**AWS CODE DEPLOY**

**Overview**

CodeDeploy is a deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.

There are several ways to handle deployments using open source tools [Ansible, Tterraform, chef, Puppet, etc..]

We can use managed service AWS CodeDeploy

**Benefits:**

* Server, serverless, and container applications
* Automated deployments
* Minimize downtime
* If your application uses the EC2/On-Premises compute platform, CodeDeploy helps maximize your application availability. During an in-place deployment, CodeDeploy performs a rolling update across Amazon EC2 instances. You can specify the number of instances to be taken offline at a time for updates. During a blue/green deployment, the latest application revision is installed on replacement instances. Traffic is rerouted to these instances when you choose, either immediately or as soon as you are done testing the new environment. For both deployment types, CodeDeploy tracks application health according to rules you configure.
* Stop and roll back. You can automatically or manually stop and roll back deployments if there are errors.
* Concurrent deployments. If you have more than one application that uses the EC2/On-Premises compute platform, CodeDeploy can deploy them concurrently to the same set of instances.

**Steps**:

* Each EC2 instance Machine [or on primise machine] must be running the **CodeDeploy Agent**.
* The agent is continuously polling AWS codeDeploy for work to do.
* CodeDeploy sends **appspec.yml file** to the instances.
* Application is pulled from github or s3.
* Ec2 will run the deployment instructions.
* CodeDeploy Agent report of success / failure of deployment on the instance.
* Ec2 Instance are grouped by deployment group [dev/qa/prod]
* Lots of flexibility to define any kind of deployment
* CodeDeploy can be chanined into CodePpipeline and use artifacts from there
* CodeDeploy can re-use existing setup tools work with any application, auto scalling integration.
* Note: Blue / green only works with EC2 instances.
* Support for AWS lambda , EC2 and ECS && does not provision resources.

1.Push

Source code +

Appspec.yml file

Github/S3

4.Download code + appspec.yml file

2.Trigger Deployment

EC2 CLUSTER +

agent

Code Deploy

3.Poll

Here's how it works:

* First, you create deployable content on your local development machine or similar environment, and then you add an *application specification file* (AppSpec file). The AppSpec file is unique to CodeDeploy. It defines the deployment actions you want CodeDeploy to execute. You bundle your deployable content and the AppSpec file into an archive file, and then upload it to an Amazon S3 bucket or a GitHub repository. This archive file is called an *application revision* (or simply a *revision*).
* Next, you provide CodeDeploy with information about your deployment, such as which Amazon S3 bucket or GitHub repository to pull the revision from and to which set of Amazon EC2 instances to deploy its contents. CodeDeploy calls a set of Amazon EC2 instances a *deployment group*. A deployment group contains individually tagged Amazon EC2 instances, Amazon EC2 instances in Amazon EC2 Auto Scaling groups, or both.
* Each time you successfully upload a new application revision that you want to deploy to the deployment group, that bundle is set as the *target revision* for the deployment group. In other words, the application revision that is currently targeted for deployment is the target revision. This is also the revision that is pulled for automatic deployments.
* Next, the CodeDeploy agent on each instance polls CodeDeploy to determine what and when to pull from the specified Amazon S3 bucket or GitHub repository.
* Finally, the CodeDeploy agent on each instance pulls the target revision from the Amazon S3 bucket or GitHub repository and, using the instructions in the AppSpec file, deploys the contents to the instance.
* CodeDeploy keeps a record of your deployments so that you can get deployment status, deployment configuration parameters, instance health, and so on.

**AWS CodeDeploy EC2 setup:**

1] Create EC2 instances with IAM role [ Because CodeDeploy will say to the EC2 instance to fetch the artifacts from S3/github.

IAM ----create role---EC2-----create policy---attach s3 policy…..enter rolename[EC2RoleFOrCodewDeploy]……..create role…..&& attach this role to EC2 instance

It Allows EC2 instances to call AWS services on your behalf.

2] Install CodeDeploy Agent in EC2.



* sudo yum update
* sudo yum install ruby
* sudo yum install wget
* cd /home/ec2-user

sudo wget <https://aws-codedeploy-ap-south-1.s3.amazonaws.com/latest/install>

This url based on region and bucket it downloads resource kit

* sudo ./install auto
* sudo service codedeploy-agent status

**CodeDeploy Application Deployment Group && 1st Deployment**

<<……………start………………………………..>>

**Step 1: Create Application[APP]**

* Application Name:
* Compute Platform : EC2/onpremise / Lambda /ECS

**Step 2: Create Deployment Group [DEV/QA/PROD]**

* Deployment Group name: <DEV> <QA> <PROD>
* Service Role : Create Role for codeDeploy to call AWS services such as Autoscaling on your behalf and attach here.
* **Deployment Type**
* InPlace : Instances will be taken offline while upgrade
* Blue/Green: No Downtime but supports only for EC2
* **Env Configuration**
* It based on Inplace or Blue green deployment

Select EC2 ASG / EC2 /

* **Deployment Settings……..[Inplace] 1 at a time, half at a time or full at a time means downtime**
* **Load Balancer**

**Step 3: Create Deployment**

* **Choose the Artifacts from S3 or Github**
  + [Make sure you already created the s3 bucket with versioning and pushed the build folder along with appspec.yaml file and scripts folder]

Manual steps for creating S3 bucket & deploying artifacts.

