**AWS Certified Developer Associate**

Contents

[**** **AWS Accounts** 1](#_Toc186207277)

[**** **Courses** 1](#_Toc186207278)

[**** **Notes** 2](#_Toc186207279)

[ **AWS Accounts and IAM** 4](#_Toc186207280)

[ **AWS Command Line Interface (CLI)** 8](#_Toc186207281)

[ **Amazon VPC, EC2, and ELB** 10](#_Toc186207282)

[ **Tips to answer question** 12](#_Toc186207283)

[ **Summary** 13](#_Toc186207284)

* **AWS Accounts**

aws.amazon.com/free

Account Root User:

email = zarod2019@gmail.com

pwd = Serafines@2024

alias = zar-training

account id = 436488467655

IAM User:

user = Zarbio

pwd = Serafines@2024

alias = zar-training

IAM User:

user = Joe

pwd = Joe@2024

* **Courses**
* AWS Certified **Developer Associate** Exam Training DVA-C02

<https://tcsglobal.udemy.com/course/aws-certified-developer-associate-exam-training/learn/lecture/35900716#overview>

<https://digitalcloud.training/aws-developer-associate-resources/>

* Resources:

<https://digitalcloud.training/aws-certified-developer-course-downloads/>

code, folder aws-dva-code: <https://github.com/nealdct/aws-dva-code/tree/main>

**AWS+Certified+Developer+Associate+Slides.pdf**

* **Notes**
* To build and deploy cloud apps use:

AWS Software Development Kits (AWS SDKs for Python, .Net, and Java)

Command Line Interface (AWS CLI)

AWS Management Console

* Amazon Web Services (AWS):

For General cloud engineering (DevOps), enterprise applications, web solutions.

Used for Startups and Web Applications.

Market leader in cloud.

* Jobs:

AWS Certified Developer

* Certifications:

Amazon Web Services Cloud Practitioner – 2023

valid 3 years

cost $100

AWS Certified Solutions Architect – Associate: for general cloud knowledge

AWS Certified **Developer** – **Associate**: for development

AWS Certified DevOps Engineer – Professional: for DevOps roles

* **AWS Certified Developer – Associate Exam**

A close-up of a certificate

Description automatically generated

Validate at: <https://aws.amazon.com/verification>

Validation Number = ZFVG1131NFVEQJWQ

[**https://aws.amazon.com/certification/certified-developer-associate/**](https://aws.amazon.com/certification/certified-developer-associate/)

A screenshot of a survey

Description automatically generated

AWS-Certified-Developer-Associate\_Exam-Guide.pdf [**https://d1.awsstatic.com/training-and-certification/docs-dev-associate/AWS-Certified-Developer-Associate\_Exam-Guide.pdf**](https://d1.awsstatic.com/training-and-certification/docs-dev-associate/AWS-Certified-Developer-Associate_Exam-Guide.pdf)

Score of 100 - 1000, minimum passing score is 720

**A close up of text

Description automatically generated**

Last version is **DVA-C02** (57 services). Instead DVA-C01 (33 services).

* TDVA-C02 Exam:

Domain 1: Development with AWS Services

Develop code for applications hosted on AWS

Develop code for AWS Lambda

Use data stores in application development

Identify components and resources for security

Domain 2: Security

Implement authentication and/or authorization for applications and AWS services

Implement encryption by using AWS services

Manage sensitive data in application code

Domain 3: Deployment

Prepare application artifacts to be deployed to AWS

Test applications in development environment

Automate deployment testing

Deploy code by using AWS CI/CD services

Domain 4: Troubleshooting and Optimization  
 Assist in a root cause analysis  
 Instrument code for observability  
 Optimize applications by using AWS services and features

* AWS Service Names: <https://aws.amazon.com/certification/policies/general-information/#AWS_Service_Names>

AWS cheat sheets: <https://tutorialsdojo.com/aws-cheat-sheets/>

[**https://medium.com/@meghanaharishankara/how-to-get-aws-developer-associate-certified-in-just-5-weeks-26c022b4b142**](https://medium.com/@meghanaharishankara/how-to-get-aws-developer-associate-certified-in-just-5-weeks-26c022b4b142)

