**Day-1 | Introduction to AWS | What is Public Cloud? | Create an AWS Account**

* Buy datacenter: servers from IBM or other company.
* Server: costly, 1CPU, 100gb (costly)
* Virtualization: Create virtual/layer server on top of actual server, each virtual server we can deploy appn in any part of world and share it. (Create 5 vm provide ip to access)  
  -> just buy one server and deploy multiple appn on
* Virtualization:
* Private cloud: Private to your organization, manage and maintain cloud platform (Banking companies, sensitive information): buy server, create own datacenter use platform like (opestack, VM ware, Xen)
* Public cloud: Startup uses it

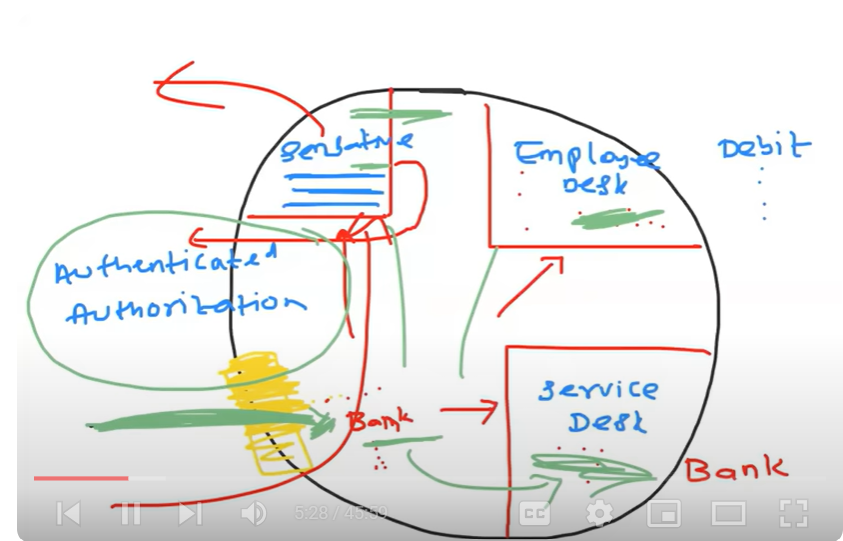
Why public cloud popular?

* Cost, avoiding headache of maintenance.

AWS Popularity: Pioneers of cloud concept, largest market share

* **Cloud repatriation:** moving from cloud to on-premise(Private cloud)  
  - Security, money is same
* **Preview:** Cloud(Private, public), AWS, Public

**Day-2 | AWS IAM deep dive with practicals and notes | IAM Project |**  
**(**[**https://www.youtube.com/watch?v=mCLYcsJ0GXQ&list=PLdpzxOOAlwvLNOxX0RfndiYSt1Le9azze&index=3**](https://www.youtube.com/watch?v=mCLYcsJ0GXQ&list=PLdpzxOOAlwvLNOxX0RfndiYSt1Le9azze&index=3)**)**

* Bank: Service desk, Employee desk (debit), Sensitive (cheque, money, user details etc)  
  - Authenticated person can enter: Bank account  
  - Verify Authorization: What can you perform  
    
  

* **AWS IAM:** *Authentication and Authorization*  
  - Company may have EC2, DB, Storages, K8s : no one should delete anything unknowingly
* Devops engineer will create an account(Root access)  
  - 501 user needed access: Devops Er will provide required access to him (Read access to DB, Write access to K8s)  
  - Devops Er will go to IAM and create a user and provide policies to it.
* **Devops Er: Root access: Do anything in AWS**  
  - ***User:*** Authentication: For employees to allow  
  - ***Policies:*** Authorization: What he can do  
  - ***Group:*** (Dev, QA, DB-Admins, Others in an organization), keep groups ready and whenever new employee joins create a user for him and push him to Group.  
  - ***Roles:*** Appn accessing from AWS service, temporary purpose, talk between AWS services
* **Task1:** Create a user and try to logins AWS console and provide him S3 bucket policy to read and write access to it. (While creating keep default and he has to change his password)
* AWS permission for users: AWS managed policy, Custom policy
* Assignment: Create user, login with user grant permission of S3

# Day-3 | EC2 Deep Dive | Deploy Jenkins on AWS |

# 

# EC2: Elastic cloud compute: (EC2: 1:elastic, 2:cloud and compute) - Elastic: Scaled up or Scaled down - Cloud: AWS public cloud - Compute: requesting AWS to provide CPU, RAM and Disk (VM machine) -Laptop: Physical server: 2 to 3 people: install hypervisor: multiple people can use it - AWS has multiple physical servers, it will take request to hypervisor and it will give you VM.

# Why: we can easily deploy more machines as we want but below things are challenge for us: - timely upgrade - security issues - servers - Management cost - Pay as you go

# - Cost

# Types of EC2 instance: a. General purpose: b. Compute optimized EC2 instance: higher ratio of power (ML, gaming) c. Memory optimized: Rich in terms of memory operation c. Storage optimized EC2 instance: High in storage d. Accelerated optimized EC2 instance:

# Region and Availability zones: - Latency: request taking time to move from one place to other place (If they are far away high latency) - Availability zones: within a region multiple availability zones will be their to provide high availability

# Task2: Create instance - Key value pair: Combination of public and private key, which is used to login instance - Instance will have public key and we will have private key. To connect instance: ssh -i C:\Users\Ramesh\_Nidode\Downloads\aws\_test.pem [ubuntu@51.20.108.90](mailto:ubuntu@51.20.108.90)

# Chmod 600 aws\_test.pem Install Jenkins:

# apt install openjdk-11-jdk

# java –version Link: https://www.jenkins.io/doc/book/installing/linux/ systemctl status Jenkins <http://publicip:8080> http://51.20.108.90:8080/

# Edit inbound rules: allow traffic to 8080 port from outside by editing inbound rule

# Now you can able to access Jenkins Credentials: Usenamr: rameshnidode96

# Password: ramesh@123

# Full name: Ramesh Nidode

# Emailed:

# 

# Day-4 | Best VPC explanation| VPC explained in 30 mins| 4K quality|

# - a specific focus on VPC (Virtual Private Cloud). You'll learn how to create and configure VPCs, subnets, and route tables, enabling you to design and manage the network infrastructure for your applications.

# Virtual Private Cloud: What, why, Components

# Region(Mumbai)- > AWS Datacenter -> Server1, 2, 3 etc.. (Created VMs inside it for clients) -> 2013-2014, easy to hack

# 

# VPC: Region (Mumbai)- > AWS Datacenter -> VPC -> Projects -> Subprojects -> Subnets - Size of VPC: IP Adress range (172.16.0.0/16-255X255 IP addresses, 65536)

# Splitting VPC ip for subprojects is called subnets!

# Internet Gateway: add conditions, good user, bad user/hacker

# Public subnet: A user first access inside VPC, by using Internet gateway he can access it.

# Loadbalancer: Forwards requests based on load,

# Route table: How should request go to the application or target group.

# User (Internet)-Internet gateway-public subnet-loadbalancer-route tables-target group of appn-security group(ports, ip adress)-allow access to appn

# 

# Day-5 | AWS Security Group and NACL | Theory + Practical | - emphasizes security best practices in AWS. You'll learn how to implement security measures such as security groups, network ACLs (Access Control Lists), and IAM policies to ensure the confidentiality, integrity, and availability of your AWS resources.

# AWS VPC [Security Groups and NACL]

# VPC introduces concept of virtual private cloud in the world of public cloud and adds security.

# Security group and NACL: Additional security

# Subnet: In VPC we will define IP range, pool of IP address. (65536 Ips, we can have those many applications components).

# Within projects, if we have group of sub-projects/application we can define private subnets (no internet).

# User can access Load balancer through Internet gateway and then LB talks/forward request to private subnet: 1. Add additional security layer of subnet (NACL) 2. Add smore security at EC2-Instance level i.e., security groups!

**AWS Security using Security Groups and NACL**

* AWS (Amazon Web Services) provides multiple layers of security to protect resources and data within its cloud infrastructure. Two important components for network security in AWS are Security Groups and Network Access Control Lists (NACLs). Let's explore how each of them works:

1. Security Groups: Security Groups act as virtual firewalls for Amazon EC2 instances (virtual servers) at the instance level. They control inbound and outbound traffic by allowing or denying specific protocols, ports, and IP addresses.

* Each EC2 instance can be associated with one or more security groups, and each security group consists of inbound and outbound rules.
* Inbound rules determine the traffic that is allowed to reach the EC2 instance, whereas outbound rules control the traffic leaving the instance.
* Security Groups can be configured using IP addresses, CIDR blocks, security group IDs, or DNS names to specify the source or destination of the traffic.
* They operate at the instance level and evaluate the rules before allowing traffic to reach the instance.
* **Security Groups are stateful**, meaning that if an inbound rule allows traffic, the corresponding outbound traffic is automatically allowed, and vice versa.
* Changes made to security group rules take effect immediately.

1. Network Access Control Lists (NACLs): NACLs are an additional layer of security that operates at the subnet level. They act as **stateless traffic filters** for inbound and outbound traffic at the subnet boundary.

* Unlike Security Groups, NACLs are associated with subnets, and each subnet can have only one NACL. However, multiple subnets can share the same NACL.
* NACLs consist of a numbered list of rules (numbered in ascending order) that are evaluated in order from lowest to highest.
* Each rule in the NACL includes a rule number, protocol, rule action (allow or deny), source or destination IP address range, port range, and ICMP (Internet Control Message Protocol) type.
* NACL rules can be configured to allow or deny specific types of traffic based on the defined criteria.
* **Subnets are stateless**, which means that if an inbound rule allows traffic, the corresponding outbound traffic must be explicitly allowed using a separate outbound rule.
* Changes made to NACL rules may take some time to propagate to all the resources using the associated subnet.

# In AWS, security is a *Shared Responsibility*

# Devops Engineer will take care security from company side.

# SG: Inbound and Outbound rules *(AWS assigns default security group, by default it allows outbound traffic accept port 25) (AWS deny all inbound traffics)*

# Inbound traffic: User trying access appn inside our VPC.

# Outbound traffic: Our application is trying to access outside application.

# *AWS assign default security group, by default it allows outbound traffic accept port 25:* - Port 25 is mailing service, does not want to record IP address of your application, there can be spamming activity.

# NACL: Network Access Control List -> NACL is applied at subnet level.

# Scenario: Development team to make it easy, they have allowed all traffic to EC2-Instance, all ips from anywhere. - This team ignored security of application, instead of accepting all traffic we can define what traffic we can deny. If we missed something in security group level but NACL plays very imp role here. - If we 10000 instances under one subnet, we can add deny traffic in NACL and add to specific subnet. - Security Groups does not have deny traffic option (It only allows) so we can make use of NACL to deny and allow traffic clearly to add additional security level.