**Create Bucket**

* + aws s3 mb s3://reactjs-master --region ap-south-1

**Enable Versioning:**

* + aws s3api put-bucket-versioning --bucket reactjs-master --versioning-configuration Status=Enabled --region ap-south-1

**Deploy build folder with appsec file & scripts folder**

* + aws deploy push --application-name Reactjs-Photgallery --s3-**location** s3://reactjs-master/artifacts/app.zip --ignore-hidden-files --region ap-south-1
  + Example Artifacts for Reactjsphotogalerry application:

 

* + **Artifacts**:
  + It should contain **appspec.yaml** file
  + Hooks lifecycle for EC2
  + Artifacts folder structure:
  + Build folder……scripts folder [nginx start & stop script] ….appsecfile…..buildspecfile
  + **appspec.yaml**

  

* **Additional Deployment Behaviour**
  + Overwrite the content
  + Retain the content
* **Deployment Group overrides**
* **Rollback options**
* **Submit…..which start the deployment**

**<<………………….End……………………..>>**

Hooks Lifecycle:

* Application Stop
* Download Bundle from S3………no scripts required
* BeforeInstall
* Install………………………………………no scripts required
* After Install
* Application Start
* Validate Service

Env Variable for Hooks:

LIFECYCLE\_EVENT : This variable contains the name of the lifecycle event associated with the script.

DEPLOYMENT\_ID : This variables contains the deployment ID of the current deployment.

APPLICATION\_NAME : This variable contains the name of the application being deployed. This is the name the user sets in the console or AWS CLI.

DEPLOYMENT\_GROUP\_NAME : This variable contains the name of the deployment group. A deployment group is a set of instances associated with an application that you target for a deployment.

DEPLOYMENT\_GROUP\_ID : This variable contains the ID of the deployment group in AWS CodeDeploy that corresponds to the current deployment.

**Monitoring Deployments with CloudWatch Events**

You can monitor CodeDeploy deployments using the following CloudWatch tools: Amazon CloudWatch Events, CloudWatch alarms, and Amazon CloudWatch Logs.

You can use Amazon CloudWatch Events to detect and react to changes in the state of an instance or a deployment (an "event") in your CodeDeploy operations. Then, based on rules you create, CloudWatch Events will invoke one or more target actions when a deployment or instance enters the state you specify in a rule. Depending on the type of state change, you might want to send notifications, capture state information, take corrective action, initiate events, or take other actions. You can select the following types of targets when using CloudWatch Events as part of your CodeDeploy operations:

* AWS Lambda functions
* Kinesis streams
* Amazon SQS queues
* Built-in targets (CloudWatch alarm actions)
* Amazon SNS topics

Cloud watch ….Events…create rule……select code deploy under Event selector…..specifu the rule on wht changes it shloud trigeer [failure start success….]….choose the application and choose deployment grp…&&&choose the target .

Use cases:

The following are some use cases:

* Use a Lambda function to pass a notification to a Slack channel whenever deployments fail.
* Push data about deployments or instances to a Kinesis stream to support comprehensive, real-time status monitoring.
* Use CloudWatch alarm actions to automatically stop, terminate, reboot, or recover Amazon EC2 instances when a deployment or instance event you specify occurs.

**Cloudwatch Logs**

There is no direct integration with code deploy and cloud watch. We need to install code deploy agent and cloud watch agent which send the log to cloud watch

**CodeDeploy Roll backs**

**Roll back when a deployment fails**

* At any point of time any deployment fails on any instance automatically deployments roll backed.

**Roll back when alarm thresholds met**

* Roll backed whenever the EC2 utilization goes ablove the threshold [ which we shld set in cloud watch alarm example cpu utilization threshold 80% then it shld trioger alarms so that deployment gets roll backed]

**CodeDeploy Onpremise setup**

**STEP 1: Create IAM user with programtic access and s3 policy**

**STEP 2 : Add config file to onpremise instance**

* Mkdir /etc/codedeploy-agent/conf
* Vim codedeploy.onpremises.yml
* ---
* aws\_access\_key\_id: secret-key-id
* aws\_secret\_access\_key: secret-access-key
* iam\_user\_arn: iam-user-arn
* region: supported-region

**STEP 3: Install AWS CLI and configure**

**STEP 4: Set AWS\_REGION env variable**

**STEP 5: Install CodeDeploy Agent**

**STEP 6: Register the on-premise with CodeDeploy**

aws deploy register-on-premises-instance --instance-name AssetTag12010298EX --iam-user-arn arn:aws:iam::80398EXAMPLE:user/CodeDeployUser-OnPrem

**STEP 7: Tag the on-Premise instances**

aws deploy add-tags-to-on-premises-instances --instance-names AssetTag12010298EX --tags Key=Name,Value=CodeDeployDemo-OnPrem

**Code Deploy in AWS LAMBDA**

**Code Deploy…..Create Application & computue platform is AWS lambda**

**Create Deployment group…attach policy [codedeploywithlamda]**