* Exams, courses, simulator:

**Jon Bonso’s practice exams**: <https://www.udemy.com/course/aws-certified-developer-associate-practice-exams-amazon-dva-c01/?couponCode=BFCPSALE24>

**AWS Certified Developer Associate Exam Training DVA-C02**

[**https://tcsglobal.udemy.com/course/aws-certified-developer-associate-exam-training/learn/lecture/35900716#overview**](https://tcsglobal.udemy.com/course/aws-certified-developer-associate-exam-training/learn/lecture/35900716#overview)

AWS Certified Developer Associate Practice Exams DVA-C02

[**https://tcsglobal.udemy.com/course/aws-developer-associate-practice-exams/learn/quiz/4852736#overview**](https://tcsglobal.udemy.com/course/aws-developer-associate-practice-exams/learn/quiz/4852736#overview)

<https://digitalcloud.training/aws-developer-associate-resources/>

Practice Exams | AWS Certified Developer Associate 2024

<https://tcsglobal.udemy.com/course/aws-certified-developer-associate-practice-tests-dva-c01/learn/quiz/4540356#overview>

AWS Certified Developer Associate DVA-C01 Exam Questions

<https://tcsglobal.udemy.com/course/simulado-amazon-aws-certified-developer-associate-2020/learn/quiz/4788484#overview>

* **AWS Accounts and IAM**
* AWS Free Tier account vs Sandbox

|  |  |
| --- | --- |
| AWS Free Tier | Sandbox / Challenge Labs |
| Create your own AWS free tier account | AWS account is hosted by a provider |
| Full control | Limited control |
| You’re responsible for bills, but we will operate in the free tier and set a billing alarm. | No cloud bills (no risk) |
| For Hands-On Lessons (HOL) | Scenario-based challenges. |

* AWS Account

Need:

Credit card

Dynamic email alias account: john {account alias 1} @gmail.com

Phone to receive SMS verification code

Account Root User: super account

* Create your AWS Free Tier Account:

12 months free

1. url: AWS Free Tier

aws.amazon.com/free

[https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=\*all&awsf.Free%20Tier%20Categories=\*all](https://aws.amazon.com/free/?all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all)

email = [zarod2019@gmail.com](mailto:zarod2019@gmail.com)

pwd = Serafines@2024

Sign in as a Root user

1. Sign in to the Console
2. Account Configuration and Create a Budget

Configure Account Alias: for IAM

Create Account Alias: ex: dct-tab-training, **zar-training** (created)

A screenshot of a web page

Description automatically generated

Enable access to billing for IAM users:

IAM Dashboard

A close up of a sign

Description automatically generated

Update billing preferences:

option Billing Preferences

A screenshot of a email

Description automatically generated

Create a budget and alarm:

option Budget, $5

User a template (simplified)

Monthly cost budget

Enter your budgeted amount ($): 5.00

option Cost Explorer

* AWS Identity and Access Management (IAM):

Used for **authentication** and **authorization**

Ways to manage AWS:

Console

Command Line Interface - CLI

API through SDKs

1. Authentication: login
2. **Authorization**: **allow** or **deny** access to resources

**Policy**: define what we are **allowed to do**

ex:

Performing API actions like run instances on EC2 that launches a virtual server

GetBucket: retrieves information about buckets

CreateUser: create a user in IAM

Core components of IAM:

Users

User groups: for adding users and applying permissions policies

Roles

Policy

IAM Users:

Best practice not to use the root user account, you must set a strong password and enable Multi-Factor Authentication (**MFA**), then we can create user accounts.

ARN (Amazon Resource Name):

A close up of a sign

Description automatically generated

* Creating IAM Users and Groups:

Create Group:

Policy: AdministratorAccess

Create User:

A screenshot of a computer

Description automatically generated

Because I don’t need single sign on in many cases.

