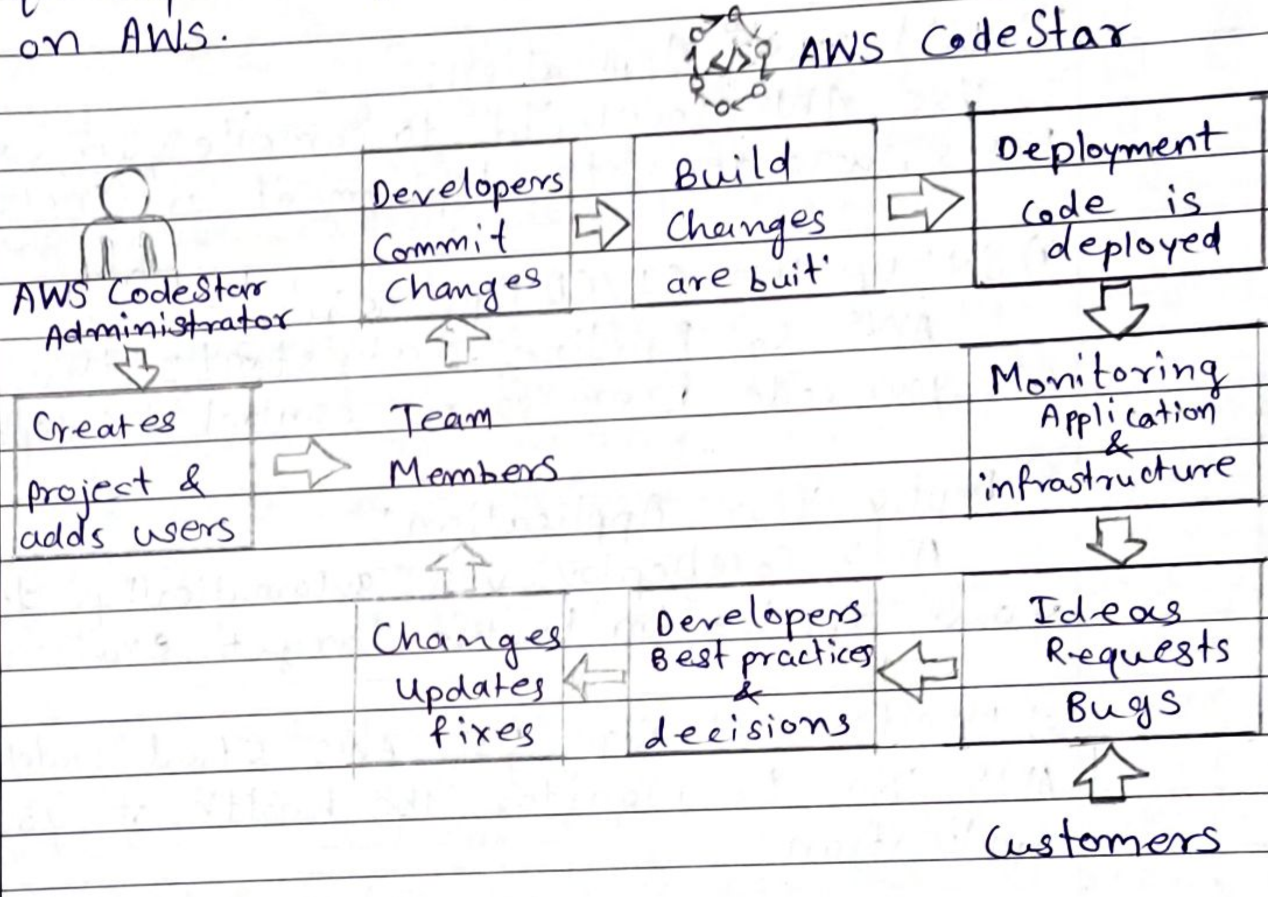
Assignment No. 01

Q.1) Demonstrate the deployment of an application using AWS CodeStar. What are the key components involved?

→ Deploying an application using AWS CodeStar involves several key components & steps. AWS CodeStar is a service that enables you to quickly develop, build, and deploy applications on AWS.


AWS CodeStar
AWS CodeStar Administrator
Creates project & adds users
Team Members
Developers Commit Changes
Build Changes are buit
Deployment code is deployed
Monitoring Application & infrastructure
Ideas Requests Bugs
Customers
Developers Best practices & decisions
Changes Updates fixes

Development Workflow of AWS CodeStar

Steps to Deploy an Application Using AWS CodeStar:

1) Create CodeStare Project:
Start by creating a new project in AWS CodeStar.

2) Configure Your Repository:
AWS codestar will create a CodeCommit repository for you.

3) Build Your Application:
Use AWS CodeBuild to compile your code, run tests, and generate deployment artifacts.

4) Set Up the CI/CD Pipeline:
AWS CodePipeline orchestrates the flow of your code from source control to deployment.

5) Deploy Your Application:
AWS CodeDeploy will automatically deploy your application to the target environment.

6) Monitor & Manage: Use AWS Cloud watch & AWS SNS to monitor the health of your application

The key Components Involved are:
1) AWS CodeStar Project
2) AWS CodeCommit
3) AWS Code Build

4) AWS Code Pipeline
5) AWS CodeDeploy
6) AWS Cloudformation
7) AWS IAM (Identity & Access Management)
8) AWS CloudWatch
9) AWS SNS (Simple Notification Service)

Q.2) Set up a simple CI/CD pipeline using AWS CodePipeline to deploy a sample web application. What steps are involved?

→ Setting up a simple CI/CD pipeline using AWS CodePipeline to deploy a sample web application involves several steps:

* Step 1: Prepare Your Sample Web Application:

① Create a Sample Application:-
- Develope a simple web application (e.g., a static HTML page or a Node.js app) on your local machine.
- Initialize a Git repo in the applications directory.