# ## Practical:

# A screenshot of a computer Description automatically generated

# -> Assignment: VPC creation

# 

# Step1: Create a vpc and more

# Step2: Create an instance.

# Step3: Connect to it ssh C:\Users\Ramesh\_Nidode\Downloads\aws\_test.pem [ubuntu@51.20.108.90](mailto:ubuntu@51.20.108.90)

# Step4: sudo apt update and python3

# Step5: python3 -m http.server 8000 - try to access by providing “ <http://34.239.185.56:8000/>” - You can not access - NACL has rule like allow all traffic but Security group is restricting it, so we need to provide an inbound rule saying it has to allow this particular port to access your application.

# A screenshot of a computer Description automatically generated

# We have 1000 machines with all allowing 8000 port but organization does not want to allow it, so now NACL we can add rule saying “deny 8000 port”, this is how it works at subnet level not at instance level.

# NAC acts as first layer of defense and then it routes to Security groups.

# 

# Day-6 | Route53 explained in 15 mins | AWS FREE COURSE |

# Route 53: DNS service (DNS: Domain Name System)

# Domain names instead of IP Adress, DNS service maps domain to the IP Adress.

# IRCTC, amazon, flip-kart there we are trying to resolve load balancer Ip address using domain name.

# Route 53: - Not easy to remember ip of each application - IP Adress can change, restarted Load balancer so if IP is not static then it will change.

# 

# DNS keeps many records in Hosted Zones.

# Domain name purchasing from GoDaddy or Route 53

# Create Hosted Zones where we create DNS records.

# Route 53 support health checks on the web application or web server.

# After adding Route53 in our architecture,

# **Project:** Configure and manage a domain name using Route 53. You'll register a domain, set up DNS records, and explore advanced features such as health checks, routing policies, and DNS-based failover.

# Day-7 | AWS Project Used In Production | Complete Implementation| 1-Tier Architecture Model

# 

# A screenshot of a computer Description automatically generated

# A black background with yellow text Description automatically generated

# NAT Gateway: Network Address Translation (NAT) service. You can use a NAT gateway so that instances in a private subnet can connect to services outside your VPC, but external services cannot initiate a connection with those instances. (NAT gateway receives request from our application in instance and it converts apps IP to its IP and allows our app to access outside application).

# Autoscaling Group: Scale in different availability zones

# Load balancer: Balances load from outside world between our servers. (Path based routing, host-based routing etc.)

# Target Group: Target groups route requests to individual registered targets, such as EC2 instances, using the protocol and port number that you specify. You can register a target with multiple target groups. You can configure health checks on a per target group basis.

# Bastion Host or Jump server: In public subnet we create Bastion host to connect with our instance in private subnet. - Instead of directly connecting to instance we ca go through it and audit and monitor them easily.

# An Internet Gateway (IGW) is a horizontally scaled, redundant, and highly available Amazon VPC component that allows communication between instances in your Amazon VPC and the Internet.

# 

# Project:

# Step1: Create vpc – VPC & more(It creates public, private subnet, Route tables, Network connections{Internet gateway, vpc s3 endpoint}: all in 2 zones) – project: aws-prod-example- NO IP v6 – NAT Gateway: 1 per AZ – VPC endpoints: None – Create VPC -> Elastic IP: IP will be same even if EC-2 instance goes down. Static IP address.

# Step2: EC2 – Autoscaling Group – Create (cannot create directly) – Launch template: 1. Name: aws-prod-example 2. Template version description: proof of concept for app deploy in aws private subnet 3. OS image/ AMI : Ubuntu, t2.micro, key value pair, creare a new security group - Name: aws-prod-example - Description: Allow ssh access - Select vpc where you want to launch - Inbound security group rules: 1. ssh-22-anywhere 2. Custom tcp-8000-anywhere

# No EBS volume or anything

# Create autoscaling group.

# Desired capacity: 2

# Min: 1 Max: 4

# Scaling policies: None

# Add notification: we can do through sns: for now, none

# Step3: For Bastion host as our instances does not have public IP

# Create instance – bastion-host – ubuntu – t2.micro-keypair- Network setting – Allow ssh – Edit it add same VPC otherwise it can not access our app – Auto assaign public IP(without it no use) – Launch instance

# Now we will SSH to bastion, from their we are ssh to private subnet.

# Copy key value pair from personal laptop to bastion server

# ls | grep aws\_log

# scp -i /Users/Ramesh\_Nidode/Downloads/aws-demo.pem /Users/Ramesh\_Nidode/Downloads/aws-demo.pem ubuntu@184.73.50.130:/home/ubuntu

# Connect to bastion server to check copied keypair

# Take private IP of 2 instance created in private subnet:

# ssh -i aws\_login.pem ubuntu@private ip

# vim index.html

# <!DOCTYPE html> <html> <body> <h1>My first AWS Project to demonstrate apps in private subnet</h1> </body> </html>

# python3 -m http.server 8000

# (This application is running in one of EC2 instance and not in other instance,)

# Load balancer: Application load balancer

# Does Http and https traffic i.e. Layer 7 loadbalancer.

# Create- Name—Internet facing- Select your VPC- AZ Public subnet- select Security Group- Listeners and routing (Create target group-select instances to access- port for selected instances is 8000- Include as pending- Create target group)- Create load balancer

# Go to loadbalancer: Security group- Allow http traffic- Inbound traffic- edit-add new rule- http 80, anywhere from internet- save rules

# Access your application from link present in loadbalancer: DNS Name

**Project:**

* Design and configure a VPC: Create a VPC with custom IP ranges. Set up public and private subnets. Configure route tables and associate subnets.
* Implement network security: Set up network access control lists (ACLs) to control inbound and outbound traffic. Configure security groups for EC2 instances to allow specific ports and protocols.
* Provision EC2 instances: Launch EC2 instances in both the public and private subnets. Configure security groups for the instances to allow necessary traffic. Create and assign IAM roles to the instances with appropriate permissions.
* Networking and routing: Set up an internet gateway to allow internet access for instances in the public subnet. Configure NAT gateway or NAT instance to enable outbound internet access for instances in the private subnet. Create appropriate route tables and associate them with the subnets.
* SSH key pair and access control: Generate an SSH key pair and securely store the private key. Configure the instances to allow SSH access only with the generated key pair. Implement IAM policies and roles to control access and permissions to AWS resources.
* Test and validate the setup: SSH into the EC2 instances using the private key and verify connectivity. Test network connectivity between instances in different subnets. Validate security group rules and network ACL settings.

By implementing this project, you'll gain hands-on experience in setting up a secure VPC with EC2 instances, implementing networking and routing, configuring security groups and IAM roles, and ensuring proper access control. This project will provide a practical understanding of how these AWS services work together to create a secure and scalable infrastructure for your applications.

# Day-9 | AWS S3 Buckets Deep Dive | 2 Demo Projects with Code |

# Solves a very common problem, i.e., storage. 2nd service AWS has introduced in its journey. (2015)

# The secret behind the success is (11 9’s): 99.99999999999.

# S3: Simple storage service, highly scalable, No restriction to store and cheap. (Each object an not store more than 5TB)

# Store in anywhere in the world and access it from anywhere in the world.

# 

# 

# Mostly used to store logs of application organization using.

# 

# S3 bucket is a global services but why we should create it in a region?

# S3 are scoped in a region so that nearer the region latency will be less.

# By default, public access is blocked in S3 bucket, objects in S3 will be encrypted.

# Region of our S3 is 99.99999999999 reliable: In 1 billion objects present in S3, over period of 100 years only 1 object may face error. (Replication mechanism of AWS)

# A screenshot of a black screen Description automatically generated

# 

What is Amazon S3?

Simple Storage Service is a scalable and secure cloud storage service provided by Amazon Web Services (AWS). It allows you to store and retrieve any amount of data from anywhere on the web.

**What are S3 buckets?**

S3 buckets are containers for storing objects (files) in Amazon S3. Each bucket has a unique name globally across all of AWS. You can think of an S3 bucket as a top-level folder that holds your data.

**Why use S3 buckets?**

S3 buckets provide a reliable and highly scalable storage solution for various use cases. They are commonly used for backup and restore, data archiving, content storage for websites, and as a data source for big data analytics.

**Key benefits of S3 buckets**

S3 buckets offer several advantages, including:

1. Durability and availability: S3 provides high durability and availability for your data.
2. Scalability: You can store and retrieve any amount of data without worrying about capacity constraints.
3. Security: S3 offers multiple security features such as encryption, access control, and audit logging.
4. Performance: S3 is designed to deliver high performance for data retrieval and storage operations.
5. Cost-effective: S3 offers cost-effective storage options and pricing models based on your usage patterns.

## Creating and Configuring S3 Buckets

Creating an S3 bucket

To create an S3 bucket, you can use the AWS Management Console, AWS CLI (Command Line Interface), or AWS SDKs (Software Development Kits). You need to specify a globally unique bucket name and select the region where you want to create the bucket.

Choosing a bucket name and region

The bucket name must be unique across all existing bucket names in Amazon S3. It should follow DNS naming conventions, be 3-63 characters long, and contain only lowercase letters, numbers, periods, and hyphens. The region selection affects data latency and compliance with specific regulations.

Bucket properties and configurations

Versioning: Versioning allows you to keep multiple versions of an object in the bucket. It helps protect against accidental deletions or overwrites.

Bucket-level permissions and policies

Bucket-level permissions and policies define who can access and perform actions on the bucket. You can grant permissions using IAM (Identity and Access Management) policies, which allow fine-grained control over user access to the bucket and its objects.

## Uploading and Managing Objects in S3 Buckets

Uploading objects to S3 buckets

You can upload objects to an S3 bucket using various methods, including the AWS Management Console, AWS CLI, SDKs, and direct HTTP uploads. Each object is assigned a unique key (name) within the bucket to retrieve it later.

Object metadata and properties

Object metadata contains additional information abouteach object in an S3 bucket. It includes attributes like content type, cache control, encryption settings, and custom metadata. These properties help in managing and organizing objects within the bucket.

File formats and object encryption

S3 supports various file formats, including text files, images, videos, and more. You can encrypt objects stored in S3 using server-side encryption (SSE). SSE options include SSE-S3 (Amazon-managed keys), SSE-KMS (AWS Key Management Service), and SSE-C (customer-provided keys).

Lifecycle management

Lifecycle management allows you to define rules for transitioning objects between different storage classes or deleting them automatically based on predefined criteria. For example, you can move infrequently accessed data to a lower-cost storage class after a specified time or delete objects after a certain retention period.

Multipart uploads

Multipart uploads provide a mechanism for uploading large objects in parts, which improves performance and resiliency. You can upload each part in parallel and then combine them to create the complete object. Multipart uploads also enable resumable uploads in case of failures.