**user = Zarbio**

**pwd = Serafines@2024**

**A screenshot of a login screen

Description automatically generated**

* IAM Authentication and MFA:

MFA:

Signing in with MFA requires an authentication code from an MFA device

Something you have: phone, token device

Something you are: retina scans, fingerprints

Used for Root account and individual IAM user accounts

* Setup Multi-Factor Authentication (MFA) for IAM User account:

2 types of users:

Root user

IAM user

Option User, tab Security credentials, Assign MFA device:

Authenticator app: Google Authenticator, AuthyPhone

* AWS Security Token Service - STS:

Provided temporary credentials.

Permissions Policy: allowed or denied to this specific entity

* Access Control Methods – RBAC & ABAC:

RBAC (Role-Based Access Control):

Job function policies = AWS managed policies

Policy documents can be pre-created by AWS

ABAC (Attribute-Based Access Control):

Using attributes, tags to define access to our resources

Permissions policy

Action = API action (RebootDBInstance, StartDBInstance, StopDBInstance)

* Switching IAM Roles:

Create an IAM role

user = Joe

pwd = Joe@2024

**EC2**:

Run **virtual servers in the Cloud** running Linux, Windows, Mac Os

Create Role:

AmazonEC2FullAccess: launch virtual servers in the cloud, permissions for load balancing EC2

Add permissions:

Users, Permissions policies, Add permissions, Create inline policy, JSON

{

"Version": "2012-10-17",

"Statement": [

{

"Sid": "Statement1",

"Effect": "Allow",

"Action": [],

"Resource": []

}

]

}

* **AWS Command Line Interface (CLI)**

Tools: AWS CLI, VS Code and Git

* Install the AWS Command Line Interface (CLI):

Install on a Linux/Windows instance

AWS CLI <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

In Windows open CMD:

commands:

aws

aws --version

aws s3 ls // don’t allowed

* Configure **Credentials** for the AWS CLI:

Configure a terminal for a user, to create an **access key** (long term credentials) for AWS CLI

Access key = Access key ID + Secret access key

In Windows Power Shell:

command:

aws configure

aws s3 ls // list buckets

A computer screen with white text

Description automatically generated

AKIAWLIF5OTDZ3R6FXUI

4P4Sb97Ct/ry6Nc21bVQgTp9Lv9SovQcnnLymz44

A black background with white text

Description automatically generated

Doesn’t exist any buckets, no errors

cat .aws/config // contents of the file

response:

region = us-east-1

cat .aws/credentials

response:

aws\_acess\_key\_id =

aws\_secret\_access\_key =

A screenshot of a computer

Description automatically generated

* Overview of Using the AWS CLI:

command:

aws help

aws ec2 help

aws ec2 describe-instances // show status of instances

aws s3 help

aws s3 mb s3://{bucket-name} // mb = make/create **bucket** (**container to upload files**)

aws s3 mb s3://mytestbucket43243jd33x

aws s3 ls // list buckets

Create and upload a file to a bucket:

command:

touch testfile.txt // linux, create a file

New-Item -Path "testfile.txt" -ItemType File -Force // windows, create a file

ls // list directories

aws s3 cp testfile.txt s3://{bucket-name} // cp = copy, upload file to a bucket

aws s3 ls s3://{bucket-name} // show content of file

aws s3 rb s3://{bucket-name} --force // remove bucket with content

aws s3 ls // list buckets

* Assuming IAM Roles (CLI):

Create IAM roles from CLI console

option Roles, EC2-Full-Access

Add profile:

A screenshot of a computer

Description automatically generated

Use arn of the role (don’t use access keys, don’t use credentials file)

command:

aws ec2 describe-instances

aws ec2 describe-instances --profile ec2-full-access // show status of instances

option Roles, S3FullAccess

command:

aws s3 ls // access denied

aws s3 ls --profile ec2-full-access

Create my own profile:

command:

aws configure --profile {name}

add access-key

aws configure --profile {name}

See config and credentials files in computer

* **Summary**

NACLs = Network Access Control Lists

CIDR = Classless Inter-Domain Routing, ex. 10.0.0.0**/16**

**NAT** = Network Address Translation, when the instance with the private address wants to connect to the Internet, it will need a public address that can access the Internet.

