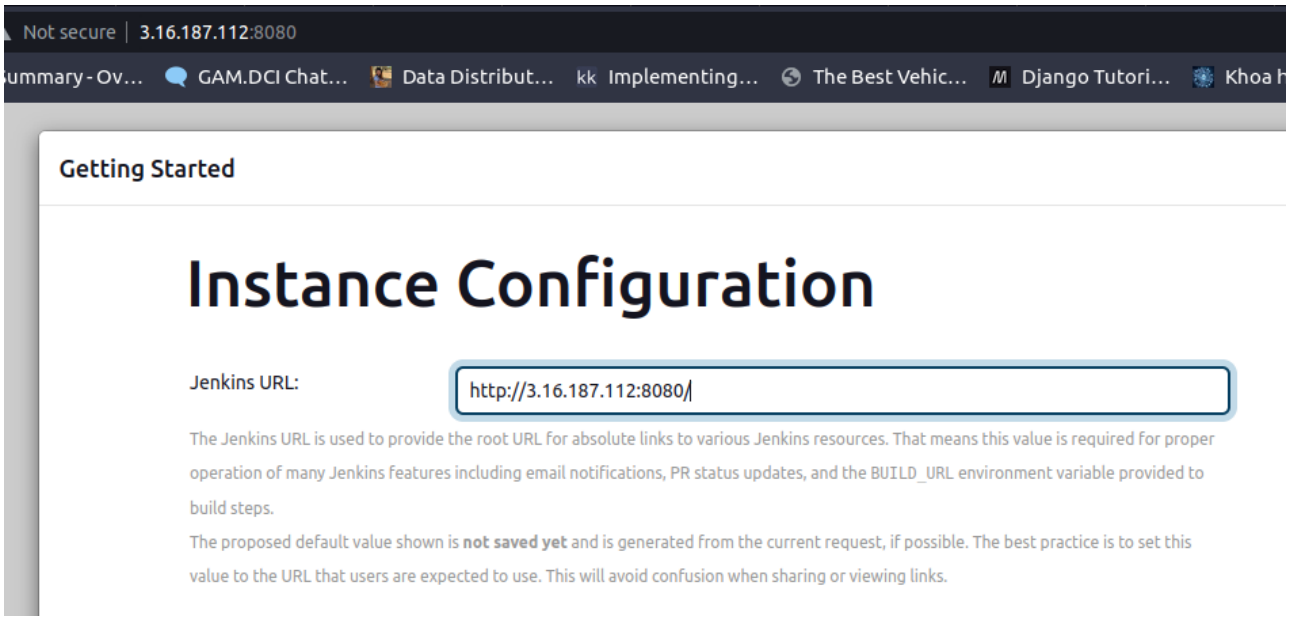
```
**
jenkins-blueocean  | *************************************************************
**
jenkins-blueocean  | *************************************************************
**
jenkins-blueocean  |
jenkins-blueocean  | Jenkins initial setup is required. An admin user has been c
reated and a password generated.
jenkins-blueocean  | Please use the following password to proceed to installatio
n:
jenkins-blueocean  |
jenkins-blueocean  | 8b[redacted]77d67279
jenkins-blueocean  |
jenkins-blueocean  | This may also be found at: /var/jenkins_home/secrets/initia
lAdminPassword
jenkins-blueocean  |
jenkins-blueocean  | *************************************************************
**
jenkins-blueocean  | *************************************************************
**
jenkins-blueocean  | *************************************************************
**
jenkins-blueocean  |
```

- Install plugins:

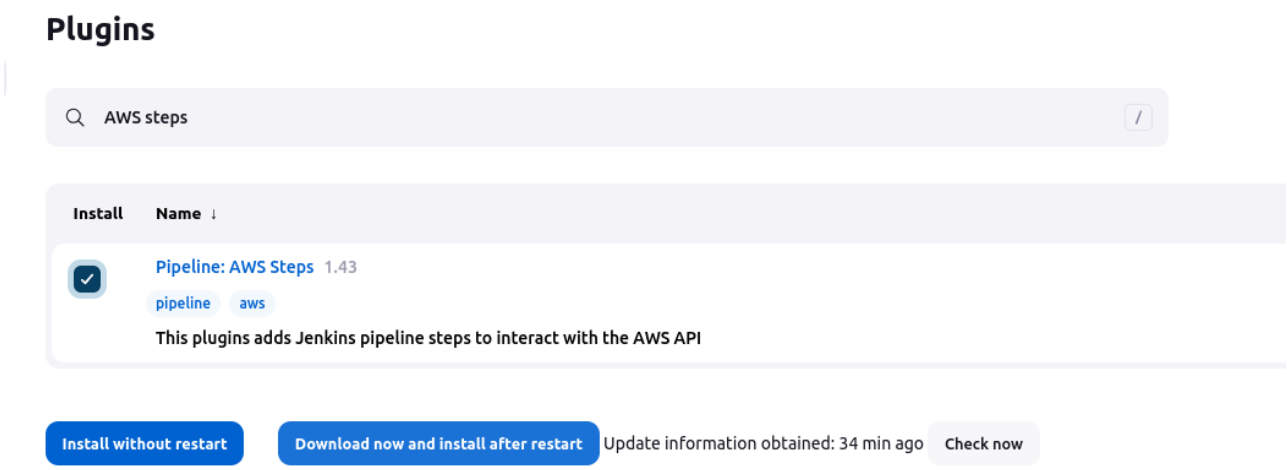


- Create an admin user:

- Assign Jenkins url:



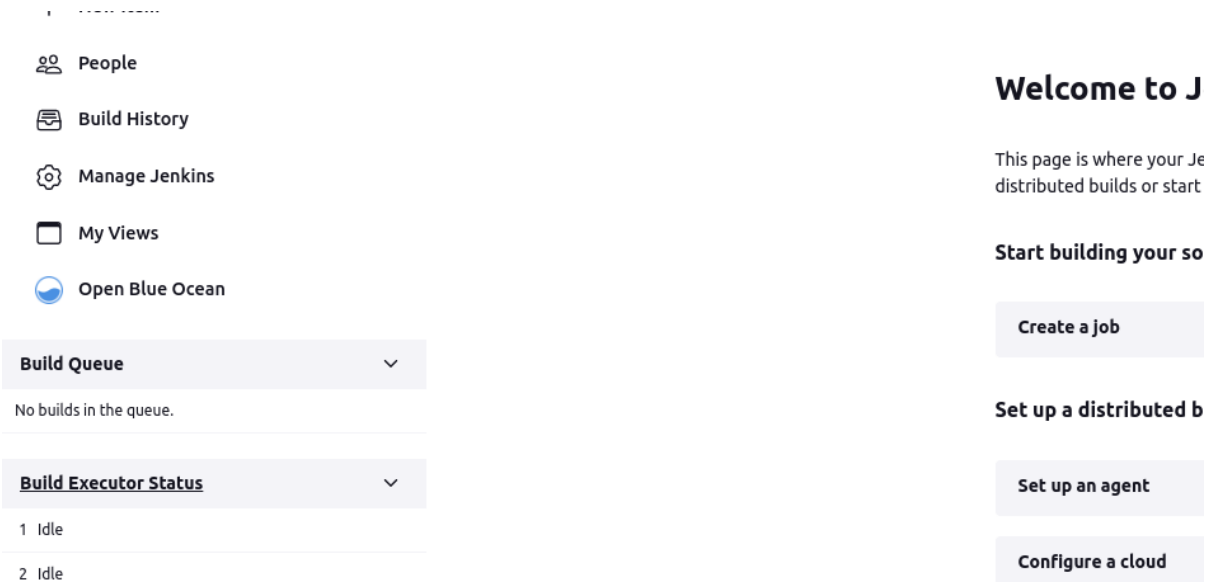
- Install additional plugins: Manage Jenkins -> Plugins -> Available Plugins -> Search `Pipeline: AWS Steps` and check it -> Install without restart -> Go back to the top page



