Author: Madhurima Rawat

Setting Up Cloud-based CICD Pipeline

This experiment sets up a Continuous Integration/Continuous Deployment (CI/CD) pipeline using GitHub Actions, Docker, and LocalStack to simulate AWS services. It provides hands-on experience in automating deployments with AWS CLI and S3, demonstrating cloud-based automation workflows.

Understanding CI/CD Pipelines with GitHub Actions, Docker, and LocalStack

📌 Overview

This experiment demonstrates how to set up a **Continuous Integration/Continuous Deployment (CI/CD) pipeline** using **GitHub Actions, Docker, and LocalStack**. It provides practical insights into **automating deployments** with AWS services, particularly using AWS CLI and S3.

This setup helps developers and DevOps engineers simulate **AWS services locally** while streamlining cloud-based deployment workflows.

🚀 How It Works

1️⃣ GitHub Actions:

- Automates build, test, and deployment processes directly from GitHub.
- Triggers workflows on **code commits, pull requests, or scheduled intervals**.

2️⃣ Docker:

- Creates **containerized environments** for running applications.
- Ensures that the pipeline runs consistently across different systems.

3️⃣ LocalStack:

- Simulates AWS cloud services locally (S3, Lambda, DynamoDB, etc.).
- Allows developers to test AWS-related workflows without real AWS costs.

4️⃣ AWS CLI & S3:

- AWS CLI automates interactions with AWS services.
- S3 (Simple Storage Service) acts as a storage bucket for deployment artifacts.

🔹 Use Cases

✅ Automated Deployment Pipelines

- Code is automatically tested, built, and deployed to cloud environments.
- Reduces manual intervention, ensuring faster release cycles.

✅ Simulating AWS Services Locally

- Developers can test AWS-dependent applications **without incurring AWS costs**.
- Ideal for **offline development** or **local testing** of cloud-native applications.

✅ Cloud-Based Workflow Testing

- Ensures **infrastructure as code** (IaC) principles by defining cloud setups in version control.
- Useful for DevOps teams deploying applications on AWS.

✅ Disaster Recovery & Backup Automation

- CI/CD can automate the creation of **S3 backups** and deployment rollbacks.
- Helps maintain data integrity and business continuity.

✅ Microservices & Serverless Development

- Supports **Lambda function deployment**, API Gateway integration, and event-driven applications.
- Helps teams working on **serverless computing** streamline their workflow.

🌎 Real-Life Examples

📌 E-commerce Platforms

- Deploy new features to AWS-hosted websites seamlessly without downtime.
- Test changes in a **LocalStack AWS simulation** before pushing them live.

📌 Financial Services

- Automate deployment of **fraud detection algorithms** in a secure pipeline.
- Ensure compliance by **testing AWS interactions locally** before deploying.

📌 Mobile App Backend Development

- Automatically deploy backend APIs (hosted on AWS Lambda) after each successful commit.
- Use LocalStack to **test S3 storage operations without using real AWS resources**.

📌 AI/ML Model Deployment

- Automate pushing trained ML models to S3 for cloud inference.