NAT gateway: to enable outbound internet access for resources in private subnets

**Role**: contains **permissions**/**policies** with **credentials**

Each **Profile** with **credential**

KMS = Key Manager Service

To encrypt the password of environment variables

IOPS = Input Output per second = performance for the disc

Use SQS FIFO queues for ordered processing

**User pool** can be used to **authenticate** but the **identity pool** is used to provide **authorized** access to AWS services.

**Idempotent** = performing it again (using the same inputs) does not change the result.

A diagram of a cloud computing system

Description automatically generated

reduce **cold starts**: latency when service initializes resources for the **first time**

**failover** = method of protecting computer systems from a failure. ex.: the second server went into failover mode within a minute.

**fallback** mechanism: strategy to handle failures by providing alternative solution.

**fan-out**: design pattern where a single request or event **triggers** **multiple** downstream **processes**. Involves services in **parallel**.

IAM policy within the Amazon Cognito identity prefix

Cross-origin resource sharing (**CORS**): access resources in other domain.

responsiveness = performance

Resiliency = recover quickly

inbound access = acceso entrante

invalidate = update

* **Stateful vs stateless:**

**Stateful**:

Maintains state (information): data or context from previous interactions is stored (memory, database).

Uses: Scenarios requiring **continuous context** (user authentication in a traditional login session, online games).

Ex: Database sessions, Desktop applications (remember user preferences or data), Chat apps.

Disadvantages: complex, harder to scale

**Stateless**:

No State Retention

Independent Request

Easy Scalable

Uses: High scalability and lightweight interactions (RESTful APIs for web services, serverless functions).

Ex: HTTP protocol (request/response is independent), RESTful APIs, Microservices.

Disadvantages: Less suited for workflows requiring persistent context.

A screenshot of a computer

Description automatically generated

* IaaS, PaaS:

IaaS = Infrastructure as a Service (virtual server). AWS manage the physical hardware and software (virtualization)

EC2

CloudFront

IaC = Infrastructure as a Code

AWS Copilot

CloudFormation

AWS SAM templates

PaaS = Platform as a Service (code/data)

Elastic Beanstalk

A screenshot of a computer

Description automatically generated

* **Deployment options**:

**All at once**: **quickest** deployment, deploys the new version to all instances simultaneously.

Beanstalk

**Rolling**: Update a few instances at a time (bucket), and then move onto the **next bucket** once the first bucket is healthy (downtime for 1 bucket at a time). Updates are applied to existing instances in batches.

Beanstalk

**Rolling with additional batch**: launches new instances in a batch ensuring that there is full availability. Still updates existing instances.

Beanstalk

**Immutable**: Launches **new instances** in a new ASG. AWS Elastic Beanstalk creates a new set of instances with the updated configuration.

Beanstalk

**Blue/Green deployment**: Zero downtime and release facility. Create a new “**stage**” environment and deploy updates there. A **new environment** (**green**) is created with the updated configuration and **traffic** is **shifted** **from** the **old environment** (**blue**) to the **new** one.

AWS Lambda, Amazon **ECS.** When CodeDeploy have your live traffic going to target group one and then is going to actually update target group two, and then it’s going to take out target group one.

Beanstalk

CodeDeploy

**Blue/Green Traffic Shifting:**

**Canary:** traffic is shifted in two increments.

**Linear:** traffic is shifted in equal increments.

**All-at-once:** All traffic is shifted from the original Amazon ECS task set.

**in-place**: **cannot** used by **ECS.** The code updated while it’s actually running, means the application is likely to experience downtime.

CodeDeploy

* **ECS - Task placement strategies**: algorithm for selecting instances for task placement/termination. Only to the **EC2** launch type.

**binpack**: place tasks based on the **least available amount** of **CPU** or memory. **Minimizes** the number of **instances** in use.