Managing large datasets with S3 Batch Operations

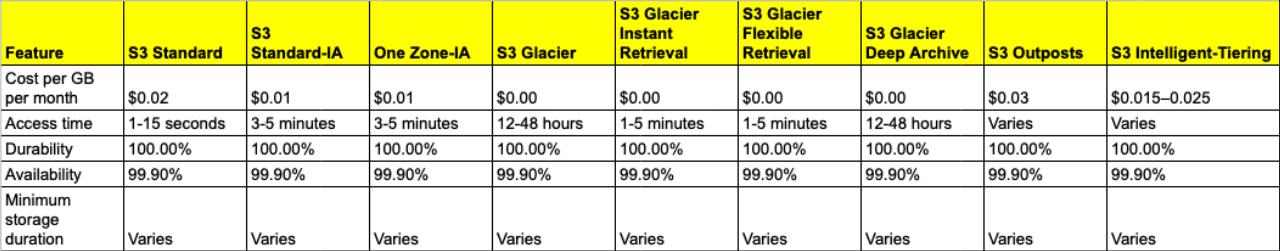
S3 Batch Operations is a feature that allows you to perform bulk operations on large numbers of objects in an S3 bucket. It provides an efficient way to automate tasks such as copying objects, tagging, and restoring archived data.

## 

**Advanced S3 Bucket Features:**

1. **S3 Storage Classes**

S3 offers multiple storage classes, each designed for different use cases and performance requirements:

[](https://private-user-images.githubusercontent.com/43399466/251470325-6b1ebcda-5b99-4358-ac1a-5bf559140571.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..4lRxJUFL6sA0pBEs-6YI7IA0tglqaTIpwksYYxD-9EQ)

1. **S3 Standard:**

* Designed for frequently accessed data.
* Stores data in a minimum of three Availability Zones

Amazon S3 Standard provides high availability for objects. This makes it a good choice for a wide range of use cases, such as websites, content distribution, and data analytics. Amazon S3 Standard has a higher cost than other storage classes intended for infrequently accessed data and archival storage.

1. **S3 Standard-IA (S3 Standard-Infrequent Access)**

* Ideal for infrequently accessed data
* Similar to Amazon S3 Standard but has a lower storage price and higher retrieval price

Amazon S3 Standard-IA is ideal for data infrequently accessed but requires high availability when needed. Both Amazon S3 Standard and Amazon S3 Standard-IA store data in a minimum of three Availability Zones. Amazon S3 Standard-IA provides the same level of availability as Amazon S3 Standard but with a lower storage price and a higher retrieval price.

1. **S3 One Zone-IA (S3 One Zone-Infrequent Access):**

* Stores data in a single Availability Zone
* Has a lower storage price than Amazon S3 Standard-IA

Compared to S3 Standard and S3 Standard-IA, which store data in a minimum of three Availability Zones, S3 One Zone-IA stores data in a single Availability Zone. This makes it a good storage class to consider if the following conditions apply:

* You want to save costs on storage.
* You can easily reproduce your data in the event of an Availability Zone failure.

1. **S3 Intelligent-Tiering:**

* Ideal for data with unknown or changing access patterns
* Requires a small monthly monitoring and automation fee per object

In the S3 Intelligent-Tiering storage class, Amazon S3 monitors objects’ access patterns. If you haven’t accessed an object for 30 consecutive days, Amazon S3 automatically moves it to the infrequent access tier, S3 Standard-IA. If you access an object in the infrequent access tier, Amazon S3 automatically moves it to the frequent access tier, S3 Standard.

**e. S3 Glacier Instant Retrieval:**

* Works well for archived data that requires immediate access
* Can retrieve objects within a few milliseconds

When you decide between the options for archival storage, consider how quickly you must retrieve the archived objects. You can retrieve objects stored in the S3 Glacier Instant Retrieval storage class within milliseconds, with the same performance as S3 Standard.

1. **S3 Glacier Flexible Retrieval  
   ->** Low-cost storage designed for data archiving

* Able to retrieve objects within a few minutes to hours

S3 Glacier Flexible Retrieval is a low-cost storage class that is ideal for data archiving. For example, you might use this storage class to store archived customer records or older photos and video files. You can retrieve your data from S3 Glacier Flexible Retrieval from 1 minute to 12 hours.

* 1. **S3 Glacier Deep Archive**

-> Lowest-cost object storage class ideal for archiving

-> Able to retrieve objects within 12 hours

S3 Deep Archive supports long-term retention and digital preservation for data that might be accessed once or twice in a year. This storage class is the lowest-cost storage in the AWS Cloud, with data retrieval from 12 to 48 hours. All objects from this storage class are replicated and stored across at least three geographically dispersed Availability Zones

* 1. **S3 Outposts**
* Creates S3 buckets on Amazon S3 Outposts
* Makes it easier to retrieve, store, and access data on AWS Outposts

Amazon S3 Outposts delivers object storage to your on-premises AWS Outposts environment. Amazon S3 Outposts is designed to store data durably and redundantly across multiple devices and servers on your Outposts. It works well for workloads with local data residency requirements that must satisfy demanding performance needs by keeping data close to on-premises applications.

1. **S3 Replication**: S3 replication enables automatic and asynchronous replication of objects between S3 buckets in different regions or within the same region. Cross-Region Replication (CRR) provides disaster recovery and compliance benefits, while Same-Region Replication (SRR) can be used for data resilience and low-latency access.
2. **S3 Event Notifications and Triggers:** S3 event notifications allow you to configure actions when specific events occur in an S3 bucket. For example, you can trigger AWS Lambda functions, send messages to Amazon Simple Queue Service (SQS), or invoke other services using Amazon SNS when an object is created or deleted.
3. **S3 Batch Operations:** S3 Batch Operations allow you to perform large-scale batch operations on objects, such as copying, tagging, or deleting, across multiple buckets. It simplifies managing large datasets and automates tasks that would otherwise be time-consuming.
4. **Security and Compliance in S3 Buckets  
   a. S3 bucket security considerations:** Ensure that S3 bucket policies, access control, and encryption settings are appropriately configured. Regularly monitor and audit access logs for unauthorized activities. **b. Data encryption at rest and in transit:** Encrypt data at rest using server-side encryption options provided by S3. Additionally, enable encryption in transit by using SSL/TLS for data transfers. **c. Access logging and monitoring:** Enable access logging to capture detailed records of requests made to your S3 bucket. Monitor access logs and configure alerts to detect any suspicious activities or unauthorized access attempts.
5. **S3 Bucket Management and Administration  
   a. S3 bucket policies:** Create and manage bucket policies to control access to your S3 buckets. Bucket policies are written in JSON and define permissions for various actions and resources. **b. S3 access control and IAM roles:** Use IAM roles and policies to manage access to S3 buckets. IAM roles provide temporary credentials and fine-grained access control to AWS resources. **c. S3 APIs and SDKs:** Interact with S3 programmatically using AWS SDKs or APIs. These provide libraries and methods for performing various operations on S3 buckets and objects.
6. **Monitoring and logging with CloudWatch:** Utilize Amazon CloudWatch to monitor S3 metrics, set up alarms for specific events, and collect and analyze logs for troubleshooting and performance optimization.
7. **S3 management tools:** AWS provides multiple management tools, such as the AWS Management Console, AWS CLI, and third-party tools, to manage S3 buckets efficiently and perform operations like uploads, downloads, and bucket configurations.
8. **Troubleshooting and Error Handling  
   a. Common S3 error messages and their resolutions:** Understand common S3 error messages like access denied, bucket not found, and exceeded bucket quota. Troubleshoot and resolve these errors by checking permissions, bucket configurations, and network connectivity. **b. Debugging S3 bucket access issues:** Investigate and resolve issues related to access permissions, IAM roles, and bucket policies. Use tools like AWS CloudTrail and S3 access logs to identify and troubleshoot access problems.  
     
     
    **c. Data consistency and durability considerations:** Ensure data consistency and durability by understanding S3's data replication and storage mechanisms. Verify that data is correctly uploaded, retrieve objects using proper methods, and address any data integrity issues.
9. **Recovering deleted objects:** If an object is accidentally deleted, you can often recover it using versioning or S3 event notifications. Additionally, consider enabling Cross-Region Replication (CRR) for disaster recovery scenarios.

# Day-10 | AWS CLI Deep Dive | Concept + Installation + Demo

# AWS command line interface acts a layer between user and AWS API.

# AWS UI is not automation friendly for infrastructure management or automation! (For quick and short actions it is used)

# AWS API: Application Programming Interface, us9ing this automation, creation, and deletion of resources.

# To automate infrastructure using AWS CLI(Quick use case option), Terraform(To create complete infrastructure), Cloud formation templates, Cloud development kit(CDK) etc.

# AWS CLI: Python utility/ Python program, using this u can pass some arguments. (Acts as layer between user and API) - U can access/start S3 bucket using API: Post api.aws.com/S3/create - provide name, versioning etc. to create.

# AWS CLI use cases: 1. Automation 2. Report generation 3. Usual interaction with AWS account.

# Abstraction layer: You don’t need to write programmatically, you just have to install python application and write paramaeters (./aws s3 name and other configuration) and AWS itself will convert this information to a program and it will create your service required by CLI.

# Invoke AWS CLI using shell etc, and pass input to AWS CLI using shell commands and AWS API translates which will be understood by AWS and it will create the resource.

# 

# Install, CLI, CLI reference docs

# To invoke API provide command: aws s3 create

# Pre-requisite for AWS CLI: Before using the AWS Command Line Interface (CLI), you need to ensure that you have the following prerequisites in place:

# AWS Access Key ID and Secret Access Key: can create and manage access keys in the AWS Management Console under the IAM (Identity and Access Management) section.

# AWS CLI Installation: Install the AWS CLI on your local machine. The installation steps depend on your operating system.

**Error:** If you're using Windows and the AWS CLI is already installed, but you're still getting the error, you may need to add the installation directory to your system's PATH.

Here's how you can do it:

1. Find the installation directory of the AWS CLI. The default path is usually **C:\Program Files\Amazon\AWSCLI** or **C:\Program Files (x86)\Amazon\AWSCLI**.
2. Add the AWS CLI directory to your system's PATH:
   * Right-click on "This PC" or "Computer" on your desktop or in File Explorer.
   * Choose "Properties."
   * Click on "Advanced system settings" on the left.
   * Click the "Environment Variables" button.
   * Under "System variables," find the "Path" variable, and click "Edit."
   * Click "New," and add the path to the directory where the AWS CLI is installed.
   * Click "OK" on all open windows.

# Python: The AWS CLI requires Python to be installed on your machine. The AWS CLI currently supports Python 3.6 and later.

