Lab 5: Continuous Integration (CI) and Continuous Delivery (CD) with Kubernetes

> **Author:** Badr TAJINI - Cloud-native-DevOps-with-Kubernetes - ESIEE - 2024/2025

This lab guides you through setting up CI/CD pipelines using GitHub Actions, demonstrating integration with AWS using OpenID Connect (OIDC), and automating deployments with OpenTofu.

Objectives:

- Set up CI with automated tests (application and infrastructure).
- Configure OIDC authentication with AWS.
- Create an automated deployment pipeline with GitHub Actions and OpenTofu.
- Understand and apply different deployment strategies.
- Explore the concept of GitOps with Flux.

Prerequisites:

- GitHub account
- AWS account
- Local Kubernetes cluster (Docker Desktop with Kubernetes enabled or Minikube)
- Installed and configured tools: Git, Docker, `kubectl`, OpenTofu, `npm`, Node.js, `aws-cli`, `flux`

Part 1: Continuous Integration (CI)

Continuous Integration (CI) involves frequently merging code into a main branch (`main`), ideally several times a day. This allows for early detection and resolution of integration issues. To facilitate CI, prioritize trunk-based development with short-lived branches and pull requests. Employing a CI server and automated tests is crucial to ensure that the code remains functional at all times.

Key Principles of CI:

- **Trunk-based development:** Work on a single main branch with short-lived feature branches.
- **Self-testing build:** Automated tests are executed after each commit.

- **CI server:** Automates the build and test process.

Managing Merge Conflicts

Merge conflicts are inevitable when multiple developers work concurrently. Short-lived branches minimize the likelihood of conflicts, and frequent merges simplify their resolution.

Preventing Errors with Self-Testing Builds

A self-testing build, triggered after each commit, runs automated tests. This ensures that only code that passes the tests is merged into the main branch. If an error does slip through, the typical solution is to revert the offending commit to quickly restore a working state.

Handling Significant Changes

- **Branch by Abstraction:** For major refactorings, introduce an abstraction layer to make changes gradually without breaking existing code. Migrate modules to the abstraction incrementally, then modify the underlying implementation of the abstraction without impacting the modules.
- **Feature Toggles:** For new features, wrap the code in conditional statements controlled by feature toggles. This allows you to merge incomplete code into `main` without exposing the feature to users. Feature toggles are disabled by default.

Example: Running Automated Tests for the Application with GitHub Actions

Objective: Configure a GitHub Actions workflow to automatically run tests of the Node.js application after each `push`.

Steps:

1. **Prepare the project:**
    - Navigate to your main project directory and create a new directory

```
$ cd ~/devops_base
$ git checkout main
$ git pull origin main
```

    - Create a new `td5` folder and copy the `sample-app` folder from `td4` into it:

```
$ mkdir -p td5
$ cp -r td4/scripts/sample-app td5/scripts/sample-app
```

2. **Create the GitHub Actions workflow:**

   - Create the `.github/workflows` directory at the root of your repository.

     ```
     $ mkdir -p .github/workflows
     $ cd .github/workflows
     ```

   - Create the file `.github/workflows/app-tests.yml` with the following content:

     ```yaml
     # .github/workflows/app-tests.yml (Example 5-2)
     name: Sample App Tests

     on: push

     jobs:
       sample_app_tests:
         name: "Run Tests Using Jest"
         runs-on: ubuntu-latest
         steps:
           - uses: actions/checkout@v3
           - name: Install dependencies
             working-directory: td5/scripts/sample-app
             run: npm install
           - name: Run tests
             working-directory: td5/scripts/sample-app
             run: npm test
     ```

     Explanation of the workflow:

   - `name` : Name of the workflow.
   - `on: push` : Triggers the workflow on every push to any branch.
   - `jobs` : Defines the tasks to run.
     - `sample_app_tests` : Name of the job.
       - `name` : User-friendly name for the job.
       - `runs-on: ubuntu-latest` : Specifies the runner environment (Ubuntu).
       - `steps` : Sequence of steps to execute.

        - `uses: actions/checkout@v3` : Checks out the source code from the repository.
        - `name: Install dependencies` : Step named for dependency installation.
            - `working-directory` : Defines the working directory.
            - `run: npm install` : Executes the `npm install` command.
        - `name: Run tests` : Step named for running tests.
            - `working-directory` : Defines the working directory.
            - `run: npm test` : Executes the `npm test` command.