**random**: place tasks randomly

**spread**: place tasks based on the specified value (**instaceId or host**).

* Others

Lambda Function Invocations:

Synchronous: Exponential backoff

* **Compute**
* **Amazon Elastic Compute Cloud - EC2**:

ecs-cli: Amazon Elastic Container Service

ec2 instance = terminal instance

**Web service** that provides **resizable compute capacity** in the cloud

Launch **virtual server** instances, is **not serverless**, these instances operate on **physical servers**.

run virtual servers in the cloud

IaaS

igw = Internet Gateway

Amazon Machine Images (**AMIs**) are templates for your instances (OS (Linux, Ubuntu, Windows Server, MacOS), software packages)

**EC2 instance profile:**

Role:

Policy: use IAM

Permission

Credential: each profile with credential

AWS credentials: use **IAM** for temporary credentials (**not use long term credentials**)

AWS Secrets Manager: use long-term credentials, **not** fully no long-term credentials

**Role > Credential**

Update the IAM **instance** **profile** that is attached to the EC2 instance to include the S3:ListBucket permission for the S3 bucket.

**Instance metadata** = data about your EC2 instance

**User data** = way to run scripts when we start our instances the first time

**Metadata** = way that we can retrieve specific information about the instance itself

aws-dva-code\amazon-ec2\user-data-metadata.md

Access Keys and IAM Roles with EC2:

Using IAM Role is more secure than using IAM User. **IAM Roles** use the AWS Security Token Service - **STS**.

commands:

aws s3 ls

pwd

aws configure

Amazon EC2 Auto Scaling:

Maintain the availability and automatic scaling of our EC2 instances.

* **AWS Batch:**

Run batch jobs (for ML model training, simulation, analysis) at any scale, in Amazon ECS, EKS, Fargate

* **AWS Elastic Beanstalk:**

Run, deploy and manage **web apps**

Handles the deployment details of **capacity provisioning**, **load balancing**, **auto-scaling**, application **health** **monitoring**.

PaaS

Only put your **.zip package** into AWS Elastic Beanstalk.

**BatchGetItem** can return one or multiple items from one or more tables.

Elastic Beanstalk **configuration** files (.**ebextensions**) can be added to your web application’s source code.

**Deployment options**:

**All at once**: **quickest** deployment, deploys the new version to all instances simultaneously.

**Rolling**: Update a few instances at a time (bucket), and then move onto the **next bucket** once the first bucket is healthy (downtime for 1 bucket at a time). Updates are applied to existing instances in batches.

**Rolling with additional batch**: launches new instances in a batch ensuring that there is full availability. Still updates existing instances.

**Immutable**: Launches **new instances** in a new ASG. AWS Elastic Beanstalk creates a new set of instances with the updated configuration.

**Blue/Green deployment**: Zero downtime and release facility. Create a new “**stage**” environment and deploy updates there. A **new environment** (**green**) is created with the updated configuration and **traffic** is **shifted** **from** the **old environment** (**blue**) to the **new** one.

**Deploy a remote Docker image to Elastic Beanstalk**. Elastic Beanstalk uses the **docker-compose.yml** file to pull and run your image if you are using Docker Compose. Otherwise, Elastic Beanstalk uses the Dockerrun.aws.json instead.

* **AWS Lambda:**

Is a serverless service. Run code in response to events, without thinking about servers or clusters

Run code 15 min max, for longer use EC2 instances.

**Uses:**

Data processing

Real-time file processing

Real-time stream processing

Build serverless backends for web, mobile, IOT, and 3rd party API requests.

Function can access:

AWS services or non-AWS services

AWS services running in VPCs (RedShift, Elasticache, RDS instances)

Non-AWS services running on EC2 instances in an AWS VPC

**Lambda Function Invocations:**

A screenshot of a computer

Description automatically generated

**Invoking a Lambda function asynchronously:**

Amazon Simple Storage Service (Amazon S3) and Amazon Simple Notification Service (Amazon SNS) invoke functions asynchronously to process events.

You can configure how Lambda **handles errors**, and can send invocation records to a downstream resource such as Amazon Simple Queue Service (**Amazon SQS**) or Amazon **EventBridge** to chain together components of your application.

A diagram of a diagram

Description automatically generated with medium confidence

**Lambda Function Concurrency:**

If the **concurrency** limit is **exceeded** **throttling** occurs with error “Rate exceeded” and a **429** “**TooManyRequestsException**”

There is **no** **maximum** **concurrency** **limit** for **Lambda** functions (depends on use case).

To **avoid** **throttling**, **request** **limit** **increases** at least 2 weeks ahead of time.