# IAM User Permissions: Ensure that the IAM user associated with your access key pair has the necessary permissions to perform the AWS CLI commands you intend to use. You can attach policies to IAM users granting specific permissions.

# Network Connectivity: Ensure that your local machine has network connectivity to AWS services. If you are behind a firewall, make sure that the necessary ports are open.

# Configure AWS CLI: After installation, you need to configure the AWS CLI with your access key pair and preferred default settings. You can do this by running the aws configure command and providing the required information.

# “aws configure”:

# AWS Access Key ID [None]: AKIAQWDZJTQCIR6F4KNL

# AWS Secret Access Key [None]: 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

# Default region name [None]:

# Default output format [None]: json

# Cross check: Run command “aws s3 ls”

# Output: 2023-11-28 10:12:57 cf-templates-y05vltwidc3s-eu-north-1

# aws ec2 run-instances --image-id ami-0ecdd20ff07e019ed --instance-type t2.micro --key-name AWS\_Test --subnet-id subnet-02a515f63d4834616 --security-group-ids sg-0f23a5e7dfeaf868d --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=aws\_cli}]'

# Example: aws ec2 run-instances --image-id ami-0ecdd20ff07e019ed --instance-type t2.micro --key-name AWS\_Test --subnet-id subnet-02a515f63d4834616 --security-group-ids sg-0f23a5e7dfeaf868

# Output: So that you can understand

{

    "Groups": [],

    "Instances": [

        {

            "AmiLaunchIndex": 0,

            "ImageId": "ami-0ecdd20ff07e019ed",

            "InstanceId": "i-0eb464bd3e9b3958d",

            "InstanceType": "t2.micro",

            "KeyName": "AWS\_Test",

            "LaunchTime": "2024-01-08T10:48:04+00:00",

            "Monitoring": {

                "State": "disabled"

            },

            "Placement": {

                "AvailabilityZone": "ap-south-1b",

                "GroupName": "",

                "Tenancy": "default"

            },

            "Platform": "windows",

            "PrivateDnsName": "ip-172-31-14-70.ap-south-1.compute.internal",

            "PrivateIpAddress": "172.31.14.70",

            "ProductCodes": [],

            "PublicDnsName": "",

            "State": {

                "Code": 0,

                "Name": "pending"

            },

            "StateTransitionReason": "",

            "SubnetId": "subnet-02a515f63d4834616",

            "VpcId": "vpc-01a91952be286ef22",

            "Architecture": "x86\_64",

            "BlockDeviceMappings": [],

            "ClientToken": "00cd359e-5139-4469-b266-1c3b445ff322",

            "EbsOptimized": false,

            "EnaSupport": true,

            "Hypervisor": "xen",

            "NetworkInterfaces": [

# To create VPC (all configuration), Load balancer in such cases AWS CLI is not so handy.

# To create complex stack AWS CLI is not recommended.

# For quick references, simple tasks, to do things very fast it will be very helpful.

# Day-11 | IaC with AWS CFT | Tips and Tricks to Write CFT | CFT vs Terraform |

# You'll learn how to create CloudFormation templates to automate the provisioning of resources, manage stacks, and ensure consistent infrastructure across deployments.

# Project: You'll work on creating a CloudFormation template that provisions a fully configured application stack, including EC2 instances, networking components, and security groups.

# CFT: Cloud formation template-> It’s a template to create service on cloud/AWS. (Write code to create infrastructure)

# AWS CFT implements the principle of IAC (Infrastructure as Code) which AWS CLI will not do.

# IaC Principles: IaC tool must act as middle layer between user and one or multiple cloud providers (AWS, Azure etc.). If user submits a template (yml, json or anything i.e, declarative or versioned in nature submitted by user) and it should be converted to Cloud providers understandings. (i.e., API calls)

# CFT targets AWS does not support other cloud providers. And it supports yml or json.

# Versioned: Version control system

# Declarative: what u see is what u have

# Template: same should present in Aws infrastructure

# By looking u should understand these are the resources present in AWS.

# Features: 1. Creating infrastructure 2. It supports both JSON, YAML 3. Drift detection: Drift detection in AWS CloudFormation is a feature that helps you identify and visualize differences between the desired (as defined in the template) and actual stack resources.

# Example of Drift Detection: Let’s say you have created a architecture of your requirement and it uses some AWS services like EC2, S3 bucket(Versioned) and you have provisioned using CFT Template but someone from your team has disabled versioning of S3 bucket and you can enable drift detection periodically and it will run periodically and lets you know that some has manually changed the content(Disabling versioning) of AWS template in the AWS console.

# YAML is preferred for freshers and it will be helpful with k8s, ansible, Devops etc.

# Advantages of yml: 1. YAML supports commenting 2. YML is readable, whereas JSON depends on brackets, curly braces so it will be sometime to hard to read. 3. YAMl is less complex. 4. JSON does not support commenting.

# CFT: YAML templates – stacks

# How to write CFT:

# Version: (only one version for CFT) Description: (What exactly is this CFT) Metadata: (Author, which team, which project, Owner of CFT) Parameters: (Pass any variables during run time) Rules: (parameters or variables will be used different for different environments, Proper naming conventions) Mappings: (Assign parameters to variables) Conditions: (Provide some conditions, Ex: this template is only for Dev and Staging) Transform: (Ignore, serverless compute conditions) Resources: (\*\*very imp, it has to be provided, resources details) Output: (what output u want, like u just want specifically public Ip then it will provide)

# CloudFormation:

# Stack: Implements the template (We provide template as input and stack will convert template to API using cloud formation service)

# Use sample template: 1. Template is Ready 2. Instead use Template references. 3. Create template in Designer (2 plugins: YAML, AWS toolkit) 4. Create template in Designer: Drag and Drop tool,

# Specify Template:

# Template Source: a. Amazon S3 b. Upload a template file

# Specify Stack Details: Stack Name

# Configure Stack Options

# CloudFormation: Stacks: Go to your stack: Drift (It will show you if someone has changed something in AWS console instead of changing in the template)

# Plugins: YAMl and AWS Tool Kit

# Task1. Create S3 bucket with CFT and create stack.

Resources:

  S3Bucket:

    Type: 'AWS::S3::Bucket'

    Properties:

      BucketName: "demo.aws.ram.exmpl.com"

# Task2: Create S3 bucket with CFT and create stack with versioning so that we can test drift detection.

Resources:

  S3Bucket:

    Type: 'AWS::S3::Bucket'

    Properties:

      BucketName: "demo.aws.ram.exmpl.com"

      VersioningConfiguration:

        Status: Enabled

# Go and manually disable the versioning of created bucket and in your stack check for drift detection and it will show if someone changes something manually.

# Task3: Creat EC2 instance using cloudformation template (ec2.yml)

AWSTemplateFormatVersion: '2010-09-09'

Description: 'EC2 Instance Example'

Resources:

  MyEC2Instance:

    Type: 'AWS::EC2::Instance'

    Properties:

      ImageId: 'ami-0ecdd20ff07e019ed'  # Replace with your desired AMI ID

      InstanceType: 't2.micro'

      KeyName: 'AWS\_Test'  # Replace with your key pair name

      SecurityGroupIds:

        - 'sg-0f23a5e7dfeaf868d'  # Replace with your security group ID

      SubnetId: 'subnet-02a515f63d4834616'  # Replace with your subnet ID

      Tags:

        - Key: Name

          Value: Ram10  # Replace with the desired name for your instance

# Step1: Cloudformation-Template is ready-upload a template file-Next

# Step2: Stack name: ram1, parameters: no input

**Step3:** Tags: no input, Permissions: IAM role: if you want to use some role instead of root account,   
 **Stack failure option:** Behavior on provisioning failure

* Specify the roll back behavior for a stack failure. [Learn more](https://docs.aws.amazon.com/console/cloudformation/stack-failure-options)

1. Roll back all stack resourcesRoll back the stack to the last known stable state.
2. Preserve successfully provisioned resourcesPreserves the state of successfully provisioned resources, while rolling back failed resources to the last known stable state. Resources without a last known stable state will be deleted upon the next stack operation.

* Delete newly created resources during a rollback

Specify whether resources that were created during a failed operation should be deleted regardless of their deletion policy. [Learn more](https://docs.aws.amazon.com/console/cloudformation/stack-failure-options)

1. Use deletion policyRetains or deletes created resources according to their attached deletion policy.
2. Delete all newly created resourcesDeletes created resources during a rollback regardless of their attached deletion policy.

* Advanced options

You can set additional options for your stack, like notification options and a stack policy.[Learn more](https://docs.aws.amazon.com/console/cloudformation/stackoptions)

* + 1. Stack policy: Defines the resources that you want to protect from unintentional updates during a stack update.
    2. Rollback configuration: Specify alarms for CloudFormation to monitor when creating and updating the stack. If the operation breaches an alarm threshold, CloudFormation rolls it back.
    3. Notification options: SNS topic
    4. Stack creation options: Timeout, (Deactivated and Activated)

# Day-12 | AWS CICD | AWS Code Commit | What is CICD on AWS? |

# Implement CICD on AWS: AWS provides a comprehensive set of CI/CD services that enables developers to automate and streamline their software delivery processes.

# AWS Code Pipeline, AWS Code Build and AWS Code Deploy are the key services involved in achieving CI/CD on AWS platform.

# AWS Code commit: Version control system (GitHub)

# AWS Code pipeline: (Instead of Jenkins for pipeline we can use it)

# AWS Cod Build: (To build your code instead of Maven)

# AWS Code Deploy: To deploy instead of K8s, EC2.

# AWS Code Commit: Managed AWS service instead of GitHub. (Pay to use it and in all organization, they use paid GitHub) Why: To manage private repositories as most of the projects will not have public repositories. - Advantages: Solves the problem of managed git solutions. Install git software on servers, it will be solved here. a. Managed Git b. Scalability c. Reliability.

# -Disadvantages: a. Less features. (GitHub: Code pilot, VS code integration, Gitlab: Auto Devops), Advanced tool collaboration b. AWS Restricted c. Less integrations with services outside AWS - Does not add more value if you use AWS Code commit we can go for GitHub, GitLab and BitBucket.

# AWS Code Commit: Store code in Private Git Repositories.

# Create Repositories – demo-repo-cc – (Enable Amazon Code guru: don’t enable) – Create.

# Through AWS UI we can upload, download, and edit only one file at a time so we need VS code or terminal.

# Options: Code, Pull Request, Commits, Git tags, Settings

# How IAM plays important role in AWS Commit: Create user and attach code-commit power user policy

# Day-13 | AWS Code Pipeline | Jenkins vs AWS Code Pipeline | Open-Source vs AWS Managed |

# Jenkins implements Continuous Integration, and it invokes continuous delivery. Code Check out – Build & UT – Code Scan – Image Build – Image Scan – Image push.