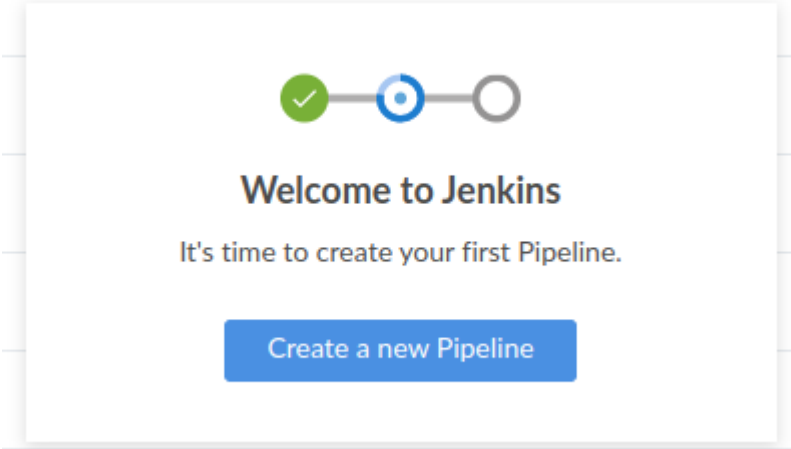
- Add AWS credentials: Manage Jenkins -> Credentials -> System -> Global credentials (unrestricted) -> Add Credentials:

> **IMPORTANT:** Replace 'duongpd7-aws-credentials' in `Jenkinsfile` of the source repository with your ID of the credentials created below

- Set up Jenkins pipeline:
    - Click Open Blue Ocean:

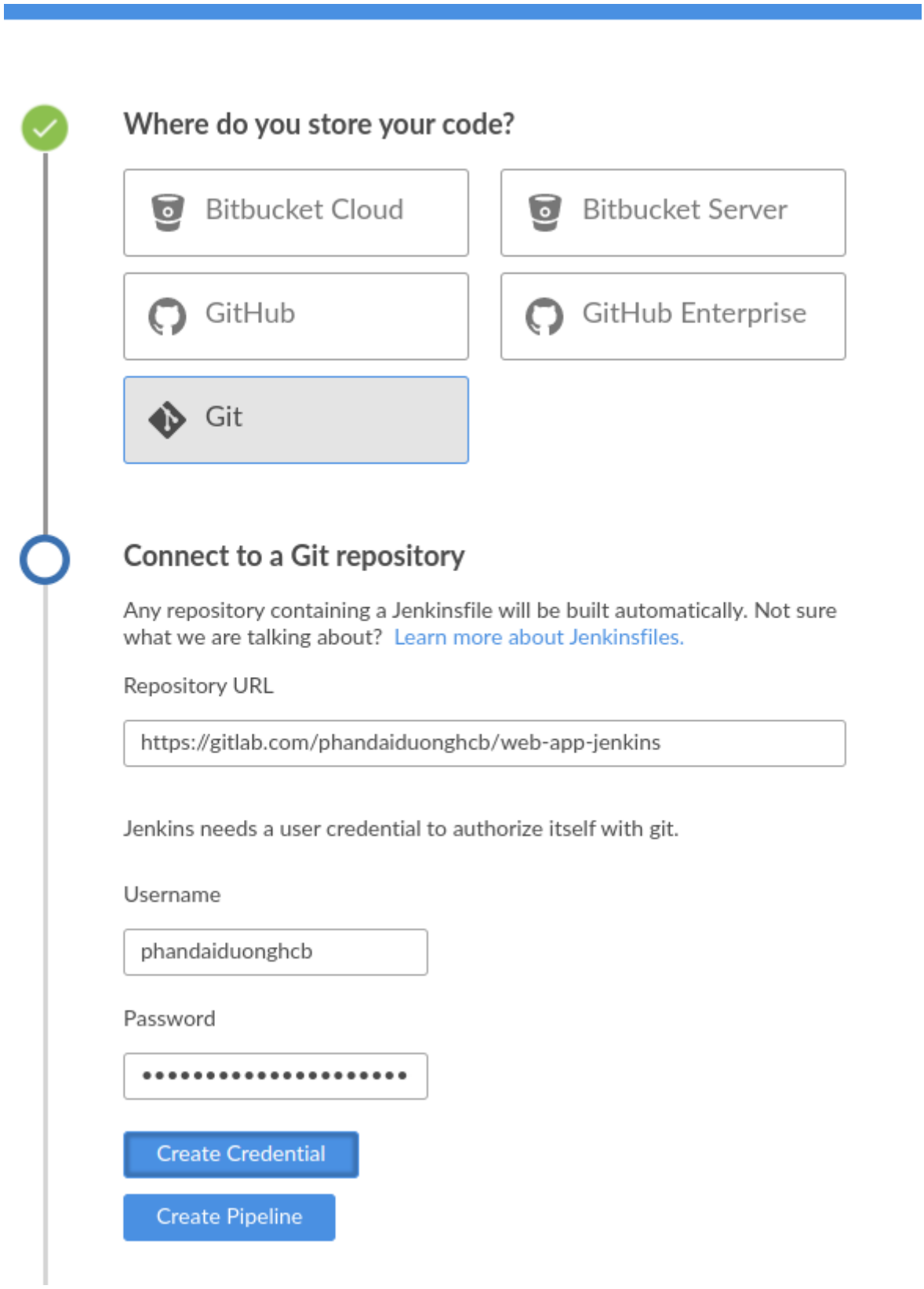


- Create a new Pipeline:



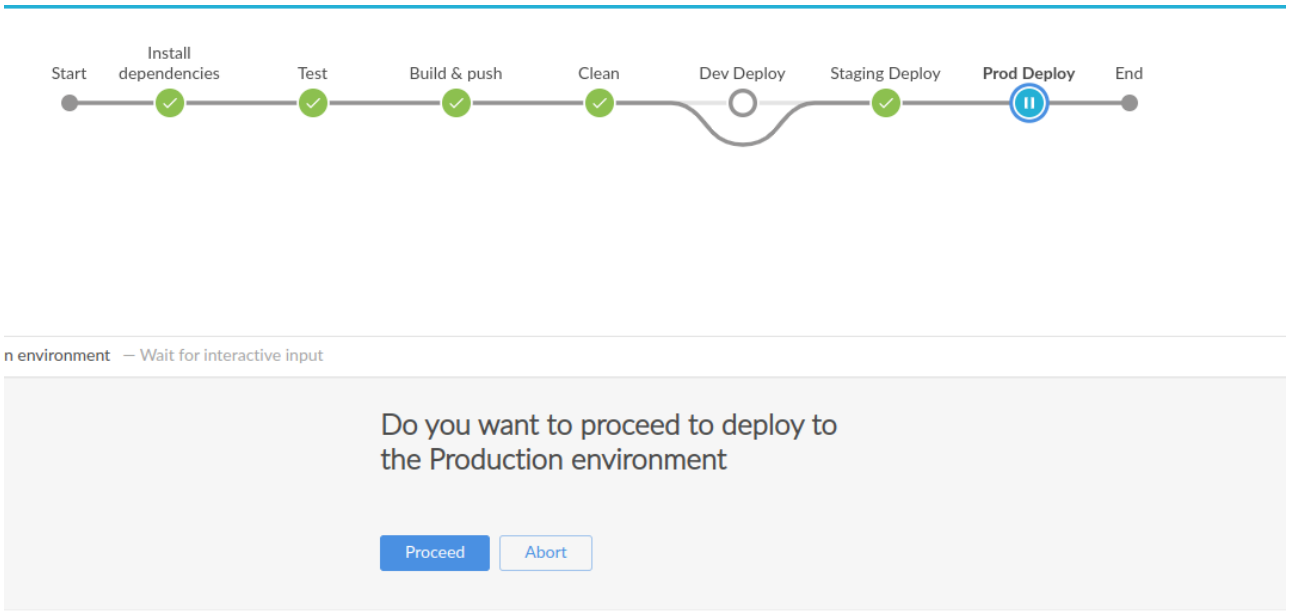
- Configure source repository for the pipeline: choose `Git` , configure your gitlab repository URL, add your credential information and click `Create pipeline`:

> **IMPORTANT**: Assign your Gitlab Personal access token (not your Gitlab password) to the Password field. Refer to the section `Create a personal access token` at Personal access tokens



- After that, the pipeline will be triggered automatically for 2 branches: `master` and `dev`. Both of them should run successfully!

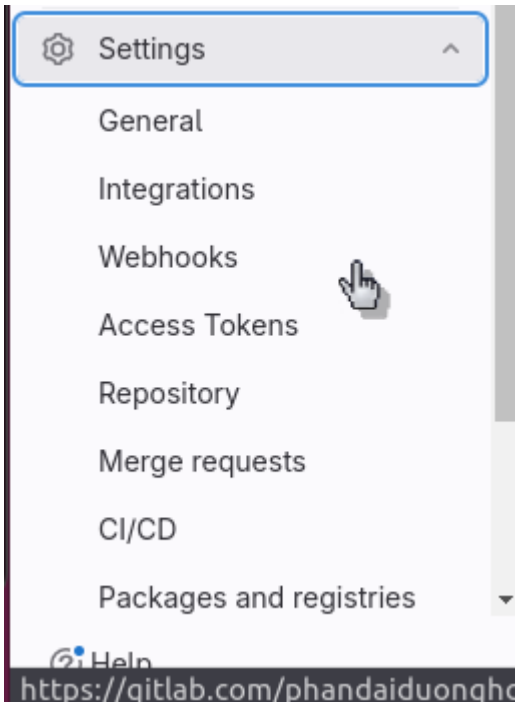
> **Note:** You need to click Proceed to continue to the production deployment process



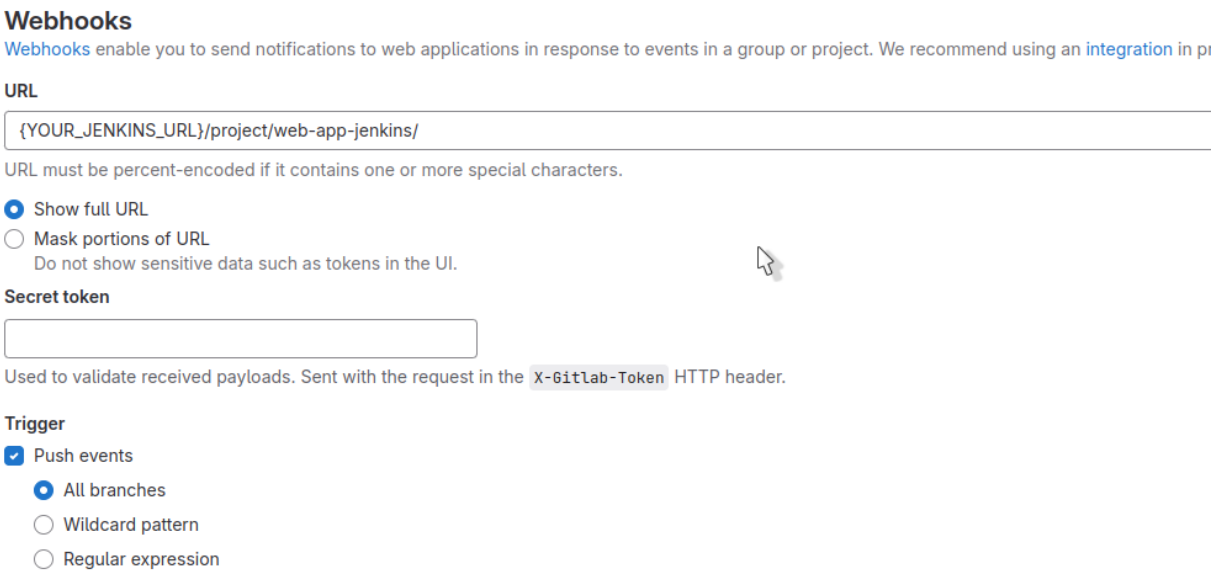
4. Create a Gitlab webhook for triggering the pipeline

