**AWS Certified DevOps Professional**

**CICD in AWS**

**Why CICD?** Manual steps make it likely to make mistakes.

We would like our code in a ‘repository’ and have it deployed to AWS

* *Automatically*
* *The right way*
* *Making sure its tested before being deployed*
* *Possibly going to different stages (dev,test,staging,prod)*
* *Manual approval when needed*

CI/CD adds **safety** and **speed**.

**Continuous integration (CI)**

Devs push code to repo i.e., github, codecommit, bitbucket

Test/build server checks code soon as its published i.e., codebuild, Jenkins

This process finds bugs early, fixes them… code is delivered faster, deploy faster

**Continuous delivery (CD)**

Code in repo that is tested appropriately is released

**Code** (github etc) > **Build** (Jenkins, codebuild) > **Test** (Jenkins, codebuild) > **Deploy** > ec2, ecs, lambda, on prem, elastic beanstalk

***elastic beanstalk*** *can be used for they deployment stage and also to provision/host the app.*

**CodeCommit**

Version control with Git. Store code, collab, see changes and versions, rollback. Code stayes in AWS, no size limit on repos, integrates with industry tools.

Auth with **SSH keys** (conf in iam console), **HTTPS** for standard login and password to repo. Both allow encryption transit.

Encryption at rest with KMS automatically.

You can use an IAM role for cross-account access.

**Main** is the default branch in CodeCommit (was formerly **master**).

**Pull request** allows developers to merge their changes from a different branch into the main branch (or another branch)

**Commits** – view all commits, visualise them and you can compare branches

In **Settings** you can add **Notification Rules** (i.e., events that trigger notifications for comments, approvals, pull requests or branches). You can select all in one notification rule.

The target can be **SNS Topic**, or ChatBot (Slack or Teams)

**Eventbridge** – can be used to monitor any type of CodeCommit events (near real-time).

*Example events: PullRequestCreated, PullRequestStatusChanged, referenceCreated etc.*

This allows good automation possibilities.

**Migrate Git Repo to CodeCommit:**

1. Create code commit repo
2. Do a git clone (takes whole repo to local PC, all project files, all commits, everything)
3. Once cloned locally, push to a different URL i.e., the codecommit repo

With EventBridge you can use **Cross-Region Replication** to copy a repo to another region. Good to reduce latency for global developers.

**Branch Security:**

By default, any users with push permissions to a CodeCommit repo can contribute to any branch.

Use IAM polices to restrict this.

**Pull Request Approval Rules:**

Helps ensure quality of code by requiring users(s) to approve open PRs before the code can be merged.

A pool of users (any IAM principal with ARN i.e., users, federated users, roles, groups) can be selected to approve the request (also define the number of people required to approve).

**CodePipeline**

Visual workflow to orchestrate your CI/CD pipeline within AWS

* Source – codecommit, ECR, S3, Bitbucket, GitHub
* Build – CodeBuild, Jenkins, CloudBees, TeamCity
* Test – CodeBuild, AWS Device Farm, 3rd party tools
* Deploy – CodeDeploy, Beanstalk, CloudFromation, ECS, S3…
* Invoke – Lambda, step functions
* Consists of stages:
  + Each stage can have sequential actions and/or parallel actions
  + Example: Build > Test > Deploy (to test) > Load Testing …
  + Manual approval can be defined at any stage

Each pipeline can create **Artifacts** – artifacts is whatever is created out of the pipeline.

Artifacts are stored in an S3 bucket and passed on to the next stage to do what it needs to do.

**Artifact Example:**

* Developer pushes code to **CodeCommit** repo
* CodeCommit (which is **orchestrated by codepipeline**) will extract all the code and create an output artifact out of it and place that artifice into an **S3 bucket**
* When **CodeBuild** is **invoked**, the same artifacts that were extracted are inputted to CodeBuild. *CodeBuild does not need straight access into CodeCommit as it is orchestrated by codepipeline which pushes the code through S3.*
* Once CodeBuild builds the code it creates a new output artifact stored in S3
* CodePipeline pushes these artifacts yet again, this time to **CodeDeploy**

**Codepipeline troubleshooting:**

For pipeline/action/stage execution state changes

**Eventbridge** – can be used to create events for failed pipelines, cancelled pipelines etc. and receive an email notification.

If there is a failure at a stage, you will see it visually through the console.

If there is no way for CodePipeline to perform a specific action i.e., invoke some code in CodeBuild or pull code from CodeCommit, then check the IAM service role of CodePipeline and make sure it has the right permissions.

Also you can look for any deny API calls in CloudTrail.

Code pipeline requires am **IAM role** to talk to the other development services.

