# AWS End-to-End CI/CD Workflow – Python Flask App

## Step 0: AWS Services Overview

- **CodeCommit:** Git-based version control (like GitHub/GitLab). Stores private repositories.
- **CodeBuild:** Serverless build service. Compiles code, runs tests, creates build artifacts.
- **CodeDeploy:** Automates deployment to EC2, ECS, Lambda, or on-prem servers.
- **CodePipeline:** Orchestrates CI/CD workflow: Source → Build → Deploy.
- **CodeArtifact:** Stores build artifacts.
- **KMS:** Encrypts secrets and artifacts.

---

## Step 1: Set Up GitHub Repository

- Create GitHub repo to store your Python app (day-14/simple-python-app/app.py).
- Steps: + → New repository → Name & description → Visibility → Initialize README → Create repository.

---

## Step 2: Create AWS CodePipeline

1. Go to AWS Console → CodePipeline → Create Pipeline.
2. **Source stage:** GitHub → Connect account → Select repository & branch.
3. **Build stage:** AWS CodeBuild → Create project → Configure build environment & commands.
4. **Deploy stage (optional):** Elastic Beanstalk / ECS / Lambda / EC2.
5. Review → Create pipeline.

---

## Step 3: Configure AWS CodeBuild

- Source: AWS CodePipeline → select pipeline.
- Build environment: OS, runtime, compute.
- Buildspec.yml defines the CI steps:

```
version: 0.2

env:
  parameter-store:
    DOCKER_REGISTRY_USERNAME: /myapp/docker-credentials/username
    DOCKER_REGISTRY_PASSWORD: /myapp/docker-credentials/password
    DOCKER_REGISTRY_URL: /myapp/docker-registry/url

phases:
  install:
    runtime-versions:
      python: 3.11
    commands:
      - pip install -r $CODEBUILD_SRC_DIR/aws-devops-zero-to-hero/day-14/simple-python-
app/requirements.txt
  pre_build:
    commands:
      - echo "$DOCKER_REGISTRY_PASSWORD" | docker login -u
"$DOCKER_REGISTRY_USERNAME" --password-stdin
  build:
    commands:
      - docker build -t "$DOCKER_REGISTRY_USERNAME/simple-python-flask-app:latest"
$CODEBUILD_SRC_DIR/aws-devops-zero-to-hero/day-14/simple-python-app
      - docker push "$DOCKER_REGISTRY_USERNAME/simple-python-flask-app:latest"
  post_build:
    commands:
      - echo "Build completed successfully!"

artifacts:
  files:
    - '**/*'
  base-directory: $CODEBUILD_SRC_DIR/aws-devops-zero-to-hero/day-14/simple-python-app
```

- **Key Notes:**
    - Parameter Store stores sensitive info (Docker creds, API keys).
    - Fully automates Docker build & push.
    - Compatible with CodePipeline → triggers on GitHub commit.

---

**Step 4: Trigger the CI Process**

- Make changes in GitHub repo → Commit & push.
- CodePipeline detects changes → triggers pipeline → invokes CodeBuild → builds & pushes Docker image.
- Optional deploy via CodeDeploy.

---

**Step 5: Comparison with Jenkins**

| **Feature** | **Jenkins** | **AWS CodeBuild/CodePipeline** |
|---|---|---|
| CI/CD Steps | Jenkinsfile (Groovy DSL) | buildspec.yml (YAML) |
| Trigger Build | Manual / SCM polling | Auto via CodePipeline |
| Build Environment | Self-managed servers | Prebuilt images on CodeBuild |
| Secrets Management | Global config / plugins | Parameter Store |

---

**Step 6: Integration Flow Diagram**

```
+-------------------+
|   GitHub Repo     |
| day-14/simple-    |
| python-app/app.py |
+--------+----------+
         |
         | Commit / Push
         v
+-------------------+
|   CodePipeline    |
| sample-python-app |
+---------+---------+
          |
          v
+-------------------+
|    Source Stage   |
| Pull latest code  |
| from GitHub       |
+-------------------+
          |
          v
+-------------------+
|     Build Stage   |
|  AWS CodeBuild    |
| Build & Push Docker|
| image to registry |
+-------------------+
          |
          v
+-------------------+
|  (Optional Deploy)|
| CodeDeploy / ECS  |
+-------------------+
```