3. **Test the workflow:**

    - To try it out, first commit and push the sample app and workflow code to your repo:

      ```
      $ git add td5/scripts/sample-app .github/workflows/app-tests.yml
      $ git commit -m "Add sample-app and workflow"
      $ git push origin main
      ```

      Create a new branch:

      ```
      $ git checkout -b test-workflow
      ```

    - Introduce an intentional error in `td5/scripts/sample-app/app.js` (e.g., change the response text to "DevOps Labs!").

      ```
      // Example 5-3: Update the app response text (td5/scripts/sample-app/app.js)
      res.send('DevOps Labs!');
      ```

    - Commit and push the changes:

      ```
      $ git add td5/scripts/sample-app/app.js
      $ git commit -m "Introduce intentional error"
      $ git push origin test-workflow
      ```

    - Create a *pull request* (PR) on GitHub.

    - Observe the workflow execution in the "Actions" tab of the PR. It should fail.

    - Click on "Details" to see the cause of the failure.

    - Correct the test in `td5/scripts/sample-app/app.test.js` :

```
// Example 5-4: Update the test to expect the new response
(td5/scripts/sample-app/app.test.js)
expect(response.text).toBe('DevOps Labs!');
```

- Commit and push the fix.

```
$ git add td5/scripts/sample-app/app.test.js
$ git commit -m "Update response text in test"
$ git push origin test-workflow
```

- Observe that the workflow re-runs and succeeds.

1.4. Machine User Credentials and Automatically-Provisioned Credentials

To run infrastructure tests (OpenTofu) that deploy AWS resources, you need to configure authentication.

Never use a real user's credentials for automation. Instead, use:

- **Machine User Credentials:** A dedicated user account for automation with limited permissions. Drawbacks: manual credential management, long-lived credentials.
- **Automatically-Provisioned Credentials (e.g., OIDC):** Dynamically generated credentials, often with short lifespans. More secure and recommended when possible.

1.5. Example: Configure OIDC with AWS and GitHub Actions

Objective: Set up OIDC to allow GitHub Actions to authenticate to AWS securely.

Steps:

1. **Create an OIDC provider in AWS:**
   - Use the provided Terraform module `github-aws-oidc` that lives in the `td5/scripts/tofu/modules/github-aws-oidc` folder to configure GitHub as an OIDC provider in your AWS account.
   - Switch back to the main branch, pull down the latest changes (i.e., the PR you just merged), and create a new branch called opentofu-tests:

```
$ git checkout main
$ git pull origin main
$ git checkout -b opentofu-tests
```

- Create a root module in `td5/scripts/tofu/live/ci-cd-permissions`.

```
$ mkdir -p td5/scripts/tofu/live/ci-cd-permissions
$ cd td5/scripts/tofu/live/ci-cd-permissions
```

- Configure the `github-aws-oidc` module in `td5/scripts/tofu/live/ci-cd-permissions/main.tf`:

(Example 5-5, modified)

```
provider "aws" {
  region = "us-east-2" # Replace with your desired region
}

module "oidc_provider" {
  source       = "github.com/your_github_name/devops-
base//td5/scripts/tofu/modules/github-aws-oidc"
  provider_url = "https://token.actions.githubusercontent.com"
}
```

2. **Create IAM roles for testing and deployment:**

- Use the provided Terraform module `gh-actions-iam-roles` to create IAM roles with specific permissions for testing and deploying your infrastructure.

- Configure the `iam_roles` module in `td5/scripts/tofu/live/ci-cd-permissions/main.tf`:

(Example 5-6, modified)

```
module "oidc_provider" {
  # ... (other params omitted) ...
}

module "iam_roles" {
  source = "github.com/your_github_name/devops-
base//td5/scripts/tofu/modules/gh-actions-iam-roles"

  name              = "lambda-sample"
  oidc_provider_arn = module.oidc_provider.oidc_provider_arn

  enable_iam_role_for_testing = true
  enable_iam_role_for_plan  = true  # Add for plan role
  enable_iam_role_for_apply = true # Add for apply role

  # TODO: Replace with your own GitHub repo name!
  github_repo      = "YOUR_GITHUB_USERNAME/YOUR_GITHUB_REPO" # ex: "bta-
devops/cloud-native-devops-kubernetes-2e"
  lambda_base_name = "lambda-sample"
  tofu_state_bucket = "YOUR_S3_BUCKET_NAME" # Replace with your bucket name
  tofu_state_dynamodb_table = "YOUR_DYNAMODB_TABLE_NAME" # Replace with your
table name
}
```