**A pipeline must have at least two stages. The second stage must be either a Build or Deployment stage.**

**You can configure rollback on stage failure for deployment.**

**Stages can have multiple ACTION GROUPS. Action groups cannot have multiple stages. This can be a manual Approval and/or Deployments etc. Action groups can be SEQUENTIAL and/or PARALLEL.**

**History** will show you how long everything took and the status’.

**CodePipeline – Events vs. Webhooks vs. Polling**

**Events** – starts a pipeline whenever we have an event (default and preferred way). These are very fast as codepipeline is triggered as soon as the event happens.

i.e., a new commit happens and there is an EventBridge trigger which starts a Pipeline.

If the repo was a **third party** (GitHub), then a **CodeStar** source connection can be used to connect GitHub to AWS and then from this a pipeline can be triggered.

**Webhooks** – older way of triggering CodePipeline. CodePipeline exposes an **HTTP endpoint** – that endpoint can be triggered by a script. The script sends a payload to codepipeline on that webhook then the pipeline starts.

**Polling** – codepipeline can poll the source – this will mean regular checks to the source repo. Not recommended as it is not as efficient as events.

**CodePipeline Manual Approval Stage**

*CodeCommit > CodeBuild >* ***Manual Approval*** *> CodeDeploy*

The **‘owner’** is AWS because it is an AWS pipline. The action is **‘Manual’** because it says manual approval.

With manual approval you can trigger an SNS topic that sends an email to an IAM user in AWS, then you will need to approve.

The IAM user needs permissions to approve:

**Codepipeline:GetPipeline\*** (to view the pipeline)

**CodePipeline:PutApprovalResult** (to say yes or no, we approve or we deny)

**CodePipeline – CloudFormation as a Target:**

**CloudFormation** is a **Deploy Action** that can be used to deploy AWS resources. Works with StackSets to deploy across multiple AWS accounts and Regions.

Example: deploy Lambda Functions using CloudFormation or CDK or SAM (alternative to CodeDeploy)

* Code will be in CodeCommit (the code could be the CF template itself)
* Create change set in CF
* Manual Approval to ensure the change set is what we want it to be (can deploy without this stage)
* Execution of the change sets (if approved)

Example 2:

* CodeBuild builds app
* Creates a template .yaml file out of the build
* CF deploys the infra in the app using CREATE\_UPDATE mode
* CodeBuild tests app (make sure app is functioning as expected, could be functional tests load tests etc.)
* If all is good the test infra can be deleted using the DELETE\_ONLY action for CF to delete the stack
* Another CF action deploys the prod infra using CREATE\_UPDATE action

**Action modes**

Create or Replace a ChangeSet, Execute a Change Set (would need manual approval)

Create or Update a Stack, Delete a Stack, Replace a Failed Stack

**Template Parameter Overrides**

Specify JSON object to override parameter values

Object can be part of the input artifact you give to CodePipeline

Static – use template config file

Dynamic – use parameter overrides

**CodePipeline – Best Practices**

One CodePiplelie and One CodeDeploy to **multiple Deployment groups** – useful when deploying to multiple environments at a time or multiple deployment groups.

**Parallel Actions** – specify same **RAunOrder value** to speed things up in codepipeline by running two CodeBuilds at a time. Helpful when building and deploying to multiple regions

Having a **Pre-Prod** before Prod and and manual approval before prod deployment. This is a common pattern.

**EventBridge**

All CodePipleline events are detected by EventBridge (state execution changes or detect failures)

The **Invoke Action** uses a Lambda and invokes it within a Pipeline to run an API call from CodePipeline

**Step Functions** – starts a State Machine within a Pipeline

**Multi-region** – actions in a pipeline can be in different regions. i.e., deploy a lambda through CF into multiple regions.

You need **S3 Artifact Stores** defined in each region where you have actions.

CodePipeline must also have **read/write** access into every artifact bucket. Using the console creates default artifact buckets. Else you must create them if using CDK or CF for example.

**CodeBuild**

Allows you to take a source of code (codecommit, S3, GitHub, Bitbucket)

In that source there will be some Build instructions in a file (**buildspec.yml**) that needs to live in the root of your code. You can also insert these instructions manually in the console. Best practice is to use **buildspec.yml**.

Once the applications is built, the output logs can be stored in **S3** and **CloudWatch Logs** for later analysis.

**CloudWatch Metrics** can look at the build statistics

**EventBridge** to detect failed builds and trigger notifications

**CloudWatch Alarms** to notify of any failures

Build projects can be defined within CodePipeline or CodeBuild. Codepipeline can invoke an existing CodeBuild project.