**Reserved Concurrency**: guarantees a set number of **concurrent executions** will be available for **critical function**. To **throttle** a **function**, set the reserved concurrency to **zero**. This **stops** any **events** from being **processed** until you remove the limit.

Used when you need to **control the maximum concurrent executions** for a function (to prevent a specific Lambda function from **over-consuming resources** in **high-traffic scenarios**) or ensure resources for **critical** functions.

**Provisioned Concurrency**: the function scales with the same burst behavior as standard concurrency. Ensures that a specific number of instances of the function are initialized (avoid **cold starts**) and ready to handle requests immediately.

Used when you need predictable performance with **low latency (real-time APIs, interactive apps)**, especially for interactive or time-sensitive workloads.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Reserved Concurrency** | **Provisioned Concurrency** |
| Concurrency Management | Sets a max limit for executions. | Pre/warms function instances. |
| Cold Start | Cold start occurs during scaling. | Eliminates cold start for provisioned instances. |
| Cost | Pay only for executions. | Pay for pre/warmed instances and executions. |
| Purpose | Limits resource usage for a function. | Reduces latency for high/performance needs. |
| Scaling Behavior | Throttles when concurrency is reached. | Scales instantly for provisioned instances. |

**Event Source Mapping:**

Is an AWS Lambda resource that reads from an event source and invokes a Lambda function.

To process items from a **stream** or **queue**, you can create an **event source mapping**.

Used for processing **events** in:

Amazon **SQS** queues

Amazon **Kinesis** **streams**

Amazon **DynamoDB** **streams**

**Event Notifications:**

Amazon **S3** can send an event (**notification**) to a Lambda function when an object is created or deleted. **Lambda** runs code in response to an event (**trigger**).

Lambda function process a file/object created in S3:

1. Create AWS **EventBridge** rule to detect object created in S3.

2. Add **trigger** of type **EventBridge** to Lambda function.

**Introducing AWS Lambda Destinations:**

Lambda asynchronous invocations can put an event or message on Amazon Simple Notification Service (**SNS**) topic, Amazon Simple Queue Service (**SQS**), **Lambda** function, or Amazon **EventBridge** for further processing

<https://aws.amazon.com/blogs/compute/introducing-aws-lambda-destinations/>

**Lambda Deployment Packages:**

Types:

**Container images**: are uploaded to the Amazon Elastic Container Registry (ECR). Then the image is deployed to the Lambda function.

**.zip file archives**: code + dependencies. Deployment package is upload from Amazon S3 or your computer. 50MB (zipped), 250MB (unzipped), 3MB (console editor).

Write and upload code as a .zip file or container image.

**Lambda** **layer**: .**zip** file contains supplementary code or data (library dependencies, custom runtime, configuration files), contains shared dependencies. 5 layers at a time.

Layers are extracted to the /opt directory in the function execution environment.

**Lambda Environment Variables:**

Use **environment variables** (**key/value**) to adjust your function's behavior **without** updating **code**

Access to **local storage** in the /**tmp** directory, **512 MB and 10,240 MB**

For AWS Lambda function, the temporary storage /tmp up to 512 MB, **not durable**, and data is **lost** once the Lambda function execution ends.

A screenshot of a computer

Description automatically generated

To make your **AWS Lambda function idempotent**, you must design your function logic to correctly handle **duplicated events**. Idempotent function logic can help reduce the following issues:

Unnecessary API calls

Code processing time

Data inconsistency

Throttles

Latency

**Lambda Versions:**

You work on **$LATEST** which is the latest version of the code. This is mutable (changeable).

Each **version** has its own **ARN**.

**Lambda aliases**: for **traffic** **splitting (traffic shifting)**, **gradual** **deployment** (**canary**), automatic rollback.

Aliases are **mutable** (**changeable**).

Function URLs do not support weighted traffic shifting

Lambda **function aliases** allow you to create **pointers** **to specific versions** of a Lambda function. An **alias** can **point** to a **specific version** (e.g., **v1**) and can be updated to point to a different version (v2) seamlessly.

To roll back, you simply update the alias to point back to the previous version.

This approach has **minimal operational overhead** and is designed to handle **version management** for Lambda functions.