Explanation:

- `name`: Base name for the IAM roles and other resources created by this module.
- `oidc_provider_arn`: ARN of the OIDC provider you created earlier.
- `enable_iam_role_for_testing`, `enable_iam_role_for_plan`, `enable_iam_role_for_apply`: Set to `true` to create IAM roles for testing, planning, and applying, respectively.
- `github_repo`: **Replace with your own GitHub repository name (YOUR_GITHUB_USERNAME/YOUR_REPO_NAME).**
- `lambda_base_name`: Must match the `name` used in the `lambda-sample` module.
- `tofu_state_bucket` and `tofu_state_dynamodb_table`: **Replace with the names of your S3 bucket and DynamoDB table (which you'll configure later).**

- Create `td5/scripts/tofu/live/ci-cd-permissions/outputs.tf` to output the ARNs of the created IAM roles:

  (Example 5-7, modified)

```
output "lambda_test_role_arn" {
description = "The ARN of the IAM role for testing"
value       = module.iam_roles.lambda_test_role_arn
}
```

3. **Deploy the infrastructure:**

```
cd td5/scripts/tofu/live/ci-cd-permissions
tofu init
tofu apply
```

   - Note down the output values: `lambda_test_role_arn`, `lambda_deploy_plan_role_arn`, `lambda_deploy_apply_role_arn`.

1.6. Running Automated Infrastructure Tests

Objective: Configure a GitHub Actions workflow to run OpenTofu tests using OIDC authentication.

Steps:

1. **Copy infrastructure code:** Copy the `lambda-sample` and `test-endpoint` modules from `td4/scripts/tofu` to `td4/scripts/tofu`.

2. **Update `lambda-sample` module:**

   - Add `td5/scripts/tofu/live/lambda-sample/variables.tf` to allow configuring the base name for resources

     (Example 5-8)

     ```
     variable "name" {
       description = "The base name for the function and all other resources"
       type        = string
       default     = "lambda-sample"
     }
     ```

   - Modify `td5/scripts/tofu/live/main.tf` to use `var.name` instead of hard-coded values (Example 5-9).

     (Example 5-9, modified)

```
#
module "function" {
  #  ... (other params omitted) ...
  name = var.name
}

module "gateway" {
  #  ... (other params omitted) ...
  name = var.name
}
```

3. **Create the workflow file:** Create `.github/workflows/infra-tests.yml` with the following content:

```yaml
# name: Infrastructure Tests

on: push

jobs:
opentofu_test:
    name: "Run OpenTofu tests"
    runs-on: ubuntu-latest
    permissions:
    id-token: write
    contents: read
    steps:
    - uses: actions/checkout@v2

    - uses: aws-actions/configure-aws-credentials@v3
      with:
      # TODO: fill in your IAM role ARN!
      role-to-assume: arn:aws:iam::111111111111:role/lambda-sample-tests
      role-session-name: tests-${{ github.run_number }}-${{ github.actor }}
      aws-region: us-east-2

  - uses: opentofu/setup-opentofu@v1

  - name: Tofu Test
    env:
      TF_VAR_name: lambda-sample-${{ github.run_id }}
    working-directory: td5/scripts/tofu/live/lambda-sample
    run: |
      tofu init -backend=false -input=false
      tofu test -verbose
```

Explanations:

- `permissions`: Adds necessary permissions for OIDC (`id-token: write`) and writing to pull requests (`pull-requests: write`).
- `role-to-assume`: **Replace with the value of `lambda_test_role_arn` and store it in a GitHub secret named `TEST_ROLE_ARN`.**
- `uses: hashicorp/setup-terraform@v3`: **Corrected to use the official Terraform setup action.**
- `env: TF_VAR_name`: Defines an environment variable to ensure unique resource names for each test run.
- `terraform init`: Initializes the working directory with `-backend=false` to disable remote state for tests.
- `terraform test`: Executes the OpenTofu tests.

4. **Commit, push, and create a pull request:**

```
git add .
git commit -m "Add infrastructure tests workflow"
git push origin opentofu-tests
```

   Create a pull request on GitHub.

5. **Observe workflow execution:** Go to the "Actions" tab on GitHub and monitor the workflow.

Part 2: Continuous Delivery (CD)

Continuous Delivery (CD) automates the deployment process to enable fast, reliable and frequent deliveries.