- Install Gitlab plugin: Manage Jenkins -> Plugins -> Available Plugins -> Search `Gitlab` and check it -> Install without restart -> Go back to the top page

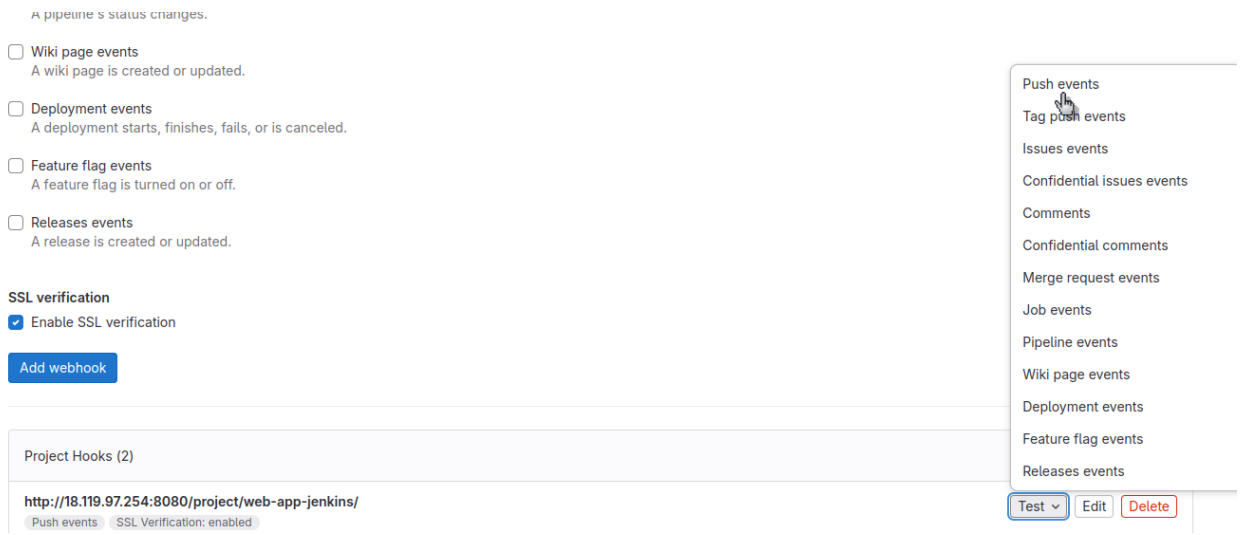
- Acess your Gitlab repository and Click webhook button in the setting section



- Create a webhook like the example below. Replace YOUR_JENKINS_URL with your jenkins server url.



- Test your webhook and the success notification should appear



Hook executed successfully: HTTP 200

- Now Jenkins will trigger the pipeline whenever there is a push to the Gitlab repository

Referenes:

- Installing Jenkins using Docker
- End-to-End Multibranch Pipeline Project Creation
- Building a Jenkins Pipeline with AWS SAM
- Jenkins webhook url for gitlab
- FastAPI