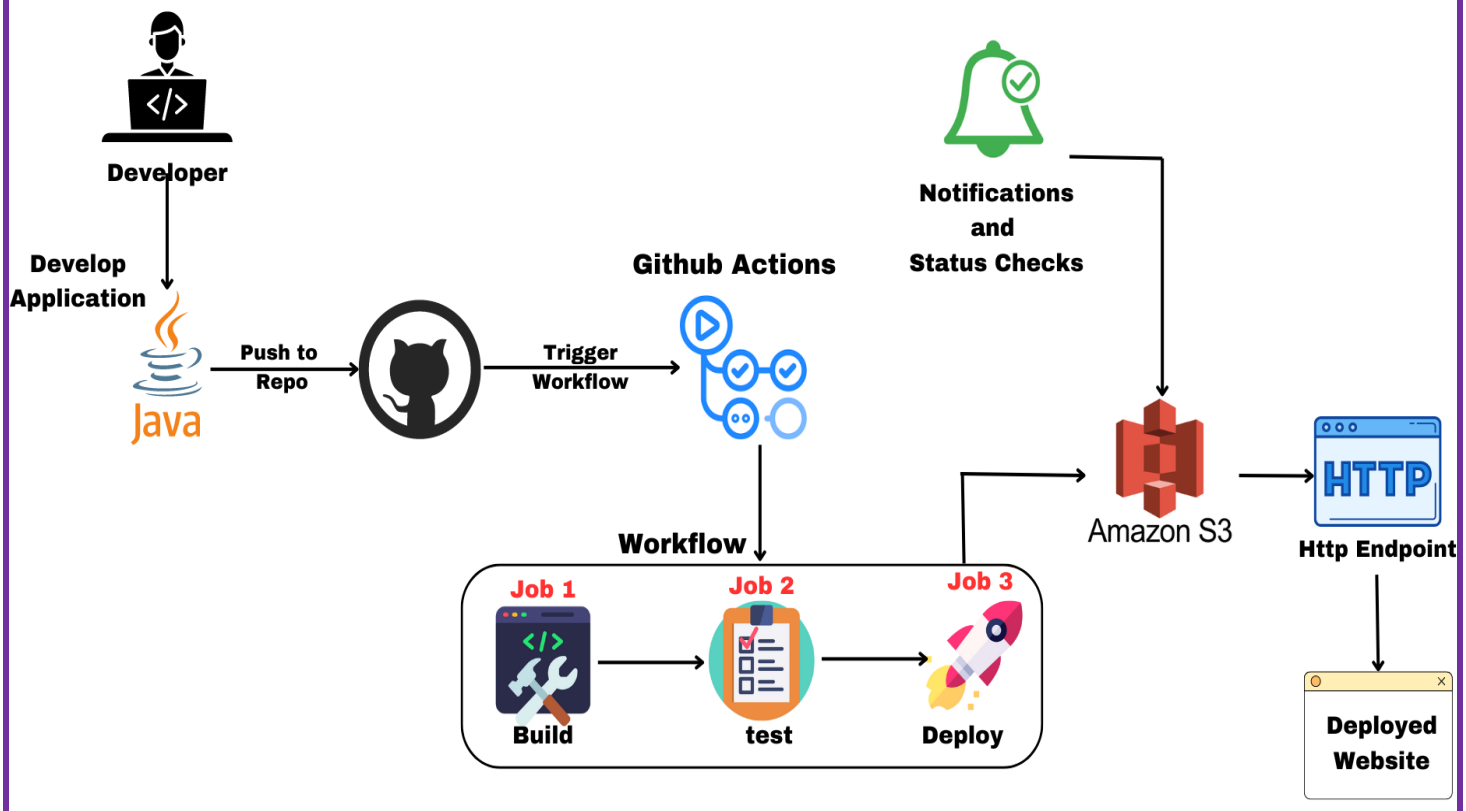
DevOps Shack

How Deploy an Application to AWS Using GitHub Actions

This document provides a comprehensive guide to automating the deployment of applications to AWS using GitHub Actions. GitHub Actions is a powerful CI/CD tool that enables developers to streamline their workflows directly within their GitHub repositories. By breaking down the process into distinct jobs—Build, Test, and Deploy—this guide ensures that your application is reliably built, tested, and deployed to AWS services like S3 or EC2. Whether you're deploying a static website or a complex application.