Deployment strategies

Familiarise yourself with the different deployment strategies and their characteristics:

Strategy	User Experience	Stateless Applications	Support
Downtime Deployment	Service Interruption	Supported	Widely Supported
Progressive Deployment without Replacement	Alternation between versions	Supported	**Not Supported**
Progressive Deployment with Replacement	Alternating between versions	Supported	**Variable Support** (depends on disk movement)
Blue/Green Deployment	Instant Failover	Supported	**Not Supported**
Canary Deployment	Initial deployment to a single replica	Supported	Dependent on primary deployment policy
Deployment by Feature Toggles	Deploy code without enabling features	Supported	Depends on primary deployment strategy
Deployment by Promotion	Progressive deployment across environments	Supported	Dependent on the main deployment strategy

Recommendations:

- Unstated applications:** Blue/green deployment if possible, otherwise progressive deployment without replacement.
- Applications with state:** Progressive deployment with replacement.
- Complex data migrations:** Planned downtime can sometimes be simpler and less risky.
  Complementary strategies:
- Canary:** Reduces the impact of faulty deployments by initially deploying to a single instance.
- Feature Toggles:** Allows code to be deployed without activating new features, enabling progressive activation and rollbacks.
- Promotion:** Deploys code across multiple environments (dev, staging, prod) for more in-depth testing.

Deployment pipelines

A deployment pipeline automates the deployment stages, from commit to release. It is preferable to run the pipeline on a dedicated deployment server for consistency, repeatability and security.

Example: Automated deployment pipeline with GitHub Actions

Objective: Set up a deployment pipeline for the `lambda-sample` module which runs on *pull requests* (plan) and after they have been merged (apply). **Steps:**

- To use the state-bucket module, first check out the main branch of your own repo, and make sure you have the latest code:

```
$ cd ~/devops_base
$ git checkout main
$ git pull origin main
```

- Next, create a new folder called tofu-state to use as a root module:

```
$ mkdir -p td5/scripts/tofu/live/tofu-state
$ cd td5/scripts/tofu/live/tofu-state
```

1. **Configuration of the Remote Backend for OpenTofu State:**
    - Create an S3 bucket and a DynamoDB table to store the OpenTofu state remotely. You can use the `state-bucket` module provided in the code (`td5/scripts/tofu/modules/state-bucket`).

      Configure the `state-bucket` mdoule in `td5/scripts/tofu/live/tofu-state/main.tf`

      ```
      provider "aws" {
        region = "us-east-2" # Your AWS region
      }

      module "state" {
        source = "github.com/your_github_name/devops-
      base//td5/scripts/tofu/modules/state-bucket"
        name   = "YOUR-UNIQUE-BUCKET-NAME" # Replace with a unique name
      }
      ```

    - To create the S3 bucket and DynamoDB table, run init and apply as usual:

```
$ tofu init
$ tofu apply
```

- Create a `backend.tf` file in `td5/scripts/tofu/live/tofu-state/` (td5/scripts/tofu/live/tofu-state/backend.tf):

  (Example 5-12, to be adapted)

```
terraform {
  backend "s3" {
    bucket         = "YOUR_S3_BUCKET_NAME" # Replace
    key            = "td5/scripts/tofu/live/tofu-state"
    region         = "us-east-2" # Your AWS region
    encrypt        = true
    dynamodb_table = "YOUR_DYNAMODB_TABLE_NAME" # Replace
  }
}
```

- Run tofu init one more time, and you should see something like this:

```
$ tofu init
```

```
Initializing the backend...
Do you want to copy existing state to the new backend?
Pre-existing state was found while migrating the previous "local" backend
to the newly configured "s3" backend. No existing state was found in the
newly configured "s3" backend. Do you want to copy this state to the new
"s3" backend? Enter "yes" to copy and "no" to start with an empty state.

Enter a value:
```

  OpenTofu will automatically detect that you already have a state file locally and prompt you to copy it to the new S3 backend. If you type yes and hit ENTER, you should see the following:

```
Successfully configured the backend "s3"! OpenTofu will automatically
use this backend unless the backend configuration changes.
```

- Update the `lambda-sample` module module to use S3 as a backend (`td5/scripts/tofu/live/lambda-sample/backend.tf`).

(Example 5-13, to be adapted)

```
terraform {
  backend "s3" {
    bucket         = "YOUR_S3_BUCKET_NAME" # Same bucket as above
    key            = "td5/scripts/tofu/live/lambda-sample" # Unique key
    region         = "us-east-2" # Your AWS region
    encrypt        = true
    dynamodb_table = "YOUR_DYNAMODB_TABLE_NAME" # Same table as above
  }
}
```

- Run `tofu init` in the `tofu-state` and `lambda-sample` directories to initialize the remote backend.

