**Setting Up a Basic CI Pipeline on GCP**

**B.Steps required to set up a CI pipeline using CircleCI on GCP**

* Step 1: Set Up a Google Cloud Project
* Step 2: Set Up a Repository and CircleCI Account
* Step 3: Configure CircleCI
* Step 4: Create a .circleci/config.yml File
* Step 5: Commit and Push Changes
* Step 6: Verify Deployment

**C. Define pipeline stages**

**Build:** In CircleCI, the Build stage is where the source code is compiled, dependencies are installed, and any necessary build artifacts are generated. Below are the key steps to configure the Build stage in CircleCI

**Define the .circleci/config.yml File**

**Define a Build Job**

* A build job typically includes the following steps:
* Checkout the code
* Set up the environment (Docker, machine, or executor)
* Install dependencies
* Run the build process
* Store artifacts

version: 2.1

jobs:

build:

docker:

- image: circleci/node:18 # Use a Node.js Docker image

steps:

- checkout # Pulls the latest code

- run:

name: Install Dependencies

command: npm install

- run:

name: Build Application

command: npm run build

- persist\_to\_workspace:

root: .

paths:

- build # Save the build artifacts

workflows:

version: 2

build-workflow:

jobs:

- build

**3. Select the Right Executor**

CircleCI supports different execution environments:

* **Docker** (default for most applications)
* **Machine** (for more system control)
* **MacOS** (for iOS/macOS apps)
* **Windows** (for Windows applications)

jobs:

build:

machine:

image: ubuntu-2204:current

steps:

- checkout

- run: make build

**4. Caching Dependencies**

steps:

- restore\_cache:

keys:

- node-deps-{{ checksum "package-lock.json" }}

- run: npm install

- save\_cache:

paths:

- node\_modules

key: node-deps-{{ checksum "package-lock.json" }}

**5. Storing Build Artifacts**

- store\_artifacts:

path: build

destination: build\_output

**6. Triggering the Build**

workflows:

version: 2

build-workflow:

jobs:

- build:

filters:

branches:

only:

- main

- develop

**Test:** Detail how to run tests as part of the CI process.

The **Test** stage in a CircleCI pipeline ensures code correctness by running unit, integration, and other tests automatically. Below is how to configure tests in your **.circleci/config.yml** file.

**1. Define a Test Job**

A test job includes:

* Checking out the code
* Setting up dependencies
* Running test commands
* Storing test results and artifacts

version: 2.1

jobs:

test:

docker:

- image: circleci/node:18 # Use a Node.js environment

steps:

- checkout # Pull the latest code

- restore\_cache:

keys:

- node-deps-{{ checksum "package-lock.json" }}

- run:

name: Install Dependencies

command: npm install

- save\_cache:

paths:

- node\_modules

key: node-deps-{{ checksum "package-lock.json" }}

- run:

name: Run Tests

command: npm test

- store\_test\_results:

path: test-results # Store test reports

- store\_artifacts:

path: coverage # Save coverage reports

**2. Parallelizing Tests for Speed**

jobs:

test:

docker:

- image: circleci/python:3.10

parallelism: 4 # Run tests across 4 containers

steps:

- checkout

- run:

name: Install Dependencies

command: pip install -r requirements.txt

- run:

name: Run Tests in Parallel

command: pytest --junitxml=test-results/results.xml --maxfail=5 -n 4

- store\_test\_results:

path: test-results

**3. Storing Test Reports**

- store\_test\_results:

path: test-results

**4. Running Tests in Workflows**

workflows:

version: 2

test-workflow:

jobs:

- test:

filters:

branches:

only:

- main

- develop

**5. Handling Failed Tests**

* Set continue-on-error: false (default) to ensure failures block deployment.
* Use notifications (Slack, email) to alert teams about failures**.**

**Deploy: Explain how to deploy the application to a Google Compute Engine instance.**

The **Deploy** stage in CircleCI automates the deployment of your application to a **Google Compute Engine (GCE)** instance. Below are the steps to configure deployment in the **.circleci/config.yml** file.

**1. Prerequisites**

Before setting up deployment, ensure you have:  
✅ A Google Cloud account and a running GCE instance.  
✅ A service account with SSH and deployment permissions.  
✅ The private SSH key added as a **CircleCI Environment Variable** (GCE\_SSH\_KEY).  
✅ The instance’s external IP address or domain.

**2. Create a Deployment Job**

version: 2.1

jobs:

deploy:

docker:

- image: circleci/python:3.10 # Use a lightweight Docker image

steps:

- add\_ssh\_keys:

fingerprints:

- "<<SSH\_KEY\_FINGERPRINT>>" # Add the SSH key stored in CircleCI

- run:

name: Deploy to GCE

command: |

ssh -o "StrictHostKeyChecking=no" ubuntu@YOUR\_GCE\_INSTANCE\_IP << 'EOF'

cd /home/ubuntu/app

git pull origin main

npm install

pm2 restart all # Restart the Node.js app using PM2

EOF

**3. Use Google Cloud SDK for Deployment**

jobs:

deploy:

docker:

- image: google/cloud-sdk # Use Google Cloud SDK

steps:

- checkout

- run:

name: Authenticate with Google Cloud

command: echo "$GCLOUD\_SERVICE\_KEY" | gcloud auth activate-service-account --key-file=-

- run:

name: Set Google Cloud Project

command: gcloud config set project YOUR\_PROJECT\_ID

- run:

name: Copy Application Files

command: gcloud compute scp --recurse . ubuntu@YOUR\_GCE\_INSTANCE\_IP:/home/ubuntu/app

- run:

name: Restart Application

command: gcloud compute ssh ubuntu@YOUR\_GCE\_INSTANCE\_IP --command="cd /home/ubuntu/app && sudo systemctl restart myapp"

**4. Integrate Deployment into CircleCI Workflows**

workflows:

version: 2

deploy-workflow:

jobs:

- test

- deploy:

requires:

- test # Only deploy if tests pass

filters:

branches:

only: main # Deploy only on the main branch

**5. Securely Store Secrets**

Store these as Environment Variables in CircleCI:

* GCE\_SSH\_KEY: Private SSH key for secure connections.
* GCLOUD\_SERVICE\_KEY: JSON key for service account authentication.

**6. Verifying Deployment**

http://YOUR\_GCE\_INSTANCE\_IP