✅ **Key Takeaways**

- commit → pipeline → build → docker push → optional deploy

- buildspec.yml = Jenkinsfile equivalent
- Secrets handled securely → Parameter Store
- Fully automated CI/CD for your Python Flask app

**CD playbook: GitHub → CodePipeline → CodeBuild → CodeDeploy → EC2**

**0) prerequisites (once)**

- AWS account + chosen **region**.
- A **GitHub repo** with:
  - your Flask app
  - buildspec.yml (build & push image + package deploy artifact)
  - appspec.yml (CodeDeploy)
  - scripts/ (start/stop/health)
- A **CodePipeline** and **CodeBuild** project from your CI part (already done).
- Docker image name you push (e.g. DOCKER_USER/simple-python-flask-app:latest).
- **Parameter Store** (if using private Docker Hub creds):
  - /myapp/docker-credentials/username (SecureString)
  - /myapp/docker-credentials/password (SecureString)

---

**1) create the EC2 instance (targets to deploy to)**

1. **Launch** EC2 (Amazon Linux 2 or Ubuntu 22.04).
   Security Group: allow inbound **80/5000** (your app port), and **22** (SSH) if needed.
2. **Attach IAM role** to EC2 (for agent + SSM + optional Parameter Store):
   - Managed policies (minimal & practical):
     - AmazonSSMManagedInstanceCore (lets SSM manage & gives basic STS/SSM)
     - CloudWatchAgentServerPolicy (optional logs)
     - If you will read secrets from Parameter Store on the instance, add an **inline policy**:
     - {
     -   "Version": "2012-10-17",

        - "Statement": [
        - { "Effect": "Allow", "Action": ["ssm:GetParameter","ssm:GetParameters"], "Resource": [
        - "arn:aws:ssm:*:*:parameter/myapp/docker-credentials/*",
        - "arn:aws:ssm:*:*:parameter/myapp/docker-registry/*"
        - ]},
        - { "Effect": "Allow", "Action": ["kms:Decrypt"], "Resource": "*" }
        - ]
        - }
        - (If you deploy from **S3** artifacts, the CodeDeploy agent handles downloads via the CodeDeploy service; the instance role above is still fine.)

3. **Install CodeDeploy agent**
    - *Amazon Linux 2*:
    - sudo yum update -y
    - sudo yum install -y ruby wget
    - REGION=<your-region>
    - cd /tmp
    - wget https://aws-codedeploy-${REGION}.s3.${REGION}.amazonaws.com/latest/install
    - chmod +x install
    - sudo ./install auto
    - sudo systemctl enable codedeploy-agent
    - sudo systemctl start codedeploy-agent
    - sudo systemctl status codedeploy-agent
    - *Ubuntu*:
    - sudo apt update -y
    - sudo apt install -y ruby-full wget
    - REGION=<your-region>

- cd /tmp
- wget https://aws-codedeploy-${REGION}.s3.${REGION}.amazonaws.com/latest/install
- chmod +x install
- sudo ./install auto
- sudo systemctl enable codedeploy-agent
- sudo systemctl start codedeploy-agent
- sudo systemctl status codedeploy-agent

4. **Install Docker** (if running container on EC2):
   - Amazon Linux 2:
   - sudo amazon-linux-extras install docker -y
   - sudo systemctl enable docker
   - sudo systemctl start docker
   - sudo usermod -aG docker ec2-user
   - Ubuntu:
   - sudo apt-get install -y docker.io
   - sudo systemctl enable docker
   - sudo systemctl start docker
   - sudo usermod -aG docker $USER

*(Reboot or re-login for docker group to take effect.)*

5. **Tag the EC2 instance** (for dynamic targeting by CodeDeploy):
   - Key: Name, Value: sample-python (use exactly what you'll use in the deployment group)

---

**2) create CodeDeploy application & deployment group**

1. **Application**
   - Name: sample-python-flask-app
   - Compute platform: **EC2/On-premises**
2. **Service role for CodeDeploy** (once)