2. **Creating IAM Roles for Deployment:**

- Modify the `ci-cd-permissions` module to enable the creation of IAM roles for `plan` and `apply` in `td5/scripts/tofu/live/ci-cd-permissions/main.tf`.

  (Example 5-14, modified)

```
module "iam_roles" {
  source = "github.com/your_github_name/devops-
base//td5/scripts/tofu/modules/gh-actions-iam-roles"

  name                       = "lambda-sample"
  oidc_provider_arn       = module.oidc_provider.oidc_provider_arn

  enable_iam_role_for_testing = true
  enable_iam_role_for_plan     = true
  enable_iam_role_for_apply    = true

  github_repo                 = "YOUR_USERNAME/YOUR_REPO" # Replace
  lambda_base_name            = "lambda-sample"
  tofu_state_bucket           = "YOUR_S3_BUCKET_NAME" # Replace
  tofu_state_dynamodb_table   = "YOUR_DYNAMODB_TABLE_NAME" # Replace
}
```

- Add `output` variables for the ARNs of the new roles in `td5/scripts/tofu/live/ci-cd-permissions/outputs.tf`.

  (Example 5-15, modified)

```
output "lambda_deploy_plan_role_arn" {
 description = "The ARN of the IAM role for plan"
 value       = module.iam_roles.lambda_deploy_plan_role_arn
}

output "lambda_deploy_apply_role_arn" {
 description = "The ARN of the IAM role for apply"
 value       = module.iam_roles.lambda_deploy_apply_role_arn
}
```

- Run `tofu apply` to create the IAM roles.
- **Note the output values** `lambda_deploy_plan_role_arn` and `lambda_deploy_apply_role_arn`.

3. **Creating GitHub Actions Workflows for Deployment:**
   - **The workflow to run** `.github/workflows/tofu-plan.yml/tofu-plan.yml` **(Example 5-16, to be adapted):**

```yaml
name: Tofu Plan

on:
    pull_request:
        branches: ["main"]
        paths: ["td5/scripts/tofu/live/lambda-sample/**"]

jobs:
    plan:
        name: "Tofu Plan"
        runs-on: ubuntu-latest
        permissions:
            pull-requests: write
            id-token: write
            contents: read
        steps:
            - uses: actions/checkout@v2

            - uses: aws-actions/configure-aws-credentials@v3
              with:
                  # TODO: fill in your IAM role ARN!
                  role-to-assume: arn:aws:iam::111111111111:role/lambda-
sample-plan
                  role-session-name: plan-${{ github.run_number }}-${{
github.actor }}
                  aws-region: us-east-2

            - uses: opentofu/setup-opentofu@v1

            - name: tofu plan
              id: plan
              working-directory: td5/scripts/tofu/live/lambda-sample
              run: |
                  tofu init -no-color -input=false
                  tofu plan -no-color -input=false -lock=false

            - uses: peter-evans/create-or-update-comment@v4
              if: always()
              env:
                  RESULT_EMOJI: ${{ steps.plan.outcome == 'success' && '✅'
|| '⚠' }}
              with:
                  issue-number: ${{ github.event.pull_request.number }}
                  body: |
                      ## ${{ env.RESULT_EMOJI }} `tofu plan` output
                      ```${{ steps.plan.outputs.stdout }}```
```

**Explanation:**
```

- This workflow is triggered on *pull requests* targeting the `main` branch and modifying files in `td5/scripts/tofu/live/lambda-sample/**`.
- It uses the `aws-actions/configure-aws-credentials@v3` action to authenticate with AWS via OIDC, assuming the IAM role specified by the GitHub secret `PLAN_ROLE_ARN` (**which you need to create and configure with the plan role ARN**).
- It runs `terraform init` and then `terraform plan` to generate the execution plan.
- It uses the `actions/github-script@v7` action to post the execution plan as a comment on the *pull request*. The code has been broken down to make it more readable.
- **Important:** The code has been corrected to use `terraform` instead of `tofu` because the `hashicorp/setup-terraform` action is being used.

- **The workflow to run `.github/workflows/tofu-apply.yml`

  (Example 5-17, to be adapted):

```yaml
name: Tofu Apply
on:
    push:
        branches: ["main"]
        paths: ["td5/scripts/tofu/live/lambda-sample/**"]
jobs:
    apply:
        name: "Tofu Apply"
        runs-on: ubuntu-latest
        permissions:
            pull-requests: write
            id-token: write
            contents: read
        steps:
            - name: Checkout repository
              uses: actions/checkout@v2