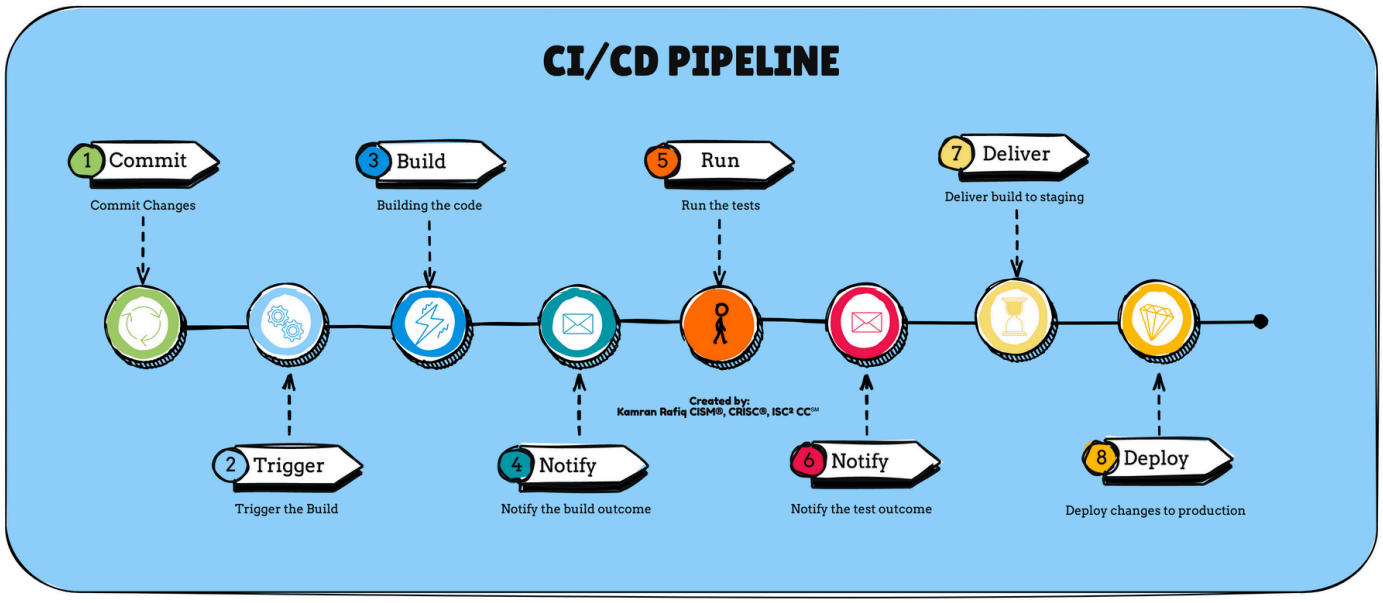
- Use GitHub Actions to validate the model before deployment.

🎯 Key Benefits

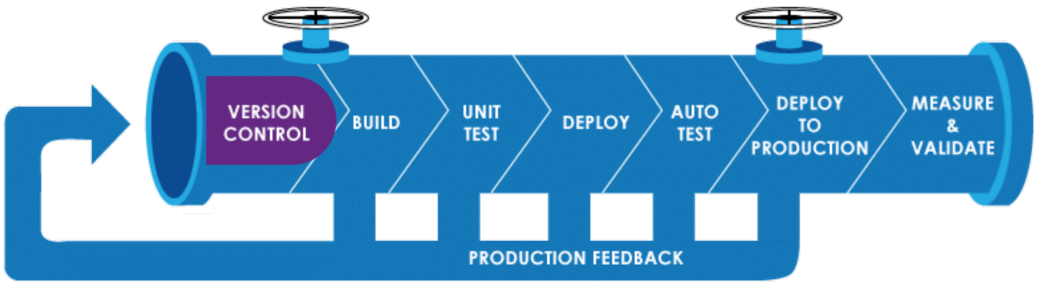
- 🔹 **Faster Development Cycles** → Reduces manual deployment efforts.
- 🔹 **Cost-Efficient Testing** → Simulates AWS without incurring costs.
- 🔹 **Reliable Cloud Automation** → Ensures seamless integration & deployment.
- 🔹 **Enhanced Security** → Controlled CI/CD workflow reduces human errors.
- 🔹 **Scalability** → Easily extendable for various AWS services.