    - Create IAM role: CodeDeployServiceRole
    - Attach AWS managed policy: AWSCodeDeployRole

3. **Deployment group**
    - Name: sample-python-deployment-group
    - Service role: CodeDeployServiceRole
    - Environment config: **Amazon EC2 instances**
    - **Tag filters**: Key Name, Value sample-python
    - Deployment type: **In-place**
    - Deployment config (strategy): choose one
        - AllAtOnce (fast for test)
        - OneAtATime (safer)
        - or a custom %/batch
    - (Optional) Load balancer & rollback/alarms for prod.

---

**3) repository layout for deployment (what CodeDeploy needs)**

```
your-repo/
  aws-devops-zero-to-hero/day-14/simple-python-app/
    app.py
    requirements.txt
    ...
  appspec.yml
  scripts/
    start_container.sh
    stop_container.sh
    health_check.sh
```

**appspec.yml (EC2, Docker-based)**

```
version: 0.0
os: linux
```

```
hooks:
  ApplicationStop:
    - location: scripts/stop_container.sh
      timeout: 300
      runas: root

  BeforeInstall:
    - location: scripts/stop_container.sh
      timeout: 300
      runas: root

  AfterInstall:
    - location: scripts/start_container.sh
      timeout: 600
      runas: root

  ApplicationStart:
    - location: scripts/health_check.sh
      timeout: 300
      runas: root
```

**scripts/stop_container.sh**

```
#!/usr/bin/env bash
set -e
CONTAINER_NAME="simple-python-flask-app"
if [ "$(docker ps -q -f name=${CONTAINER_NAME})" ]; then
  docker stop ${CONTAINER_NAME} || true
fi
```

```
if [ "$(docker ps -aq -f status=exited -f name=${CONTAINER_NAME})" ]; then
  docker rm ${CONTAINER_NAME} || true
fi
```

**scripts/start_container.sh**

```
#!/usr/bin/env bash
set -e

# optional: fetch docker hub creds from Parameter Store (if your repo is private)
USERNAME_PARAM="/myapp/docker-credentials/username"
PASSWORD_PARAM="/myapp/docker-credentials/password"

# if aws cli not present, install a minimal version (AL2 usually has it)
if ! command -v aws >/dev/null 2>&1; then
  echo "aws cli not found; please install or bake it into the AMI."
fi

DOCKER_USERNAME=$(aws ssm get-parameter --name "$USERNAME_PARAM" --with-decryption --query 'Parameter.Value' --output text || echo "")
DOCKER_PASSWORD=$(aws ssm get-parameter --name "$PASSWORD_PARAM" --with-decryption --query 'Parameter.Value' --output text || echo "")

# login only if both present (skip for public images)
if [ -n "$DOCKER_USERNAME" ] && [ -n "$DOCKER_PASSWORD" ]; then
  echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
fi

IMAGE="${DOCKER_USERNAME:-<public_user>}/simple-python-flask-app:latest"
CONTAINER_NAME="simple-python-flask-app"
```

```
docker pull "$IMAGE"


# run on port 80 (or 5000); adapt as needed
if [ "$(docker ps -aq -f name=${CONTAINER_NAME})" ]; then
  docker rm -f ${CONTAINER_NAME} || true
fi


docker run -d --name ${CONTAINER_NAME} -p 80:5000 "$IMAGE"
```

**scripts/health_check.sh**

```
#!/usr/bin/env bash
set -e
# wait a few seconds for container to boot
sleep 5
curl -fsS http://localhost/ || (echo "health check failed" && exit 1)
```

If your Flask app listens on 5000, we mapped container 5000 → host 80. Visiting the EC2 public DNS on port 80 should work.

---

**4) adjust your buildspec (CI) to output the deploy bundle**

You're already building & pushing the image. Add **artifact packaging** so CodePipeline → CodeDeploy can consume appspec.yml + scripts.

