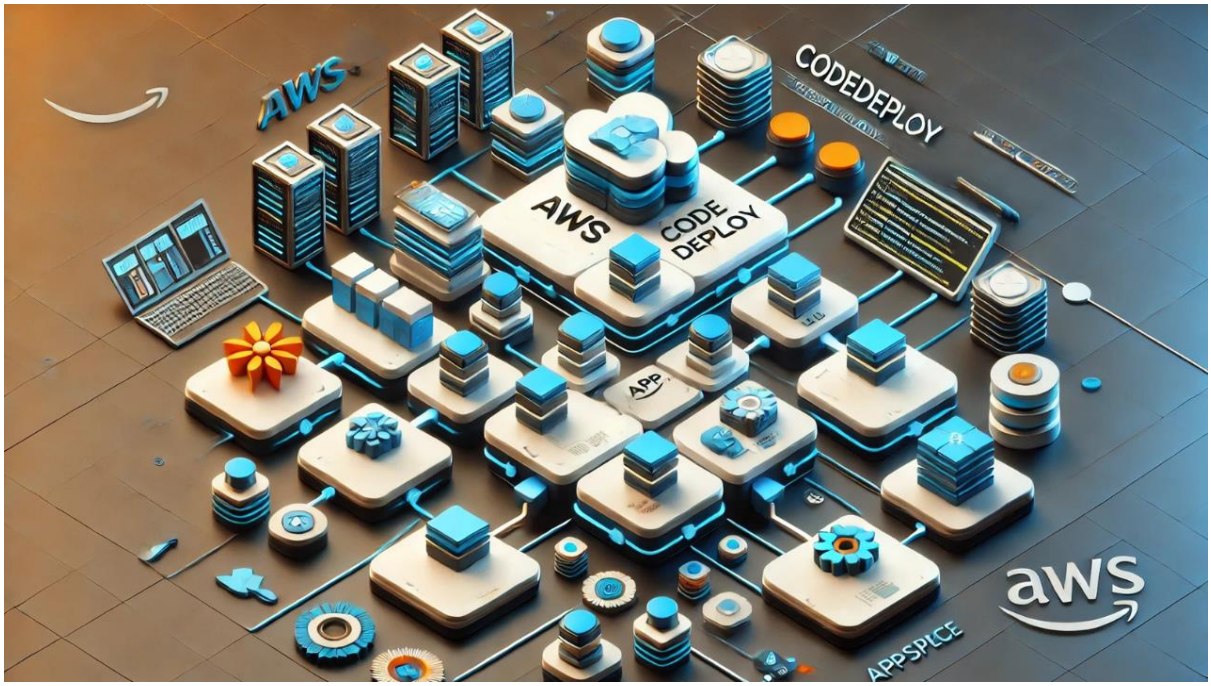
DEVOPS SHACK

AWS Code Deploy Beginners Guide

AWS CodeDeploy is a fully managed deployment service that automates application deployments to Amazon EC2 instances, on-premises servers, serverless Lambda functions, and ECS (Amazon Elastic Container Service). Below is an in-depth guide to AWS CodeDeploy, including the primary concepts, deployment types, and example configurations.



Key Components of AWS CodeDeploy

1. **Application**: The CodeDeploy application represents the set of resources where deployment happens, like EC2 instances or Lambda functions.
2. **Deployment Group**: A set of instances or Lambda functions targeted for deployment. It can also specify deployment configuration, auto-scaling groups, and alarms.
3. **AppSpec File**: A YAML or JSON file defining the deployment actions for CodeDeploy to follow, such as scripts to run during various stages of the deployment.
4. **Deployment Configuration**: This specifies how CodeDeploy rolls out your application. Predefined configurations include:

- CodeDeployDefault.OneAtATime: Deploy to one instance at a time.
- CodeDeployDefault.HalfAtATime: Deploy to half of the instances at a time.
- CodeDeployDefault.AllAtOnce: Deploy to all instances simultaneously.

CodeDeploy Deployment Types

1. **In-Place Deployment**: CodeDeploy stops the application on each instance, deploys the latest version, and restarts it. It's ideal for servers but requires temporary downtime.
2. **Blue/Green Deployment**: CodeDeploy provisions new instances for the deployment (green environment) and switches traffic from the old (blue) environment to the new environment after testing. This method minimizes downtime and is suited for critical applications.

AppSpec File for AWS CodeDeploy

The AppSpec file is central to CodeDeploy and varies based on the deployment type (EC2/on-premises or Lambda).