Overview

GitHub Actions allows you to automate the deployment of your application to AWS. In this guide, we will:

- Set up an AWS environment.
- Configure AWS credentials in GitHub Actions.
- Create a GitHub Actions workflow for deployment.
- Deploy the application to AWS.

Pre-requisites

- An AWS account with IAM (Identity and Access Management) permissions.
- An S3 bucket (for static website hosting) or an EC2 instance (for server-based applications).
- AWS CLI configured on your local machine (optional but helpful for testing).
- A GitHub repository for your application.

Workflow

1. Code Push to GitHub

Trigger: The entire workflow begins when code is pushed to the GitHub repository, typically to a specific branch like main. This push event triggers the GitHub Actions workflow.

Importance: It ensures that any new changes to the codebase automatically start the CI/CD process, maintaining continuous integration and delivery.

2. Job 1 - Build

Checkout Code: The workflow first checks out the latest version of the code from the repository. This ensures that the subsequent steps work with the most up-to-date code.

Install Dependencies: The necessary project dependencies are installed (e.g., via npm install for a Node.js project). This step prepares the environment for building the project.

Build Project: The project is built, meaning the code is compiled or processed as required. For example, in a web project, assets might be bundled, or in a compiled language, the source code might be compiled into binaries.

Outcome: A successful build produces the necessary artifacts (e.g., compiled code, static files) that are ready for testing and deployment.

3. Job 2 - Test

Checkout Code: Once again, the code is checked out to ensure the test job operates on the correct version of the code.

Run Tests: The workflow runs the test suite, which might include unit tests, integration tests, or other automated checks. This step ensures that the code behaves as expected and that no new changes have introduced bugs.

Dependency on Build: This job depends on the successful completion of the build job (needs: build). If the build fails, the test job will not run, preventing wasted effort on broken code.

4. Job 3 - Deploy

Install AWS CLI: The AWS Command Line Interface (CLI) is installed on the runner to enable interaction with AWS services. This step is crucial for the deployment process.

Configure AWS Credentials: The workflow configures AWS credentials using secrets stored in the GitHub repository. This setup allows the runner to authenticate with AWS securely.

Deploy to S3 or EC2: The final deployment step pushes the built and tested application to AWS. Depending on the setup:

S3: The workflow syncs the built files to an S3 bucket, ideal for static websites or assets.

EC2: The workflow uses scp or similar tools to transfer files to an EC2 instance, suitable for server-based applications.

Dependency on Test: The deploy job depends on the successful completion of the test job (needs: test). This dependency ensures that only tested code is deployed to production, maintaining application stability.

Flow Summary

The workflow starts with a push to the GitHub repository, triggering a series of jobs: Build, Test, and Deploy. Each job has a clear purpose:

Build: Prepares the code for deployment.

Test: Ensures the code is reliable and bug-free.

Deploy: Deploys the verified code to AWS.

This structured approach ensures that only thoroughly tested and properly built code is deployed, maintaining high standards of quality and reliability in the production environment. The dependencies between jobs ensure that if any step fails, the process halts, preventing the deployment of flawed code.

Step 1: Set Up AWS Environment

Depending on your application type, you need to set up the appropriate AWS services. Below are the common scenarios:

A. Static Website Deployment to S3

- Create an S3 Bucket:
- Log in to your AWS Console.
- Navigate to the S3 service.
- Click "Create bucket" and provide a unique name.
- Configure the bucket for public access (if necessary).
- Enable static website hosting under the "Properties" tab.

Set Up Bucket Policy:

- Navigate to the "Permissions" tab of your bucket.
- Add a bucket policy to allow public access (if needed) for static website hosting.

B. Application Deployment to EC2

Launch an EC2 Instance:

- Log in to your AWS Console.
- Navigate to the EC2 service.
- Click "Launch Instance" and follow the setup wizard.
- Choose an Amazon Machine Image (AMI) (e.g., Ubuntu).
- Configure security groups to allow necessary inbound traffic (e.g., HTTP, SSH).

Install Necessary Software:

SSH into your EC2 instance and install any required software (e.g., Node.js, Nginx, Docker).