**buildspec.yml (example—merge with your existing one):**

```
version: 0.2


env:
  parameter-store:
    DOCKER_REGISTRY_USERNAME: /myapp/docker-credentials/username
    DOCKER_REGISTRY_PASSWORD: /myapp/docker-credentials/password
    DOCKER_REGISTRY_URL: /myapp/docker-registry/url
```

```
phases:
  install:
    runtime-versions:
      python: 3.11
    commands:
      - echo "Installing deps..."
      - pip install -r $CODEBUILD_SRC_DIR/aws-devops-zero-to-hero/day-14/simple-python-
app/requirements.txt

  pre_build:
    commands:
      - echo "Docker login..."
      - echo "$DOCKER_REGISTRY_PASSWORD" | docker login -u
"$DOCKER_REGISTRY_USERNAME" --password-stdin

  build:
    commands:
      - echo "Building & pushing image..."
      - docker build -t "$DOCKER_REGISTRY_USERNAME/simple-python-flask-app:latest"
$CODEBUILD_SRC_DIR/aws-devops-zero-to-hero/day-14/simple-python-app
      - docker push "$DOCKER_REGISTRY_USERNAME/simple-python-flask-app:latest"
      - echo "Packaging deploy artifact..."
      - mkdir -p artifact
      - cp appspec.yml artifact/
      - cp -r scripts artifact/

  post_build:
    commands:
      - echo "Build & package completed."
```

```
artifacts:
  files:
    - appspec.yml
    - scripts/**/*
  base-directory: artifact
```

Result: CodeBuild publishes an artifact zip containing **appspec.yml** + **scripts/**.
CodeDeploy will run those on the EC2 instance.

---

**5) add/update the Deploy stage in CodePipeline**

1. **Edit** your existing pipeline → **Add stage** → Deploy.
2. **Add action**:
    - Action provider: **AWS CodeDeploy**
    - Input artifact: output of your Build stage (the artifact zip)
    - Application name: sample-python-flask-app
    - Deployment group: sample-python-deployment-group
3. **Save** pipeline.

Now the pipeline order is: **Source (GitHub) → Build (CodeBuild) → Deploy (CodeDeploy)**.

---

**6) run a deployment**

1. Commit & push any change to your GitHub repo.
2. Watch CodePipeline:
    - **Source**: gets latest commit
    - **Build**: builds & pushes Docker image, outputs artifact
    - **Deploy**: CodeDeploy sends artifact to the instance(s); agent runs hooks:
        - Stop old container
        - Pull new image
        - Start new container

        - Health check

3. Test in browser: http://<EC2-Public-DNS>/

---

**7) scale from 1 to many EC2s (tags = magic)**

- Add the same tag (e.g., Name=sample-python) to every EC2 you want in the deployment group.
- CodeDeploy **auto-targets** those instances.
- Choose a safer deployment config for prod (e.g., **OneAtATime** / batch %).

---

**8) troubleshooting quickies**

- **Agent down**: sudo systemctl status codedeploy-agent (start/enable if needed)
- **Permissions**:
    - CodeDeploy **service role** must exist and be selected in the deployment group.
    - EC2 **instance role** must allow SSM GetParameter/KMS Decrypt if you use Parameter Store.
- **Docker permission**: add your user to docker group; run hooks as root in appspec (as shown).
- **Ports**: open SG port **80** (or whatever you map).
- **Logs**:
    - /opt/codedeploy-agent/deployment-root/deployment-logs/codedeploy-agent-deployments.log
    - /var/log/aws/codedeploy-agent/codedeploy-agent.log
    - Your script echoes show up in CodeDeploy console events.

---

**9) alternative (no Docker)**

If you prefer deploying raw Flask + gunicorn with systemd, swap scripts to:

- install venv + deps in AfterInstall
- create a systemd unit to run gunicorn on :80 or behind nginx

- start service in ApplicationStart
  *(I can drop in ready app.service + scripts if you want this path.)*

---

**10) final mental model**

- **CodePipeline** = conductor
- **CodeBuild** = builds image + packages appspec.yml & scripts/
- **CodeDeploy** = ships & runs hooks on tagged EC2s
- **EC2** = runs CodeDeploy agent + Docker → serves your Flask app