# 

# AWS Code Pipeline: Invoking CI & CD.

# AWS Code Build is responsible for Continuous integration, but even you can use some scripts and do it.

# 

# AWS code commit is very less used. Instead, they use GitHub or GitLab.

# Why Code Pipeline: Managed CI CD pipeline, but why Jenkins is preferred? Jenkins can be created in EC2 instance, and we can configure Autoscaling, AWS Cloud watch and AWS Alarms that way even using AWS we can manage Jenkins on AWS. By doing this we can avoid AWS Managed service.

# Major Drawback: Very much restricted to AWS (If client moves to hybrid or Azure clous then all our pipeline will be of no use)

# If our code is in Jenkins and then we can delete the Jenkins instance and take backup and use those pipelines from Git repository and use them.

# Why Jenkins over AWS managed pipeline: 1. Features of Jenkins 2. Integration of Jenkins: 3. Not restricted to AWS platform only

# Why AWS over Jenkins: 1. Everything is managed by AWS 2. If cost is not a barrier for me then even scaling is not a problem for AWS.

# Day-14 | AWS END TO END CI | REALTIME PROJECT | DEMO WITH NOTES |

# Instead of AWS Code Commit we will use GitHub.

# Already we have repository in GitHub. <https://github.com/rameshdn105/aws-devops-zero-to-hero/tree/main/day-14/simple-python-app>

# Buildspec.yml: For building the file.

# Appspec.yml: code deploy file, inside GitHub repository where source code is available.

# App.py: app.py file could be used as the main script that orchestrates the functionality of the entire application. - Web Applications: In web development using frameworks like Flask or Django, developers often create an app.py file to define the main application instance, routes, and other configurations. For example, in Flask, the app.py file might contain code to create a Flask application instance and define routes for handling HTTP requests.

# Docker file: docker file to create an image application.

# Requirement.txt: a common convention in Python projects to declare the dependencies that your project needs to run.

# Start\_container.sh: add script for docker image pull and docker run.

# Stop\_container.sh: stop running container (no container running in my case given echo “hi”)

# AWS Code Build as a build service as it is a managed service AWS will provide you images for managed service.

# Step1: GitHub (Repository) Step2: AWS Code Build - Create an IAM user and grant required access - Code Build – Build Project – Create build project Project name: Description: Enable badge: Enable concurrent build limit: at a time how many commits and build can happen Source: Source Provider: Code commit or GitHub (If any Source provider is not supported by AWS Code Build (GitLab) then u can just create a mirroring of those code to GitHub and use them here) Repository: Public repository or Repository in my GitHub account Repository URL:

# Environment: Environmental Image: Managed or Custom image (VM image or docker image {Amazon linux, ubuntu}) Runtime: Standard Image: Ubuntu latest Image Version: Latest Environment type: Linux

# Service Role: (Role: Code build need access to perform work AWS) IAM-Roles-Creation of Role-AWS service-codebuild-Name-create Build Spec: () Build spec file: From Repo file or write their itself. Insert build commands: Put commands (YAMl etc) AWS provides examples, you just need to uncomment them. Build commands:

# version: 0.2 env: #variables: #key: “value” #key: “value” parameter-store: DOCKER\_REGISTRY\_USERNAME: /myapp/docker-credentials/username DOCKER\_REGISTRY\_PASSWORD: /myapp/docker-credentials/password DOCKER\_REGISTRY\_URL: /myapp/docker-credentials/url phases: install: runtime-versions: python: 3.11 (java, Golang etc) pre-Build: #pre-requisites for you build commands: - pip install -r day-14/sample-python-app/requirements.txt build: #checkout happens automatically bcz we have integrated GitHub already. commands: - cd day-14/sample-python-app - echo “Building Docker Image” - docker build -t “$DOCKER\_REGISTRY\_URL/$ DOCKER\_REGISTRY\_USERNAME/simple-python-flask-app:latest” . - docker push “$DOCKER\_REGISTRY\_URL/$ DOCKER\_REGISTRY\_USERNAME/simple-python-flask-app:latest” post\_build: commands: - echo “Build is successful” - Above is CI stage, in next class we can do continuous delivery. - Code Build is ready but we did not use Code Pipeline yet. How to store sensitive information: AWS System Manager: Create parameter: Name (app/what kind of credentials/username/) Description: Tier: Standard or Advanced Type: Secure string

# SECRET MANAGEMENT ON AWS: -> Docker hub, Ansible, Terraform, Database: all credentials should be managed secretly. -> AWS provided different secret managers like: 1. System Manager 2. Secrets Manager 3. HashiCorp Vault (Not a AWS offering)

# Systems manager: Parameter store -> Easy to retrieve (Provide IAM role to access it by AWS services) -> Simple to integrate.

# Secret Manager: Might have to rotate sensitive information. (When highly secured and you want it to change multiple time or periodically) Password rotation policy, add additional security -> Higher price Ex: Let’s say our certificate is expired then if we rotate them using public key then we can retrieve it. -> Password is DB related, we are using it in day-to-day activities. If we want to rotate it, then it keeps changing periodically.

# Based on sensitivity you can use one of the above. Ex: We have Docker username, URL and password Docker username: System manager Docker URL: System manager Docker password: Secrets manager

# HashiCorp Vault: AWS managed offering, we will be tied to AWS platform so when we use hybrid cloud then this plays a very important role. (Above solutions should not be a bottleneck for team) ->Open source, as it is a community driven project so they will give back up and provide many additional features and encryptions over AWS managed security.

# <app.yml>

# Day-15 | AWS ULTIMATE CICD PIPEPLINE | END TO END DEMO | AWS CODE PIPELINE |

# Code-Deploy: Fully managed deployment.

# Code deploy-create- (application name, compute platform)

# Create an instance to host your application using code deploy.

# Install agent inside EC2 instance. Code-Deploy agent (GitHub actions: agent installed on worker node) <https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent-operations-install-ubuntu.html> - follow steps in above link to install code-deploy agent on created EC2 instance. - Steps: connect to EC2 instance using Mobaxtreme or ssh -i ~/path of pem file.pem ubuntu@publicip sudo apt update sudo apt install ruby-full sudo apt install wget wget <https://bucket-name.s3.region-identifier.amazonaws.com/latest/install> - For example:

<https://aws-codedeploy-us-east-2.s3.us-east-2.amazonaws.com/latest/install>  
 chmod +x ./install  
 sudo ./install auto  
 systemctl status codedeploy-agent **-** If the CodeDeploy agent is installed and running, you should see a message like The AWS CodeDeploy agent is running.

* **CODE-Deploy agent is installed on EC2 instance.**

1. Provide required permission to EC2 instance with role to talk to the AWS Code-Deploy. (Add Code deploy full access to EC2 by creating a new role)
2. Restart Code Deploy-agent service. (sudo service codedeploy-agent restart)
3. Configure code deploy: Till now we have created application. We need to provide source code, type of application, how to execute, how to deploy (is it a docker image, shell script or html?) etc.  
   - Provide EC2 instance in code deploy as an agent.  
   - Target groups are called as deployment groups  
   - Provide deployment group name and service role: it should have both codedeploy and EC2Instance full access.  
   - Deployment type: In-place, Env confg: Amazon EC2 Instances  
   - Tag: Name, sample-python, if you provided same tag for 10 instances al will be used to deploy application.
4. Disable load balancer and create code deploy application.   
   Now registered application in EC2 instances and integrated between code deploy and EC2 instance.

# Code-deploy needs tags for identification of EC2 instances, we can provide multiple instances with same tags so that code-deploy identifies and uses them. - Tag: identification purpose, differentiate between diff project while doing cost optimization.

# Create instance with tags connect it using cli: ssh -i ~/path of pem file.pem ubuntu@ip sudo apt update

# Put appspec.yml in root level of repository.

# Go to GitHub/username/aws-zero-to-hero and click. it will convert GitHub to vs code so it will be helpful for modifying files.

# Code deploy: Applications: sample-python-app: Deployments: create - Created deployment by providing source and provided other details. - Deployment will fail as our code should be in root repository (https://github.com/rameshdn105/aws-devops-zero-to-hero) but instead my code is in sub folder(https://github.com/rameshdn105/aws-devops-zero-to-hero/tree/main/day-14/simple-python-app) of root repository.

# Day-16 | AWS CLOUD WATCH DEEP DIVE | DEMO - LIVE EC2 CPU ALERTING THROUGH SNS |

* What is AWS CloudWatch?

AWS CloudWatch is a powerful monitoring and observability service provided by Amazon Web Services. It enables you to gain insights into the performance, health, and operational aspects of your AWS resources and applications. CloudWatch collects and tracks metrics, collects and monitors log files, and sets alarms to alert you on certain conditions.

**Advantages of AWS CloudWatch:**  
1. Comprehensive Monitoring: CloudWatch allows you to monitor various AWS resources such as EC2 instances, RDS databases, Lambda functions, and more. You get a unified view of your entire AWS infrastructure.

1. Real-Time Metrics: It provides real-time monitoring of metrics, allowing you to respond quickly to any issues or anomalies that might arise.
2. Automated Actions: With CloudWatch Alarms, you can set up automated actions like triggering an Auto Scaling group to scale in or out based on certain conditions.
3. Log Insights: CloudWatch Insights lets you analyze and search log data from various AWS services, making it easier to troubleshoot problems and identify trends.
4. Dashboards and Visualization: Create custom dashboards to visualize your application and infrastructure metrics in one place, making it easier to understand the overall health of your system.

**Problem Solving with AWS CloudWatch:** CloudWatch helps address several critical challenges, including:

1. Resource Utilization: Tracking resource utilization and performance metrics to optimize your AWS infrastructure efficiently.
2. Proactive Monitoring: Identifying and resolving issues before they impact your applications or users.
3. Troubleshooting: Analyzing logs and metrics to troubleshoot problems and reduce downtime.
4. Scalability: Automatically scaling resources based on demand to ensure optimal performance and cost efficiency.

**Practical Use Cases of AWS CloudWatch:**  
Auto Scaling: CloudWatch can trigger Auto Scaling actions based on defined thresholds. For example, you can automatically scale in or out based on CPU utilization or request counts.

1. Resource Monitoring: Monitor EC2 instances, RDS databases, DynamoDB tables, and other AWS resources to gain insights into their performance and health.
2. Application Insights: Track application-specific metrics to monitor the performance of your applications and identify potential bottlenecks.
3. Log Analysis: Use CloudWatch Logs Insights to analyze log data, identify patterns, and troubleshoot issues in real-time.
4. Billing and Cost Monitoring: CloudWatch can help you monitor your AWS billing and usage patterns, enabling you to optimize costs.