Step 2: Configure AWS Credentials in GitHub Actions

To allow GitHub Actions to interact with your AWS environment, you'll need to set up AWS credentials in your GitHub repository.

Create an IAM User in AWS:

- Go to the IAM service in the AWS Console.
- Create a new user with programmatic access.
- Attach the necessary policies (e.g., AmazonS3FullAccess for S3, AmazonEC2FullAccess for EC2).

Store AWS Credentials in GitHub Secrets:

In your GitHub repository, navigate to Settings > Secrets and variables > Actions.

Create two new secrets:

AWS_ACCESS_KEY_ID: Your IAM user's access key.

AWS_SECRET_ACCESS_KEY: Your IAM user's secret key.

Step 3: Create a GitHub Actions Workflow

Create a YAML file in your repository to define the GitHub Actions workflow.

Workflow for S3 Bucket Deployment

```
name: CI/CD Pipeline to AWS S3

on:
  push:
    branches:
      - main
```

```
jobs:
  build:
    name: Job 1 - Build
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Install dependencies
        run: npm install

      - name: Build project
        run: npm run build

  test:
    name: Job 2 - Test
    runs-on: ubuntu-latest
    needs: build
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Run tests
        run: npm test

  deploy:
```

```
  name: Job 3 - Deploy to AWS S3
  runs-on: ubuntu-latest
  needs: test
  steps:
   - name: Checkout code
     uses: actions/checkout@v3

   - name: Install AWS CLI
     run: |
       sudo apt-get install awscli -y

   - name: Configure AWS credentials
     run: |
       aws configure set aws_access_key_id ${{ secrets.AWS_ACCESS_KEY_ID }}
       aws configure set aws_secret_access_key ${{
secrets.AWS_SECRET_ACCESS_KEY }}
       aws configure set default.region us-east-1

   - name: Deploy to S3
     run: |
       aws s3 sync ./build s3://your-bucket-name --delete
```

Workflow for EC2 Deployment

```
name: CI/CD Pipeline to AWS EC2

on:
  push:
    branches:
      - main

jobs:
  build:
    name: Job 1 - Build
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Install dependencies
        run: npm install

      - name: Build project
        run: npm run build

  test:
    name: Job 2 - Test
    runs-on: ubuntu-latest
    needs: build
```

```
  steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Run tests
      run: npm test

  deploy:
    name: Job 3 - Deploy to AWS EC2
    runs-on: ubuntu-latest
    needs: test
    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Install AWS CLI
      run: |
        sudo apt-get install awscli -y

    - name: Configure AWS credentials
      run: |
        aws configure set aws_access_key_id ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws configure set aws_secret_access_key ${{
secrets.AWS_SECRET_ACCESS_KEY }}
        aws configure set default.region us-east-1

    - name: Deploy to EC2
```

```
    run: |
      scp -i path/to/your-key.pem -r * ec2-user@your-ec2-ip:/var/www/html/
```

Explanation of the Workflow

Job 1 - Build:

Description: This job checks out the code, installs dependencies, and builds the project. This ensures that the code is compiled or processed as needed before further steps.

Steps:

Checkout code: Retrieves the code from the repository.

Install dependencies: Installs any required project dependencies (e.g., npm install).

Build project: Executes the build command (e.g., npm run build).

Job 2 - Test:

Description: This job runs after the build job is successful. It ensures that the code passes all tests before deployment.

Steps:

Checkout code: Retrieves the code from the repository.

Run tests: Executes the test suite (e.g., npm test).

Job 3 - Deploy:

Description: This job runs only after the test job is successful. It handles the deployment of the built application to AWS (either S3 or EC2).

Steps for S3 Deployment:

Install AWS CLI: Installs AWS CLI to interact with AWS services.

Configure AWS credentials: Configures AWS CLI with your credentials.

Deploy to S3: Syncs the built files to the specified S3 bucket.

Steps for EC2 Deployment:

Install AWS CLI: Installs AWS CLI.

Configure AWS credentials: Configures AWS CLI.

Deploy to EC2: Uses scp to securely copy the files to the EC2 instance.

Conclusion

By following this guide, you can successfully automate the deployment of your application to AWS using GitHub Actions. Whether you're deploying a static website to S3 or a more complex application to EC2, GitHub Actions provides a flexible, scalable solution for CI/CD pipelines.