            - name: Configure AWS credentials
              uses: aws-actions/configure-aws-credentials@v3
              with:
                  role-to-assume: arn:aws:iam::111111111111:role/lambda-
sample-apply
                  role-session-name: apply-${{ github.run_number }}-${{
github.actor }}
                  aws-region: us-east-2

            - name: Setup OpenTofu
              uses: opentofu/setup-opentofu@v1

            - name: Tofu apply
              id: apply
              working-directory: td5/scripts/tofu/live/lambda-sample
              run: |
                  tofu init -no-color -input=false
                  tofu apply -no-color -input=false -lock-timeout=60m -auto-
approve

            - name: Find current PR
              uses: jwalton/gh-find-current-pr@master
              id: find_pr
              with:
                  state: all

            - name: Create or update comment
              uses: peter-evans/create-or-update-comment@v4
              if: steps.find_pr.outputs.number
              env:
                  RESULT_EMOJI: ${{ steps.apply.outcome == 'success' &&
'✅' || '⚠' }}
              with:
```

```
                    issue-number: ${{ steps.find_pr.outputs.number }}
                    body: |
                        ## ${{ env.RESULT_EMOJI }} `tofu apply` output
                        ```${{ steps.apply.outputs.stdout }}```
```

**Explanation:**

- This workflow is triggered by **pushes to the `main` branch** and if the modified files are in `td5/scripts/tofu/live/lambda-sample/**`.
- It uses the `aws-actions/configure-aws-credentials@v3` action to authenticate with AWS via OIDC, assuming the IAM role specified by the GitHub secret `APPLY_ROLE_ARN` (**which you need to create and configure with the apply role ARN**).
- It runs `terraform init` and then `terraform apply` to apply the changes.
- It uses the `actions/github-script@v7` action to post the output of the `terraform apply` command as a comment on the associated pull request.

4. **Test the deployment pipeline:**

   - Commit these new workflow files directly to the main branch and then push them to GitHub:

```
$ git add .github/workflows
$ git commit -m "Add plan and apply workflows"
$ git push origin main
```

   - Create a new branch:

```
git checkout -b deployment-pipeline-test
```

   - Make a change to the `lambda-sample` module, such as changing the text it returns, as shown in **(Example 5-18)** by update the app response text in `td5/scripts/tofu/live/lambda-sample/src/index.js` :

```
exports.handler = (event, context, callback) => {
  callback(null, {statusCode: 200, body: "DevOps Labs!"});
};
```

   - Update the response text in the tests `td5/scripts/tofu/live/lambda-sample/deploy.tftest.hcl` **(Example 5-19)**:
```

```
assert {
  condition     = data.http.test_endpoint.response_body == "Fundamentals of
DevOps!"
  error_message = "Unexpected body: ${data.http.test_endpoint.response_body}"
}
```

- Commit and push the changes:

```
git add td5/scripts/tofu/live/lambda-sample/src/index.js
td5/scripts/tofu/live/lambda-sample/deploy.tftest.hcl
git commit -m "Update response text"
git push origin deployment-pipeline-test
```

- Create a pull request on GitHub.
- Observe the execution of the `app-tests`, `infra-tests`, and `tofu-plan` workflows.
- Verify the comment with the `tofu plan` output.
- Merge the pull request.
- Observe the execution of the `tofu-apply` workflow and the publication of the output in a comment.

Important:

- Remember to replace placeholder values (like S3 bucket names, DynamoDB table names, and IAM role ARNs) with your own values.
- **Store the IAM role ARNs in GitHub secrets named `PLAN_ROLE_ARN` and `APPLY_ROLE_ARN`.**

Exercise [Practice]

Here are a few exercises you can try at home to go deeper:

- Update the pipeline to automatically detect changes in an any folder with OpenTofu code (rather than only the lambda-sample folder), and to automatically run plan and apply in each one (see changed-files action).
- If a PR updates multiple folders, use a matrix strategy to run plan and apply across multiple folders concurrently.

Conclusion

This lab has covered the essentials of setting up CI/CD pipelines with GitHub Actions, OpenTofu, and Kubernetes. You've learned how to automate tests, manage infrastructure deployments, and implement key DevOps principles. Remember to adapt the code examples to your specific needs and explore the documentation of the tools used for a deeper understanding.