# Day-17 | AWS LAMBDA INTRODUCTION | HOW DEVOPS ENGINEERS USE SERVERLESS ARCHITECTURE ?|

# -> Introduction to Serverless Computing

Today, we're going to embark on an exciting journey into the world of serverless computing and explore AWS Lambda, a powerful service offered by Amazon Web Services.

So, what exactly is "serverless computing"? Don't worry; it's not about eliminating servers altogether. Instead, serverless computing is a cloud computing execution model where you, as a developer, don't have to manage servers directly. You focus solely on writing and deploying your code, while the cloud provider takes care of all the underlying infrastructure.

* **Lambda (FaaS: Functions as a Service)**: Compute and Serverless  
  1. Cost optimization (By deleting unwanted things if you have written scripts to do it) (Devops engineer, Security)  
  2. Services governance (Monitor and alert)  
  3. Security and Compliance (No one has to create gp2 EBS volume as it has security issue, we can create lambda to search EBS volume with GP2 and it should notify us. Or someone created S3 bucket with public Ip).
* **Task1:** Go to Lambda – Author from scratch – name -runtime – programming language (.net 6, go 1.x, Java, Python, Ruby) – Advanced settings (Enable function URL {u will get public ip address to access application}) – People with access can access it – create – provide python function – lambda handler is name
* Controlling lambda function: Environment variables,

## **Understanding AWS Lambda**

In this serverless landscape, AWS Lambda shines as a leading service. AWS Lambda is a compute service that lets you run your code in response to events without the need to provision or manage servers. It automatically scales your applications based on incoming requests, so you don't have to worry about capacity planning or dealing with server maintenance.

## **How Lambda Functions Fit into the Serverless World**

At the heart of AWS Lambda are "Lambda functions." These are individual units of code that perform specific tasks. Think of them as small, single-purpose applications that run independently.

Here's how Lambda functions fit into the serverless world:

**Event-Driven Execution**: Lambda functions are triggered by events. An event could be anything, like a new file being uploaded to Amazon S3, a request hitting an API, or a specific time on the clock. When an event occurs, Lambda executes the corresponding function.

**No Server Management**: As a developer, you don't need to worry about managing servers. AWS handles everything behind the scenes. You just upload your code, configure the trigger, and Lambda takes care of the rest.

**Automatic Scaling**: Whether you have one user or one million users, Lambda scales automatically. Each function instance runs independently, ensuring that your application can handle any level of incoming traffic without manual intervention.

**Pay-per-Use**: One of the most attractive features of serverless computing is cost efficiency. With Lambda, you pay only for the compute time your code consumes. When your code isn't running, you're not charged.

**Supported Languages**: Lambda supports multiple programming languages like Node.js, Python, Java, Go, and more. You can choose the language you are comfortable with or that best fits your application's needs.

Real-World Use Cases

Now, let's explore some real-world use cases to better understand how AWS Lambda can be applied:

**Automated Image Processing**: Imagine you have a photo-sharing app, and users upload images every day. You can use Lambda to automatically resize or compress these images as soon as they are uploaded to S3.

**Chatbots and Virtual Assistants**: Build interactive chatbots or voice-controlled virtual assistants using Lambda. These assistants can perform tasks like answering questions, fetching data, or even controlling smart home devices.

**Scheduled Data Backups**: Use Lambda to create scheduled tasks for backing up data from one storage location to another, ensuring data resilience and disaster recovery.

**Real-Time Analytics**: Lambda can process streaming data from IoT devices, social media, or other sources, allowing you to perform real-time analytics and gain insights instantly.

**API Backends**: Develop scalable API backends for web and mobile applications using Lambda. It automatically handles the incoming API requests and executes the corresponding functions.

**Day-18 | AWS Cost Optimization | Most Popular Cloud and DevOps project| Event Driven Serverless**

1. Reduce overhead of Infrastructure  
2. Optimize infrastructure cost!

Default execution time for lambda: 5sec (u can modify as well)

* Cloud cost goes down only if you use efficiently.  
  Ex: a. Developer created EC2 instance and attached a volume to it, and volume has very sensitive info and he took backup of volume by snapshots of it every day. Now he has deleted EC2 but he forgot to delete volume and snapshot he created everyday which will increase the cost.  
  b. S3 bucket: If you go on dumping your content into S3 bucket.
* Devops engineer should understand which are stale resource and he should send out notifications about them.

# AWS Cloud Cost Optimization - Identifying Stale Resources

## **Identifying Stale EBS Snapshots:** In this example, we'll create a Lambda function that identifies EBS snapshots that are no longer associated with any active EC2 instance and deletes them to save on storage costs.

### **Description:**

The Lambda function fetches all EBS snapshots owned by the same account ('self') and also retrieves a list of active EC2 instances (running and stopped). For each snapshot, it checks if the associated volume (if exists) is not associated with any active instance. If it finds a stale snapshot, it deletes it, effectively optimizing storage costs.

**Task1. EC2 instances created and while deleting EC2 he forgot to delete snapshots. (Write a Lambda function to notify unattached EBS snapshots)**

Step1: Lambda: Python  
Step2: List all EBS snapshot  
Step3: Filter out snapshots which are stale  
Step4:c Delete filtered snapshots

Code copied from bot3 or chatgpt  
Create instance-create snapshot of volume attached to instance  
Run the lambda function: it fails as it doesn’t have enough permissions to run the function  
**Add required permission to a custom role- create policy -add “describe instances, describe volumes, describe volumes, delete snapshot”**

# Day-19 | AWS CLOUD FRONT | Ft: Piyush | Content Delivery Network |

* Managed AWS service to provide solution for CDN (Content Delivery network)  
  Ex: YouTube, Snapchat, Instagram, Flipkart, Amazon etc.
* **CDN solves the problem by keeping multiple local copies to avoid high latency by keeping nearest edge locations.**
* CDN helps enhancing the lowest possible latency as content is cached in nearest edge location capability.
* CDN helps even in cost as no need to upload and download content in S3 buckets.
* CDN improves Security as user is not accessing application present in S3 bucket.

# Comprehensive Guide to CDN and CloudFront on AWS for Beginners

If you've never heard of CDN or CloudFront before, don't worry. we'll start from scratch and gradually build up your understanding. By the end, you'll be well-versed in these technologies. So let’s get started.

## 1. Introduction to Content Delivery Networks (CDN)

Imagine you have a website with lots of cool content, like images, videos, and documents. When a user visits your site from a different location far away from your server, the content might take a long time to load. That's where CDN comes to the rescue!

A CDN is like a network of servers spread across various locations worldwide. These servers store a copy of your website's content. When a user requests your website, the content is delivered from the server closest to the user, making it super fast! It's like having a local store for your website content everywhere in the world.

## 2. What is CloudFront?

CloudFront is Amazon Web Services' (AWS) very own CDN service. It integrates seamlessly with other AWS services and allows you to deliver content, videos, applications, and APIs securely with low-latency and high transfer speeds.

## 3. How Does CloudFront Work?

Let's understand how CloudFront works with a simple example:

Imagine you have a website with images stored on an Amazon S3 bucket (a cloud storage service). When a user requests an image, the request goes to CloudFront first.

Here's how the process flows:

* **Step 1**: CloudFront checks if it already has the requested image in its cache (storage). If it does, great! It sends the image directly to the user. If not, it proceeds to Step 2.
* **Step 2**: CloudFront fetches the image from the S3 bucket and stores a copy in its cache for future requests. Then, it sends the image to the user.

The next time someone requests the same image, CloudFront will deliver it from its cache, making it super-fast and efficient!

## 4. Benefits of CloudFront

* **Fast Content Delivery**: CloudFront ensures your content reaches users with minimal delay, making your website lightning fast.
* **Global Reach**: With servers in various locations worldwide, CloudFront brings your content closer to users, regardless of where they are.
* **Security**: CloudFront provides security features like DDoS protection and SSL/TLS encryption to keep your content and users safe.
* **Scalability**: CloudFront can handle traffic spikes effortlessly, ensuring a smooth experience for your users.
* **Cost-Effective**: Pay only for the data transfer and requests made, making it cost-effective for businesses of all sizes.

## 5. Setting Up CloudFront on AWS Now, let's get our hands dirty and set up CloudFront on AWS!

### Step 1: Create an S3 Bucket

1. Go to the AWS Management Console and navigate to Amazon S3.
2. Create a new bucket to store your website content.

### Step 2: Upload Content to the S3 Bucket

1. Upload images, videos, or any other content you want to serve through CloudFront to your S3 bucket.

### Step 3: Create a CloudFront Distribution

1. Go to the AWS Management Console and navigate to CloudFront.
2. Click "Create Distribution."
3. Choose whether you want to deliver a web application or content (like images and videos).
4. Configure your settings, such as the origin (your S3 bucket), cache behaviors, and security settings.
5. Click "Create Distribution" to set up CloudFront.

### Step 4: Update Website URLs

1. Once your CloudFront distribution is deployed (it may take a few minutes), you'll get a CloudFront domain name (e.g., d1a2b3c4def.cloudfront.net).
2. Replace the URLs of your website content with the CloudFront domain name.

That's it! Your content is now being delivered through CloudFront.

## 6. Use Cases and Scenarios

### Scenario 1: E-Commerce Website

Let's say you have an e-commerce website that sells products globally. By using CloudFront, your product images and videos load quickly for customers all over the world, improving the shopping experience.

### Scenario 2: Media Streaming

You're running a video streaming platform. With CloudFront, you can stream videos to users efficiently, regardless of their location, without buffering issues.

### Scenario 3: Software Downloads

If you offer software downloads, CloudFront can distribute your files faster, reducing download times and providing a better user experience.

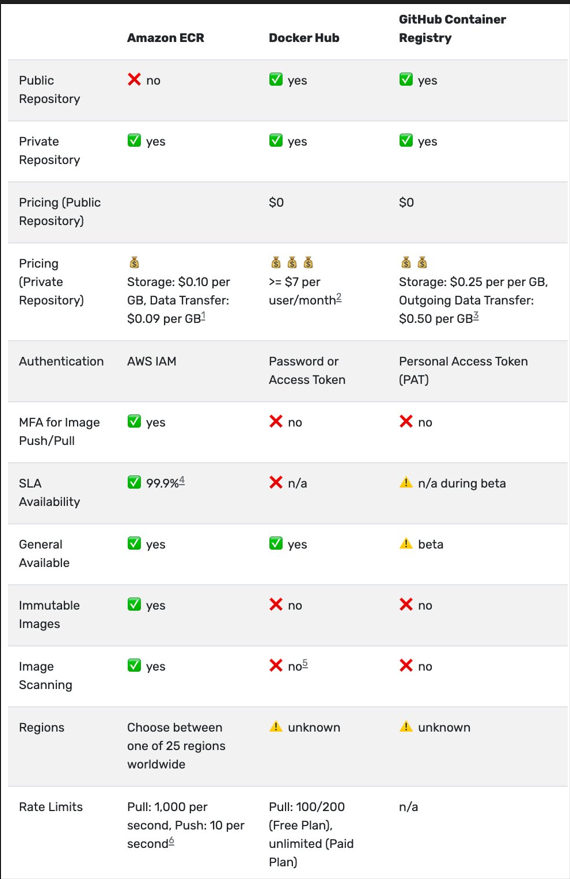
## 7. Tips and Best Practices

* **Caching Strategies**: Configure cache settings wisely to balance freshness and speed for different types of content.
* **Invalidation**: Learn how to invalidate or clear cached content when you make updates to your website.
* **Monitoring and Reporting**: Use AWS tools to monitor your CloudFront distribution's performance and gain insights into user behavior.

