CI/CD Pipeline for Dockerized Applications on AWS

1. Introduction:

Project Title: CI/CD Pipeline for Dockerized Applications on AWS

Objective: To establish a CI/CD pipeline that automates the building, testing, and deployment of Dockerized applications using AWS services, ensuring continuous integration and delivery for rapid, reliable updates.

2. Project Scope:

The scope of this project includes:

- Managing source control using AWS CodeCommit.
- Automating continuous integration and builds with AWS CodeBuild.
- Deploying applications using AWS CodePipeline and CodeDeploy.
- Storing Docker images in Amazon ECR.
- Orchestrating and deploying containers using Amazon ECS.
- Monitoring and logging using Amazon CloudWatch.

3. Architecture:

Source Control: AWS CodeCommit.
CI/CD Services: AWS CodePipeline, CodeBuild, CodeDeploy.
Container Registry: Amazon ECR.
Container Orchestration: Amazon ECS.
Monitoring and Logging: Amazon CloudWatch.

4. Steps and Configuration:

Step 1: Set Up Development Environment:

1. Install Docker: Follow the Docker Installation Guide to install Docker on your local machine.
2. Install AWS CLI: Follow the AWS CLI Installation Guide.
3. Install Git: Follow the Git Installation Guide.

Step 2: Create Dockerized Application:

1. **Write Dockerfile:** Create a Dockerfile in the root of your application directory.

   ```
   FROM node:14
   WORKDIR /app
   COPY package*.json ./
   RUN npm install
   COPY . .
   EXPOSE 3000
   CMD ["node", "index.js"]
   ```

2. Build Docker Image:

   ```
   docker build -t my-app:v1 .
   ```

Step 3: Push Docker Image to Amazon ECR:

1. **Create an ECR Repository**:

   ```
   aws ecr create-repository --repository-name my-app
   ```

2. **Authenticate Docker to ECR:**

   ```
   aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <your_account_id>.dkr.ecr.us-east-1.amazonaws.com
   ```

3. **Tag and Push the Docker Image:**

   - docker tag my-app:v1 <your_account_id>.dkr.ecr.us-east-1.amazonaws.com/my-app:v1
   - docker push <your_account_id>.dkr.ecr.us-east-1.amazonaws.com/my-app:v1

Step 4: Set Up Source Control with AWS CodeCommit:

1. **Create a CodeCommit Repository:**

    - Navigate to CodeCommit and create a new repository named my-app-repo.

2. **Push Your Application Code to CodeCommit:**

```
git init
git add .
git commit -m "Initial commit"
git remote add origin https://git-codecommit.us-east-1.amazonaws.com/v1/repos/my-app-repo
git push -u origin master
```

Step 5: Configure AWS CodePipeline:

1. **Create a Pipeline in CodePipeline:**

    - Navigate to CodePipeline and create a new pipeline named my-app-pipeline.
    - Choose CodeCommit as the source provider and select your repository and branch.

2. **Add a Build Stage Using CodeBuild:**

    - Create a new CodeBuild project.
    - Configure the build environment to use Docker.
    - Add the following buildspec.yml file to your repository:

```
version: 0.2
phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <your_account_id>.dkr.ecr.us-east-1.amazonaws.com
  build:
    commands:
      - echo Build started on `date`
```

```
      - echo Building the Docker image...
      - docker build -t my-app .
      - docker tag my-app:latest <your_account_id>.dkr.ecr.us-east-
1.amazonaws.com/my-app:latest
  post_build:
    commands:
      - echo Pushing the Docker image...
      - docker push <your_account_id>.dkr.ecr.us-east-
1.amazonaws.com/my-app:latest
      - printf '[{"name":"my-app","imageUri":"%s"}]'
<your_account_id>.dkr.ecr.us-east-1.amazonaws.com/my-app:latest >
imagedefinitions.json
artifacts:
  files: imagedefinitions.json
```

3. **Add a Deploy Stage Using CodeDeploy:**

   - Create a new CodeDeploy application.
   - Define a deployment group and configure it to use Amazon ECS.
   - Update your pipeline to include the deploy stage using the CodeDeploy application.

Step 6: Deploy to Amazon ECS:

1. **Create an ECS Cluster**:

   - Navigate to ECS and create a new cluster.

2. **Create a Task Definition:**

   - Define a new task with the following JSON configuration:

```
{
  "family": "my-app-task",
  "containerDefinitions": [
    {
      "name": "my-app",
      "image": "<your_account_id>.dkr.ecr.us-east-1.amazonaws.com/my-
app:latest",
      "essential": true,
      "memory": 512,
      "cpu": 256,
      "portMappings": [
```

```
        {
          "containerPort": 3000,
          "hostPort": 80
        }
      ]
    }
  ]
}
```

3. **Create a Service**:

    - Create a new service in your ECS cluster to run and manage your task definition.

Step 7: Monitor and Optimize:

1. **Set Up CloudWatch:**

    - Use Amazon CloudWatch to monitor metrics for your ECS services, tasks, and containers.
    - Set up alarms to notify you of any issues.

2. **Set Up Logging**:

    - Configure your ECS tasks to send logs to CloudWatch Logs for easier monitoring and debugging.

Website Application Details:

The application used in this project is a simple Node.js web application. Below are the details of the application:

1. **Application Code**:

```
const express = require('express');
const app = express();
const port = 3000;
app.get('/', (req, res) => {
  res.send('Hello World!');
});
app.listen(port, () => {
  console.log(`App running at http://localhost:${port}`);
});
```

2. **Package Dependencies:**

   - The application requires Express.js, which is specified in the package.json file.

```
{
  "name": "my-app",
  "version": "1.0.0",
  "description": "A simple Node.js web application",
  "main": "index.js",
  "scripts": {
    "start": "node index.js"
  },
  "dependencies": {
    "express": "^4.17.1"
  },
  "author": "",
  "license": "ISC"
}
```

6. Conclusion

This project successfully established a CI/CD pipeline to automate the building, testing, and deployment of Dockerized applications using AWS services. The implementation ensures continuous integration and delivery, enhancing the efficiency and reliability of application deployments.

7. Future Enhancements

Potential improvements for this project could include:

- Implementing auto-scaling for ECS services based on load.
- Adding automated testing stages in CodeBuild.
- Integrating with other AWS services for enhanced monitoring and security.