📷 Visual Representation

🔗 Version Control in CI/CD



🔗 CI/CD Pipeline Execution



🚀 Conclusion

This **CI/CD pipeline experiment** with GitHub Actions, Docker, and LocalStack demonstrates how to **automate deployments, streamline cloud workflows, and reduce AWS costs**. It is an essential practice for teams looking to improve **deployment efficiency and cloud-native development.**

CI/CD Pipelines with GitHub Actions, Docker, and LocalStack

1. Creating an S3 Bucket

Command:

```
aws --endpoint-url=http://localhost:4566 s3 mb s3://my-ci-cd-artifacts
```

Explanation:

- `aws s3 mb` → Creates a new S3 bucket.
- `s3://my-ci-cd-artifacts` → The name of the bucket being created.
- `--endpoint-url=http://localhost:4566` → Uses **LocalStack** to simulate AWS services.

Output:

```
make_bucket: my-ci-cd-artifacts
```

2. Attempting to Create a CodeCommit Repository

Command:

```
aws --endpoint-url=http://localhost:4566 codecommit create-
repository --repository-name my-repo
```

Explanation:

- `aws codecommit create-repository` → Creates a new AWS CodeCommit repository.
- `--repository-name my-repo` → Assigns the repository name as `my-repo` .
- `--endpoint-url=http://localhost:4566` → Uses **LocalStack**.

Error Output:

```
An error occurred (InternalFailure) when calling the
CreateRepository operation: API for service 'codecommit'
not yet implemented or pro feature - please check
https://docs.localstack.cloud/references/coverage/ for further information
```

3. Initializing a Git Repository

Command:

```
git init
```

Explanation:

- `git init` → Initializes a new **Git repository** in the current directory.

Output:

```
Initialized empty Git repository in C:/Users/rawat/Documents/8
SEMESTER/Cloud Computing/Lab/Experiment 10/Codes/.git/
```

4. Staging and Committing Files

Commands:

```
git add .
git commit -m "Initial commit"
```

Explanation:

- `git add .` → Stages all files for commit.
- `git commit -m "Initial commit"` → Commits the staged files with a message.

Output:

```
[master (root-commit) 2dfb5b6] Initial commit
 3 files changed, 1153 insertions(+)
 create mode 100644 Command Prompt Input and Output Explanation.md
 create mode 100644 Command Prompt Input and Output Explanation.pdf
 create mode 100644 Command Prompt Input and Output.txt
```

5. Uploading a ZIP File to S3

Command:

```
aws --endpoint-url=http://localhost:4566 s3 cp my-code.zip
s3://my-ci-cd-artifacts/
```

Explanation:

- `aws s3 cp` → Copies a file to S3.
- `my-code.zip` → The file being uploaded.
- `s3://my-ci-cd-artifacts/` → Destination bucket in S3.
- `--endpoint-url=http://localhost:4566` → Uses **LocalStack**.

Error Output:

```
The user-provided path my-code.zip does not exist.
```

6. Creating a ZIP Archive

Command:

```
powershell Compress-Archive -Path * -DestinationPath my-code.zip
```

Explanation:

- `Compress-Archive -Path * -DestinationPath my-code.zip` → Creates a ZIP archive of all files in the directory.

7. Uploading the ZIP File Again

Command:

```
aws --endpoint-url=http://localhost:4566 s3 cp my-code.zip
s3://my-ci-cd-artifacts/
```

Output:

```
upload: .\my-code.zip to s3://my-ci-cd-artifacts/my-code.zip
```

8. Listing the Uploaded Files in S3

Command:

```
aws --endpoint-url=http://localhost:4566 s3 ls s3://my-ci-cd-artifacts/
```

Output:

```
2025-03-08 10:32:42     289415 my-code.zip
```

9. Setting Up a Remote Git Repository

Commands:

```
git remote add origin https://github.com/madhurimarawat/Cloud-
Computing.git
git branch -M main
git push -u origin main
```

Explanation:

- `git remote add origin <repo-url>` → Links the local repository to GitHub.
- `git branch -M main` → Renames the current branch to `main`.
- `git push -u origin main` → Pushes the code to GitHub.