## 8. Conclusion

By using CloudFront, you can dramatically improve your website's performance, making users happier and potentially boosting your application and business.

# Day-20 | AWS ECR | ECR vs Docker Hub | Free AWS job ready course |

* ECR – AWS service to store and manage Containers. (Container Registry)  
  - **ECR: Elastic, Container and Registry (Elastic: Highly scalable and available in nature)**
* **Container** is a package contains application, software, and dependencies to run the application.
* Container registry is used to store docker images o anyone from the world can access it.
* Docker Hub: Free, Public Repository, Private Repositories (Only authorized people), If DockerHub is down, no direct support when compared to AWS.
* ECR: Private Repository concept, Public also we can create but by default it will create Private Repositories, Security (IAM), If ECR is down AWS will support immediately, good integration with other services like EKS, ECS, far gate etc.
* 

## 1. What is AWS ECR?

AWS Elastic Container Registry (ECR) is a fully managed container image registry service provided by Amazon Web Services (AWS). It enables you to store, manage, and deploy container images (Docker images) securely, making it an essential component of your containerized application development workflow. ECR integrates seamlessly with other AWS services like Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS).

## 2. Key Benefits of ECR

* **Security**: ECR offers encryption at rest, and images are stored in private repositories by default, ensuring the security of your container images.
* **Integration**: ECR integrates smoothly with AWS services like ECS and EKS, simplifying the deployment process.
* **Scalability**: As a managed service, ECR automatically scales to meet the demands of your container image storage.
* **Availability**: ECR guarantees high availability, reducing the risk of image unavailability during critical times.
* **Lifecycle Policies**: You can define lifecycle policies to automate the cleanup of unused or old container images, helping you save on storage costs.

## 3. Getting Started with AWS ECR

### Creating an ECR Repository

1. Go to the AWS Management Console and navigate to the Amazon ECR service.
2. Click on "Create repository" to create a new repository.
3. Enter a unique name for your repository and click "Create repository."

### Installing AWS CLI

To interact with ECR from your local machine, you'll need to have the AWS Command Line Interface (CLI) installed. Follow the instructions in the [AWS CLI User Guide](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html) to install it.

### Configuring AWS CLI

After installing the AWS CLI, open a terminal and run the following command to configure your CLI with your AWS credentials:

aws configure

Enter your AWS Access Key ID, Secret Access Key, default region, and preferred output format when prompted.

## 4. Pushing Docker Images to ECR

Now that you have your ECR repository set up and the AWS CLI configured, let's push a Docker image to ECR.

1. Build your Docker image locally using the docker build command:

docker build -t <your-image-name> <path-to-dockerfile>

1. Tag the image with your ECR repository URI:

docker tag <your-image-name>:<tag> <your-aws-account-id>.dkr.ecr.<your-region>.amazonaws.com/<your-repository-name>:<tag>

1. Log in to your ECR registry using the AWS CLI:

aws ecr get-login-password --region <your-region> | docker login --username AWS --password-stdin <your-aws-account-id>.dkr.ecr.<your-region>.amazonaws.com

1. Push the Docker image to ECR:

docker push <your-aws-account-id>.dkr.ecr.<your-region>.amazonaws.com/<your-repository-name>:<tag>

## 5. Pulling Docker Images from ECR

To pull and use the Docker images from ECR on another system or AWS service, follow these steps:

1. Log in to ECR using the AWS CLI as shown in Step 3 of the previous section.
2. Pull the Docker image from ECR:

docker pull <your-aws-account-id>.dkr.ecr.<your-region>.amazonaws.com/<your-repository-name>:<tag>

## 6. Cleaning Up Resources

As good practice, remember to clean up resources that you no longer need to avoid unnecessary costs. To delete an ECR repository:

1. Make sure there are no images in the repository, or delete the images using docker rmi locally.
2. Go to the AWS Management Console, navigate to the Amazon ECR service, and select your repository.
3. Click on "Delete" and confirm the action.

# Day-21 | AWS ECS | ECS vs EKS vs Kubernetes | Free AWS job ready course|

* ECS: Amazon Elastic Container Service, fully managed container orchestration service that helps you to deploy, manage, and scale containerized applications more efficiently.
* **AWS Fargate (CAAS: Container As a Service)** is a technology that you can use with Amazon ECS to run containers without having to manage servers or clusters of Amazon EC2 instances. With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers.
* **EC2 Vs ECS Fargate Vs ECS Vs Lambda**

1. **Fargate** is a managed compute layer by AWS: They manage the underlying host to run the containers into for you.   
   - Automates the provisioning, management, and scaling of the compute resources that power containers managed by Amazon Elastic Compute Service (ECS) and Amazon Elastic Kubernetes Service (EKS).
2. **EC2** instances are managed and deployed by yourself, so it is up to you to deal with maintaining these.   
   - They both have pros and cons, and I personally much prefer to use Fargate as it is a lot less overhead management, "I just want to have containers running".
3. **ECS** you can create capacity providers, which is ECS driving an AutoScaling group so that you don't have to do this. It will provision EC2 instances for you based on the number of services that need to be running.
4. **Lambda** runs standalone, time limited functions in response to events and with no infrastructure to manage.

* We have 1000s of containers running on top of Fargate and not having to manage any EC2 hosts is worth the additional costs. Also not having access to the underlying host is, to me, a great bonus security-wise as people do not connect and hack their way into doing things on the hosts they are not supposed to be doing. It's also a lot less worry to patch, update the OS, etc.
* Docker problems: Auto healing and Autoscaling
* **ECS Disadvantages:**   
  1. (Task Define, Tasks, Services, Clusters) Not an opensource platform, AWS restricted, AWS services.  
  2. You cannot move pods, services, deployments, Ingress resources 80%-90% resources can be moved and used again if we use K8s, but AWS does not allow this flexibility.  
  3. ECS does not have all features of K8s like: Custom resource definitions (CRD’s), Configure lot of Ingress rules/controllers (load balancer, advanced web application firewalls, advanced security features, advanced secret management features etc.)
* AWS ECS Advantages: If you set up large no. of services or resources on AWS then it is big decision to move all to other cloud or platform.
* **In AWS we can create K8s in multiple ways:  
  a. Create Instances and install K8s on top them.  
  b. EKS   
  c. ECS.**
* **Structure of ECS:**- Create a cluster –Go with Fargate or run EC2 instances.  
  - Task definition: How does your container look like Ex: Pod in K8s  
  - Services: using this services we can add Load balancing capacity, create ingress policy with Application load balancer in AWS platform.   
  - ECS Advantage: Use in built IAM from for Authentication.
* **ECS Project:**

1. **Step1: Create culture in ECS  
   -** Create cluster: demo-ecs-cluster (In back end it will trigger CloudFormation and it creates his stack)
2. **Step2:** **Build and Push your docker image to ECR**  
   - Login to ECR (replace and with your actual values)$ aws ecr get-login-password --region | docker login --username AWS --password-stdin .dkr.ecr..amazonaws.com  
   - Build the Docker image (replace with your ECR repository name)$ docker build -t .dkr.ecr..amazonaws.com/:latest .  
   - Push the Docker image to ECR (replace with your ECR repository name)$ docker push .dkr.ecr..amazonaws.com/:latest
3. **Create Task definition in AWS ECS  
   -** Create definition (May be using CLI or UI of AWS)  
   1. Name: demo-ecs-example **2.** Infra requirement: AWS Fargate, OS: Linux/86\_64, Task size: CPU:1 vCPU 2GB,  
   3. Task Role: It is a task IAM role allows containers in the task to make API requests to AWS services. We can create a task IAM role from the IAM console.  
   4. Task execution role: It is an execution IAM role is used by the container to make API requests on your behalf. If you don’t already have a task execution IAM role created, we can create one for you.  
   5. Contianer-1   
   - Container details: name, Image URI, Essential container  
   - No change after that  
   - Container port: 3000, TCP  
   6. Use log collection: Cloud watch  
   7. Storage  
   8. Monitoring  
   Create it. Task definition is active.
4. **After creating task once it is in active status  
   -** Task definitions: Run task – provide cluster here created earlier - create  
   - Task will run our container  
   - Go to cluster created and it will have task created it in.

## Introduction

In the ever-evolving world of cloud computing, containerization has emerged as a pivotal technology, enabling developers to package their applications along with all dependencies into a single, portable unit. Amazon Elastic Container Service (ECS), a fully managed container orchestration service from AWS, simplifies the deployment, management, and scaling of containerized applications.

This blog post aims to be your ultimate guide to AWS ECS. We'll start from the fundamentals and gradually delve into the comparisons with its alternatives. We'll also discuss the pros and cons of ECS, provide step-by-step instructions for installation and configuration, and finally, guide you through deploying your first application on ECS.

## Table of Contents

1. What is AWS ECS?
2. Why Choose ECS Over Other Container Orchestration Tools?
3. ECS Fundamentals
   * Clusters
   * Task Definitions
   * Tasks
   * Services
4. Pros of Using AWS ECS
5. Cons of Using AWS ECS
6. Installation and Configuration
   * Prerequisites
   * Setting Up ECS CLI
   * Configuring AWS Credentials
7. Deploying Your First Application on ECS
   * Preparing the Application
   * Creating a Task Definition
   * Configuring the Service
   * Deploying the Service
   * Monitoring the Service
8. Conclusion

## 1. What is AWS ECS?

AWS ECS is a fully managed container orchestration service that allows you to run Docker containers at scale. It eliminates the need to manage your own container orchestration infrastructure and provides a highly scalable, reliable, and secure environment for deploying and managing your applications.

## 2. Why Choose ECS Over Other Container Orchestration Tools?

Before diving deep into ECS, let's compare it with some popular alternatives like Kubernetes and Docker Swarm.

### Comparison with Kubernetes:

Kubernetes is undoubtedly a powerful container orchestration tool with a vast ecosystem, but it comes with a steeper learning curve. ECS, on the other hand, offers a more straightforward setup and is tightly integrated with other AWS services, making it a preferred choice for AWS-centric environments.