Example AppSpec File for EC2/On-Premises

Below is a basic example of an AppSpec file (appspec.yml) for an EC2 or on-premises deployment:

```
version: 0.0
os: linux
files:
  - source: /app
    destination: /var/www/html

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

  AfterInstall:
    - location: scripts/start_server.sh
      timeout: 300
      runas: root

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

  ValidateService:
    - location: scripts/validate.sh
      timeout: 300
      runas: root
```

Explanation:

- **files**: Specifies the files and directories to copy to the deployment instance.
- **hooks**: Defines lifecycle events, each executing a specified script. Common events:
  - **BeforeInstall**: Runs before the application is installed.
  - **AfterInstall**: Runs after the application is installed but before it starts.
  - **ApplicationStart**: Runs once the application is started.
  - **ValidateService**: Runs to validate the service after deployment is complete.

AppSpec File for Lambda

For Lambda deployments, the AppSpec file is simpler:

```
version: 0.0
Resources:
 - myLambdaFunction:
     Type: AWS::Lambda::Function
     Properties:
       Name: my-function-name
       Alias: prod
```

Explanation:

- **Resources**: Lists Lambda functions and aliases to be updated.
- **Alias**: Specifies the Lambda alias to update after deployment.

Example: Creating a CodeDeploy Application for EC2

1. **Step 1: Create an Application**

```
aws deploy create-application --application-name MyApp
```

2. **Step 2: Create a Deployment Group**

```
aws deploy create-deployment-group --application-name MyApp \
  --deployment-group-name MyDeploymentGroup \
  --deployment-config-name CodeDeployDefault.OneAtATime \
  --ec2-tag-filters Key=Name,Value=MyAppInstance,Type=KEY_AND_VALUE \
  --service-role-arn arn:aws:iam::123456789012:role/CodeDeployServiceRole
```

3. **Step 3: Deploy an Application**

```
aws deploy create-deployment --application-name MyApp \
  --deployment-group-name MyDeploymentGroup \
  --revision file://appspec.yml \
  --deployment-config-name CodeDeployDefault.OneAtATime
```

Example: Using CodeDeploy with Blue/Green Deployment on ECS

1. **Define Your AppSpec File for ECS**

```
version: 0.0
Resources:
 - TargetService:
     Type: AWS::ECS::Service
     Properties:
       TaskDefinition: "my-task:1"
       LoadBalancerInfo:
         ContainerName: "my-container"
         ContainerPort: 80
```

2. **Create a Deployment Group with Blue/Green Configuration**

```
aws deploy create-deployment-group --application-name MyApp \
  --deployment-group-name MyDeploymentGroup \
  --deployment-config-name CodeDeployDefault.ECSAllAtOnce \
  --service-role-arn arn:aws:iam::123456789012:role/CodeDeployServiceRole \
  --blue-green-deployment-configuration '{
     "terminateBlueInstancesOnDeploymentSuccess": {
       "action": "TERMINATE",
       "terminationWaitTimeInMinutes": 5
     },
     "deploymentReadyOption": {
       "actionOnTimeout": "CONTINUE_DEPLOYMENT",
       "waitTimeInMinutes": 5
     }
  }'
```

3. **Initiate the Deployment**

```
aws deploy create-deployment --application-name MyApp \
  --deployment-group-name MyDeploymentGroup \
  --revision revisionType=AppSpecContent,appSpecContent='file://appspec.yaml'
```

Monitoring and Managing Deployments

- **AWS Console**: CodeDeploy provides a console to visualize and monitor your deployments, check instance statuses, and handle deployment rollbacks.
- **CLI Commands**:
    - Check Deployment Status:

```
aws deploy get-deployment --deployment-id <deployment-id>
```

- **Rollbacks**: CodeDeploy can be configured to automatically roll back on failure or be manually triggered in the console or CLI.

Best Practices for CodeDeploy

1. **Automate Rollbacks**: Always enable rollback options for failed deployments to minimize downtime.
2. **Use Blue/Green for Zero Downtime**: Especially for production workloads, blue/green deployments help maintain high availability.
3. **Optimize AppSpec Files**: Define clear lifecycle event hooks and validate each deployment stage with scripts.
4. **Integrate with CI/CD Pipelines**: CodeDeploy integrates seamlessly with AWS CodePipeline, enabling end-to-end automation of your CI/CD workflow.
5. **Security Considerations**: Follow IAM best practices, ensuring the least privilege for all CodeDeploy roles and policies.

AWS CodeDeploy simplifies deployments, helping manage complex workflows and allowing quick and reliable updates across instances, serverless functions, or containerized services. With these examples and practices, you're equipped to get started with CodeDeploy and scale your deployment capabilities efficiently.