Error Output:

```
To https://github.com/madhurimarawat/Cloud-Computing.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to
'https://github.com/madhurimarawat/Cloud-Computing.git'
hint: Updates were rejected because the remote contains
work that you do not
hint: have locally. This is usually caused by another
repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
```

Fix:

To resolve this issue, run:

```
git pull origin main --rebase
git push -u origin main
```

10. Pulling the Latest Changes from GitHub

Command:

```
git pull origin main --rebase
```

Explanation:

- Fetches changes from the **remote repository** and applies them using **rebase** instead of a merge.
- Ensures a linear commit history by reapplying local changes on top of the latest remote changes.

Output:

```
remote: Enumerating objects: 240, done.
remote: Counting objects: 100% (240/240), done.
remote: Compressing objects: 100% (212/212), done.
remote: Total 240 (delta 100), reused 43 (delta 21), pack-reused 0
Receiving objects: 100% (240/240), 9.22 MiB | 1.11 MiB/s, done.
Resolving deltas: 100% (100/100), done.
From https://github.com/madhurimarawat/Cloud-Computing
 * branch            main       -> FETCH_HEAD
 * [new branch]      main       -> origin/main
Successfully rebased and updated refs/heads/main.
```

11. Staging All Changes

Command:

```
git add .
```

Explanation:

- Stages all modified and newly created files in the **current directory** for the next commit.

12. Checking for an Ongoing Rebase

Command:

```
git rebase --continue
```

Explanation:

- Used to **continue** an ongoing rebase operation if there are conflicts.
- In this case, the **error** means there was **no ongoing rebase**, so this step was unnecessary.

Output:

```
fatal: no rebase in progress
```

13. Pushing Changes to GitHub

Command:

```
git push -u origin main
```

Explanation:

- Pushes local changes to the **remote repository** (`origin`), setting `main` as the **upstream branch**.
- This makes future `git push` commands **simpler** by automatically pushing to `origin main` .

Output:

```
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 282.43 KiB | 31.38 MiB/s, done.
Total 5 (delta 1), reused 1 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/madhurimarawat/Cloud-Computing.git
   b201b02..eb4faf7  main -> main
branch 'main' set up to track 'origin/main'.
```

Output Breakdown:

- **Delta compression** → Reduces the size of transmitted data.

- **Objects written successfully** → Confirms the **push** was successful.
- **Tracking branch set up** → Future `git push` commands will default to `origin main`.

14. Viewing the YAML Deployment Workflow

Link:

View the deployment YAML file

Purpose:

This GitHub Actions workflow automates a **manual deployment process** by performing the following steps:

Workflow Breakdown:

- **Triggering the Workflow**
  - The workflow is **manually triggered** using `workflow_dispatch`, meaning it does **not** run automatically on commits or merges.
- **Job Execution**
  - A **single job** named `deploy` is executed on **Ubuntu-latest**, the default GitHub-hosted runner.
- **Steps in the Workflow**

  i. **Checkout Repository**
  - Uses `actions/checkout@v4` to fetch the repository contents into the GitHub Actions runner.

  ii. *(Optional)* **Install AWS CLI**
  - This step is commented out but would install the AWS CLI if needed.

  iii. *(Optional)* **Zip the Repository**
  - Another commented-out step that creates a ZIP archive of the repository.

  iv. *(Optional)* **Upload to LocalStack S3**
  - Demonstrates an attempt to upload the ZIP file to a **LocalStack S3 bucket**.
  - ⚠️ **This step does NOT work in GitHub Actions**, since LocalStack would need to be running on the same machine.

v. **Print Success Message**

- Simply prints `"Successfully run!"` to indicate that the workflow has been executed.

Key Considerations:

- This workflow is primarily a **template** for deploying to **LocalStack S3**.
- Since GitHub Actions **runs on cloud-hosted VMs**, it cannot access **LocalStack running locally**.
- We can modify this workflow to deploy to a real AWS S3 bucket by configuring proper AWS credentials.