### Comparison with Docker Swarm:

Docker Swarm is relatively easy to set up and is suitable for small to medium-scale deployments. However, as your application grows, ECS outshines Docker Swarm in terms of scalability, reliability, and seamless integration with AWS features like IAM roles and CloudWatch.

## 3. ECS Fundamentals

To understand ECS better, let's explore its core components:

### Clusters:

A cluster is a logical grouping of EC2 instances or Fargate tasks on which you run your containers. It acts as the foundation of ECS, where you can deploy your services.

### Task Definitions:

Task Definitions define how your containers should run, including the Docker image to use, CPU and memory requirements, networking, and more. It is like a blueprint for your containers.

### Tasks: A task represents a single running instance of a task definition within a cluster. It could be a single container or multiple related containers that need to work together.

### Services: Services help you maintain a specified number of running tasks simultaneously, ensuring high availability and load balancing for your applications.

## 4. Pros of Using AWS ECS

* **Fully Managed Service**: AWS handles the underlying infrastructure, making it easier for you to focus on deploying and managing applications.
* **Seamless Integration**: ECS seamlessly integrates with other AWS services like IAM, CloudWatch, Load Balancers, and more.
* **Scalability**: With support for Auto Scaling, ECS can automatically adjust the number of tasks based on demand.
* **Cost-Effective**: You pay only for the AWS resources you use, and you can take advantage of cost optimization features.

## 5. Cons of Using AWS ECS

* **AWS-Centric**: If you have a multi-cloud strategy or already invested heavily in another cloud provider, ECS's tight integration with AWS might be a limitation.
* **Learning Curve for Advanced Features**: While basic usage is easy, utilizing more advanced features might require a deeper understanding.
* **Limited Flexibility**: Although ECS can run non-Docker workloads with EC2 launch types, it is primarily optimized for Docker containers.

## 6. Installation and Configuration

Let's get our hands dirty and set up AWS ECS step-by-step.

### Prerequisites:

* An AWS account with appropriate IAM permissions.
* The AWS CLI and ECS CLI installed on your local machine.

### Setting Up ECS CLI:

ECS CLI is a command-line tool that simplifies the process of creating and managing ECS resources.

$ ecs-cli configure --region <region> --access-key <access-key> --secret-key <secret-key> --cluster <cluster-name>

### Configuring AWS Credentials:

Ensure you have the necessary AWS credentials configured using aws configure command.

## 7. Deploying Your First Application on ECS

In this section, we'll deploy a simple web application using ECS.

### Preparing the Application:

1. Create a Dockerfile for your web application.
2. Build the Docker image and push it to Amazon ECR (Elastic Container Registry).

### Creating a Task Definition: Define the task using the ECS CLI or the AWS Management Console.

### Configuring the Service: Create an ECS service to manage the desired number of tasks and set up load balancing.

### Deploying the Service: Use the ECS CLI or the AWS Management Console to deploy the service.

### Monitoring the Service: Monitor your ECS service using AWS CloudWatch metrics and logs.

## 8. Conclusion

In conclusion, AWS ECS offers a robust and user-friendly platform for deploying and managing containerized applications. We covered the fundamentals of ECS, compared it with its alternatives, discussed its pros and cons, and walked through the installation, configuration, and deployment of a sample application.

# Day22: Kubernetes End to End project on EKS | EKS Install and app deploy with Ingress |

# EKS: Managed controlled plane (Master node) not managed data plane (Worker node). Helps with worker node.

# For worker node we can go for Fargate (AWS serverless compute)

# EKS will provide highly available master node.

# Elastic K8s Service, managed Kubernetes service that makes it easy for you to run Kubernetes on AWS and on-premises. Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications. Kubedm:

# Control Plane or Master Node (High availability go with 3 master node K8s cluster) - API, ETCD, Scheduler, Cloud confg, Controller.

# Data plane or Worker node (3 worker nodes) - CNI (controller network interface), Container Runtime (CR), DNS, Kube proxy

# This is tedious process and Error prone. Why to prefer Kubedm instead of Kops when it’s so tedious.

# Master went down.

# API server issue

# ETCD crashed.

# Certificate expired.

# Scheduler is not working We are responsible for handling these issues. As a Devops engineer we will have 100’s or 1000’s of clusters and we can t sit and troubleshoot these.

# Two ways to install K8s on AWS: 1. Create VM’s and use tools like Kops, Kubedm etc. 2. EKS (Increased significantly as managing K8s is difficult) 3. On-Premise: K8s cluster on data centers.

# Communications:

# Nodeport: only accessed by private Ip

# Load balancer: Too costly for more clusters.

# Ingress: Preferred compared to all. Ingress controller Ingress class

# K8s End to End:

Pre-requisites:   
**1. kubectl** – A command line tool for working with Kubernetes clusters. For more information, see Installing or updating kubectl.  
**2. eksctl** – A command line tool for working with EKS clusters that automates many individual tasks. For more information, see Installing or updating.  
**3. AWS CLI** – A command line tool for working with AWS services, including Amazon EKS. For more information, see Installing, updating, and uninstalling the AWS CLI in the AWS Command Line Interface User Guide. After installing the AWS CLI, we recommend that you also configure it. For more information, see Quick configuration with aws configure in the AWS Command Line Interface User Guide.

# -> Creating cluster is setup using CLI (15-20min): eksctl create --name demo-clusture –region –us-east-1 –fargate

# Aws eks update-kubeconfig –-name demo-cluster –region –us-east-1

# Deployment app pod using deployment: eksctl create fargateprofile \ --cluster demo-cluster \ --region us-east-1 \ --name alb-sample-app \ --namespace game-2048

# Deploy the deployment, service and ingress: kubectl apply -f https:…………………….yaml (file all required infor)

# Ingress controller: Once you add ingress controller on AWS EKS then only your ingress rules mentioned in yaml will communicate and it will give output.

# Create ingress: kubectl get -n game-2048 (with no address so we need ingress controller)

# Deploy ingress controller: Pre-requisite - IAM OIDC provider: ALB controller needs AWS resources to talk to eksctl utils associate-iam-oidc-provider –cluster demo-cluster –approve

# Setup ALB connector on ADDon

# To run all commands below u shoud have AWS CLI, kubctl CLI, EKSCTL and Helm installed in same directory (C:\Program Files\Amazon) and directory path should be added in 'path' (thisPC-properties-advanced settings-Advance-Envronment variables-path-newpath-C:\Program Files\Amazon)

# Main two files:

# 1. one is yaml file(name space creation, deployment, service, networking ingress) https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/examples/2048/2048\_full.yaml

# 2. Policies define in jason: https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/install/iam\_policy.json

# EKS cluster:

# 1. Created cluster demo-cluster: $ eksctl create cluster --name demo-cluster --region ap-south-1 –fargate 2. $ aws eks update-kubeconfig --name demo-cluster --region ap-south-1 - Added new context arn:aws:eks:ap-south-1:827813820069:cluster/demo-cluster to C:\Users\user\.kube\config

# 3. Create Fargate profile

# $ eksctl create fargateprofile \

# --cluster demo-cluster \

# --region us-east-1 \

# --name alb-sample-app \

# --namespace game-2048

# -> Along with system create frgate profile, we are creating above defined fargate profile i.e, alb-sample-app.

# 4. Deploy the deployment, service and Ingress: $ kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/examples/2048/2048\_full.yaml

# namespace/game-2048 created

# deployment.apps/deployment-2048 created

# service/service-2048 created (It has created ith no Externa Ip: that means anyone within AWS vpc they ca n communcate with it using noeport, but our goal is to make it accessible by outside)

# ingress.networking.k8s.io/ingress-2048 created (Ingress created with port but no addres, as their is no ingress controller or loablaancer so that outsder can use it to access our application)

# 5. configure IAM OIDC provider: before ALB cotroller(K8s pod) add on we should configure AWS IAM OIDC bcz ALB controller needs to access the application load blancer. (Pods will communicate to services using IAM OIDC)

# $ eksctl utils associate-iam-oidc-provider --cluster $cluster\_name --approve

# 6. How to setup alb add on

# -> AL B controller is nothing but it is a pod, which requires access to AWS services such as ALB (For this it as to talk to the aPI)

# $ curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/install/iam\_policy.json

# Provies all policies provie in above jsn policy: Followed by aLB controller docs if any change they will provide you.

# -> Create IAM Policy

# aws iam create-policy \

# --policy-name AWSLoadBalancerControllerIAMPolicy \

# --policy-document file://iam\_policy.json

# -> Create IAM Role: these service accounts will be used for our application depending on the requirement. (Accordingly they will provide roles)

# eksctl create iamserviceaccount \

# --cluster=<your-cluster-name> \

# --namespace=kube-system \

# --name=aws-load-balancer-controller \

# --role-name AmazonEKSLoadBalancerControllerRole \

# --attach-policy-arn=arn:aws:iam::<your-aws-account-id>:policy/AWSLoadBalancerControllerIAMPolicy \

# --approve

# 7. Deploy ALB controller:

# a. Add helm repo: $ helm repo add eks https://aws.github.io/eks-charts

# b. Update the repo: $ helm repo update eks

# c. Istall: $ helm install aws-load-balancer-controller eks/aws-load-balancer-controller \

# -n kube-system \

# --set clusterName=demo-cluster \

# --set serviceAccount.create=false \

# --set serviceAccount.name=aws-load-balancer-controller \

# --set region=ap-south-1 \

# --set vpcId=vpc-0819a9242fd603715

# -> AWS LB controlle create and it has atleast 2 replicas (2 replicas will be one in each availability zones and continuously watch for igress resources)

# -> $ kubectl get deployment -n kube-system aws-load-balancer-controller

# -> Ingress controlller has created load balancer by seeeing ingress resource

# 8. eksctl delete cluster --name demo-cluster --region us-east-1

# Overall: create pod, deployment, services, ingress, igress controller (Ingress controller is created one time)

# Day-30 | THREE-TIER ARCHITECTURE IMPLEMENTATION ON AWS

# <https://www.showwcase.com/article/35459/building-a-resilient-three-tier-architecture-on-aws-with-deploying-mern-stack-application>

# User – Front end (UI: product description) – Database (It was send by backend) – Database (Backend send infor back to Front end) – Front end

# 

# For simple application like calculator I don’t need database. For this I need Front end and Backend it is called as 2-Tier application.

# A diagram of a diagram

# Step1: We will create VPC (3 subnets, Sub1:Front end, Sub2:Backend, Sub3: Database)

# Sub1: Front end: ASG (puts multiple EC2 instances in multiple availability zones i.e., deploys application two AZ)

# User – Route 53 – CDN – ELB: ASG (EC2 in Sub1) – ALB: (ESG: sub2) – RDS (Primary database)

# THREE-TIER-ARCHITECTURE-MODEL