**SUPPORTED ENVIRONMENTS**

Pre-built image for you to run in CodeBuild

Java, Ruby, Python, Go, Node.js, Android, .NET Core, PHP

For any other environments:

You can extend a Docker image to test whatever language. Up to you to support your own environment.

Code repo (containing source code and buildspec.yml)

CodeBuild (fetches source code, runs a container which is the build environment, this container loads the source code and runs instructions for buildspec.yml)

**Docker image** for CodeBuild can be **pre-packaged** or **custom**.

Some build files can be lengthy – in this case some files can be **cached in S3**.

If enabled, all logs will go to CloudWatch logs and S3.

One CodeBuild is done – it can produce some **artifacts** in an S3 bucket. This is where you can find your final outputs of CodeBuild.

**buildspec.yml layout**

* must be at root of code
* **env** – define environment variables for execution of buildspec.yml:
  + variables – plaintext variables
  + parameter-store – variables stored in SSM Parameter Store
  + secrets-manager – variables stored in AWS Secrets Manager (passwords and secrets)
* **phases** – specify commands to run (defines what CodeBuild will be doing):
  + **install** – install dependencies you may need for your build (i.e., installing pre-necessary packages)
  + **pre\_build** – final commands to execute before just build
  + **build** – the actual build commands
  + **post\_build** – finishing touches (e.g., create a zip output)
* **artifacts** – what to upload to S3 (encrypted with KMS). What files from the docker container should be extracted and sent into S3.
* **cache** – block to say which files in your dependencies are going to be cached in (in S3) for spinning up future builds

*You can run CodeBuild locally in case of deep troubleshooting beyond logs. You first need to install Docker and then you leverage the CodeBuild agent.*

**VPC**

By default, CodeBuild containers are launched outside your **VPC**, but can be launched within to access resources within a VPC.

You can specify VPC config:

* **VPC ID**
* **Subnet IDs**
* **Security Group IDs**

Then your build can access resources in your VPC (RDS, EC2, ALB etc.)

Use cases for being inside a VPC: Integration tests, data query, talk to internal load balancers etc.

**CodeBuild – Environment Variables**

**Default Environment Variables** – defined and provided by AWS for every part of your Build.

Includes default region, build ARN, build ID, build image and other meta data

**Custom Environment Variables:**

* **Static** – defined at build time (override using start-build API call if needed)
* **Dynamic** – pull value from SSM Parameter Store or Secrets Manager

**Security:**

* Use a service **Role** to access resources
* In-transit and at-rest encryption for cache, logs etc
* Build output artifacts are encrypted using KMS

**Build Badges** – show a badge to display status of latest build (i.e., a build is starting or failing) . Dynamically generated.

Badges are at the branch level. Can be for CodeCommit, GitHub, BitBucket etc.

**Trigger examples:**

CodeCommit > EventBridge > CodeBuild

CodeCommit > EventBridge > Lambda > CodeBuild

GitHub > WebHook > CodeBuild

**Validate Pull Requests**

Validate proposed code changes in PRs before they get merged.

Ensures high level of code quality and avoid code conflicts. Can also display the badge on your PR.

**Test Reports**

Get a visual report from CodeBuild UI. See pass rate % and see what is failing. No need to go through logs.

* Contains **detail about tests** that are run during builds:
  + **Unit tests, Configuration tests, functional tests**
* Create **test cases** with any **test** **framework** that can create **report files** in the formats:
* Junit XML, NUnit XML, NUnit 3 XML
* Cucumber JSON, TestNG XML, Visual Studio TRX

**Whatever framework is used, automatically you can create test reports directly in CodeBuild. REPORT GROUPS CAN BE DEFINED in buildspec.yml**

**CodeDeploy**

* Deployment Service that automates application deployment.
* Deploy new versions of apps to EC2, on prem, lambda or ECS.
* Automated rollback capability in case of failed deployments or trigger CloudWatch alarm
* Gradual deployment control
* **appspec.yml** defines how deployment happens

**EC2/On-prem Platform**

* Deploys to EC2 or prem
* **In-place** deployments or **blue/green** deployments
* Must run **CodeDeploy** **Agent** on the target instances
* Define deployment speed:
  + **AllAtOnce –** most downtime as this updates all instances at once
  + **HalfAtATime –** reduce capacity by 50%
  + **OneAtATime** – slowest, lowest availability impact
  + **Custom** – define your %

**In-Place** Deployment works for Half At A Time setting

**Blue-Green** – two app environments running in parallel – one running the old version one running the new. Load balancer switches to new env version. Simplifies rollback process.

**CodeDeploy Agent**

Must be installed on the EC2 instances as pre-requisites. Either using commands or via **SSM** to automate it.

Instances must have sufficient permissions to **access S3** to get deployment bundles.