② Create a Build Specification File (buildspec.yml):-
- The buildspec.yml file defines the build commands & settings for AWS CodeBuild. An example for a Node.js application might look like this:

```
version: 0.2
phases:
  install:
    commands:
      - npm install
  build:
    commands:
      - npm run build
artifacts:
  files:
    - '**/*'
```

* Step 2: Set Up AWS CodeCommit Repository
① Create a CodeCommit Repository:
- In the AWS Management Console, navigate to CodeCommit and create a new repository.
- Clone the repository to your local machine and push your sample application code to it.

* Step 3: Set Up AWS CodeBuild
① Create a CodeBuild Project:
- In the AWS Management Console, go to CodeBuild & create a new project.

* Step 4: Set Up AWS Code Deploy
① Create an EC2 Instance:
- Launch an EC2 instance (e.g., Amazon Linux or Ubuntu) where the application will be deployed.

② Create a CodeDeploy Application:
- In the AWS Management Console, navigate to CodeDeploy & create new application.

③ Create an AppSpec File (appspec.yml):
- The appspec.yml file defines how your application should be deployed.

* Step 5: Set Up AWS CodePipeline
① Create a CodePipline:
- In the AWS Management Console, navigate to CodePipline & create a new pipline.

② Configure the Pipeline:
- Review & confirm the pipline stages.

* Step 6 : Test the Deployment :
① Push Changes to CodeCommit :
· Make changes to your sample application and push them to CodeCommit repository.

② Monitor the Pipeline:
- Go to the CodePipeline dashboard to monitor the stages.

③ Verify the Deployment:
- Once the deployment, is complete, access the application via the public IP address or DNS of the EC2 instance to ensure it's working correctly

Q.3) List & explain the steps required to install Kubernetes on a Linux system.
→ The Steps required to install kubernetes on a Linux System:

1) Prepare the Linux Environment:
- Update your system, disable swap, set hostnames, and ensure network settings are correctly configured for kubernetes.

2) Install Docker:
- Install Docker as the container runtime for kubernetes.

3) Install Kubernetes Components:
- Add the kubernetes repository & install kubeadm, kubelet, and kubectl.

4) Initialize the Kubernetes Control Plane:
- On the master node, initialize the Kubernetes control plane to set up the cluster.

5) Install a Pod Network Add-on:
- Install a network plugin (e.g., Calico, Flannel) to ennable communication between pods across the cluster.

6) Join Worker Nodes to the Cluster:
- Use the kubeadm join command on each worker node to connect them to the kubernetes cluster.

7) Verify the cluster installation:
- Check the status of the nodes and the pod network to ensure the cluster is functioning correctly.

8) Deploy a Test Application:
- Deploy a simple application (like Nginx) to verify that the kubernetes installation is successful.

Q.4) Compare and contrast Kubernetes Services and Ingresses. How do they differ in exposing deployments.?

->

Kubernetes Services	Kubernetes Ingresses
① It exposes a set of Pods as a network service.	① It manages external access to services within the cluster
② It operates at Layer 4 (TCP / UDP).	② It Operates at Layer 7 (HTTP / HTTPS).
③ The Routes of traffic are based on IP address and port.	③ The Routes of traffic are based on HTTP host & path.
④ It is simple round-robin load balancing.	④ It supports advanced load balancing features, like URL-based routing.
⑤ SSL/TLS termination is not inherently supported.	⑤ Supports SSL/TLS termination & HTTPS routing
⑥ It allows limited customization (mainly port-based).	⑤ It allows for complex routing rules based on host and path.

- Kubernetes Services expose deployments by providing a stable IP & port, allowing direct access within or outside the cluster, depending on the service type (cluster IP, Node Port, Load Balancer). In contrast, Ingresses expose deployments by routing external HTTP/HTTPS traffic to specific services based on defined host and path rules, requiring an ingress controller for managing external access.

Q.5) List & explain the basic commands in Terraform to install build & destory infrastructure.

-> Basic Terraform Commands:

1) terraform init:
- This command initialize a Terraform working directory. It sets up the backend for storing the state and downloads the necessary provider plugins specified in your configuration files.

2) terraform plan:
- This command creates an execution plan, showing you what Terraform will do when you apply your configuration. It doesn't make any changes, but previews the actions that will be taken, allowing you to review them before applying.

3) terraform apply:
- This command applies the changes required to reach the desired state of the configuration. It builds or updates the infrastructure as defined in your Terraform files. After you review the plan, Terraform will prompt you to confirm

before proceeding with the actual changes

4) terraform destroy:
-This command is used to destroy the infrastructure managed by Terraform. It removes all the resources defined in the configuration files, effectively tearing down the environment.

Q.6) Create a basic Terraform configuration that defines an AWS S3 bucket. What steps do you follow?

→ To define an AWS S3 bucket using Terraform, follow these steps:

1) Install Terraform:
Ensure Terraform is installed on your system.

2) Set Up your working Directory:
Create a directory for your terraform configuration files.

3) Create a Terraform Configuration file:
Write a main.tf file defining the AWS provider and the S3 bucket resources.

4) Initialize Terraform:
Initialize the Terraform working directory to set up provider plug ins.

5) Validate the configurations:
Check the syntax errors and validate the configuration.

6) Preview the changes:
Review the changes terraform will make with a plan.

7) Apply the Configuration:
Create the S3 bucket as defined in your configuration.

8) Verify the Bucket:
Confirm the bucket creation in the AWS S3 console.

9) Destroy the Infrastructure (optional):
Remove the bucket and resources if needed.