S3 will store the **application revision**. When updating between versions, the codedeploy agent (with the appropriate permissions attached to the EC2) will download the application revision from the bucket.

**CodeDeploy – Lambda Platform**

Automates traffic shift for Lambda aliases (shift from V1 to V2 under a prod alias).

**Deployment strategies:**

Traffic can be shifted by a varying percentage using **X**

* **X can grow Linear – grows traffic % every N minutes until 100%**
  + ***LambdaLinear10PercentEveryMinutes***
* **X can grow via Canary – try X percent then ramp up straight to 100%**
  + ***LambdaCanary10Percent5Minutes***

Feature is integrated within the SAM framework

* **AllAtOnce** – immediate switch (no time to test if its working)

**CodeDeploy – ECS Platform**

CodeDeploy can help you automate the deployment of a new **ECS Task Definition.**

**Only works for Blue/Green Deployments.**

There will be two **Target Groups** in a **ECS Cluster** one V1 one V2.

***CodeDeploy affects the LB which redirects the traffic from V1 to V2.***

**Deployment strategies:**

Traffic can be shifted by a varying percentage using **X**

* **X can grow Linear – grows traffic % every N minutes until 100%**
  + ***LambdaLinear10PercentEveryMinutes***
* **X can grow via Canary – try X percent then ramp up straight to 100%**
  + ***LambdaCanary10Percent5Minutes***
* **AllAtOnce** – immediate switch (no time to test if its working)

**CodeDeploy – In-place Deployments for EC2**

Use EC2 Tags ASG to identify instances you want to deploy to

In the ASG scenario, for any new instance launched – CodeDeploy will deploy with the new version.

**Hooks** – scripts run during EC2 instance update lifecycle

Some moments to run scripts are blocked by CodeDeploy

**appspec.yml** defines the Hooks. i.e., ‘how should these scripts be run?’

**Deployment Hooks Examples:**

**BeforeInstall** – used for preinstall tasks, such as decrypting files, backing up current version

**AfterInstall** – used for tasks such as configuring your application or changing file permissions

**ApplicationStart** – start services stopped during ApplicationStop

**ValidateService** – verify deployment was completed successfully

**BeforeAllowTraffic** – run tasks on EC2 instances before registered to the LB i.e, health checks

**CodeDeploy – Blue/Green Deployments for EC2**

**For Blue/Green you must have a load balancer.**

Tags – manual mode

ASG – automatic deployment

**BlueInstanceTerminationOption** – For blue/green you can specify instance termination wait time, default 1 hr max 2 days.

Action Keep Alive – instances are kept running until you terminate (but deregistered from ELB and deploy group).

**Deployment hooks are the same.**

**Depoyment Configurations –** Allows for control over the deployment configuration.

Specifies the **number of instances that must remain available** at any time during the deployment.

* Predefined Deployment Configurations
  + **CodeDeployDefault.AllAtOnce** – deploy to as many instances as possible at once
  + **CodeDeployDefault.HalfAtATime** – deploy to up to half of the instances at a time
  + **CodeDeployDefault.OneAtATime** – deploy to only one instance at a time

***You can also create your own custom deployment configurations.***

**Triggers –** CodeDeploy can publish instance events to SNS. i.e., success or failure etc.

**CodeDeploy – Deployment to ECS**

Automatically handle a deployment of a new ECS Task Definition to an ECS service.

Only supports blue/green and the service must be connected to an LB.

The ECS Task Definition and new Container Images must be already created.

**appspec.yml** must be in an S3 Bucket for reference by the Task Definition. This is used by CodeDeploy.

Hooks on ECS are Lambda Functions to be executed once per deployment.

**CodeDeploy – Deployment to Lambda**

Specify lambda version to deploy in appspec.yml file.

CodeDeploy updates a Lambda function with the new Lambda function version.

CodeDeploy Agent not required.

All you need is: Name, Alias, CurrentVersion and TargetVersion in the appspec.yml file.

Hooks – a lambda function to be executed once per deployment. BeforeAllowTrafic. AfterAllowTraffic checks health.

**CodeDeploy** deployments can be **rolledback** manually or automatically. **Last known good version** is deployed.

Time must be synced between ec2 and codedeploy.

**Potential failures** for **troubleshooting:**

AGENT not installed or running, no IAM role, using HTTP Proxy (configure **:proxy\_uri**), date and time sync

**CodeArtifact** – artifact management system.

Artifact management is storing and retrieving software package dependencies (software packages that depend on one another to be built).

Has integration with Gradle, npm, yarn, twine, pip and NuGet.

All artifacts live within a VPC. Can integrate with EventBridge.

You can have a **resource policy** to access packages.