**Weighted aliases** allow you to **route traffic** (**shift traffic**) to different versions of a function based on weights that you assign.

A screenshot of a computer

Description automatically generated

**Best Practices for Lambda Functions:**

<https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html>

**Cache** static assets locally in the **/tmp** directory.

Use environment variables to pass operational parameters to your function.

Avoid using recursive code in your function.

* **Container**

Services related to running and managing **Docker** **containers** on AWS: **ECS, EKS, ECR**

* **Amazon Elastic Container - ECS**:

Container Management Service, highly secure, reliable, and scalable way to run **containers**

run applications on a managed cluster of Amazon EC2 instances

An Amazon **ECS Cluster** is a logical grouping of tasks or services.

An ECS Task is created from a **Task Definition**. An ECS Task is a running **Docker container**. Each **task** will come from an **image** (**Docker images**) and that can be stored in a repository such as the **Amazon Elastic Container Registry (ECR)**.

ECS **Services** are used to maintain a desired count of tasks.

A screenshot of a computer program

Description automatically generated

Images are built from a **Dockerfile**.

Only **Docker** containers are supported on **ECS**.

A **task definition** uses **Docker images** to launch **Docker** **containers** in Amazon **ECS**. Is a text file in **JSON** format that describes one or more containers, up to a maximum of 10.

**Supports:**

**Docker** containers

**Task placement strategies**: algorithm for selecting instances for task placement/termination. Only to the **EC2** launch type.

**binpack**: place tasks based on the **least available amount** of **CPU** or memory. **Minimizes** the number of **instances** in use.

**random**: place tasks randomly

**spread**: place tasks based on the specified value (**instaceId or host**).

**Features:**

**ECS with EC2 is not serverless.**

**ECS with Fargate is serverless**: serverless compute for containers, managed for you and fully scalable.

Fully managed container orchestration

Docker support

Windows container support

Elastic Load Balancing integration: using ALB or NLB

A screenshot of a computer

Description automatically generated

**Launch Types: EC2 and Fargate**

A screenshot of a computer

Description automatically generated

EC2 Auto Scaling Group (ASG)

* **Amazon Elastic Container Service for Kubernetes (EKS):**

run, and scale **Kubernetes** without thinking about **cluster** management

**Amazon ECS vs Amazon EKS**

* **AWS Copilot:**

deploying and managing **containerized** applications on AWS (ECS, AWS Fargate)

Is a **Command Line Interface** for launching and managing containers.

automate deployments using **AWS CodePipeline**, which integrates with services like **AWS CodeCommit**, **AWS CodeBuild**, and **AWS CodeDeploy**.

Infrastructure-as-code (IaC) templates

A screenshot of a computer

Description automatically generated

* **Amazon Fargate:**

Serverless compute engine for **containers** for **ECS** and **EKS**

there is **no EC2 host** to **install** the CloudWatch agent on. Applies to ECS on EC2 but not ECS on Fargate.

Uses:

Run containers without having to manage servers or clusters of Amazon EC2 instances.

Manage your **applications**, not infrastructure

Improve security through isolation for Amazon **ECS tasks** and Amazon EKS pods.

A screenshot of a computer

Description automatically generated

**run your tasks and services with the Fargate launch type**: you package your application in containers, specify the CPU and memory requirements, define networking and IAM policies, and launch the application. Each **Fargate task** has its own **isolation** boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

**Amazon ECS with the Fargate launch type**: The workloads are highly variable and therefore the company prefers to be charged **per running task**.

* **Elastic Container Registry (Amazon ECR):**

**fully managed container registry** that stores **container** **images**.

Integrated with Amazon ECS and Amazon EKS

Package the application into a **container image** by using the Docker CLI. Upload the image to Amazon Elastic Container Registry (Amazon ECR).

* **Analytics**
* **Amazon Athena:**

Query data in **S3** using **SQL**, for data stored in **relational**, **non-relational**, object, custom data sources.

**Amazon Athena <–> S3**

* **Amazon Maci:**

automated scanning and **pattern matching**

Use to run a job on the S3 buckets that contain the affected data. Filter the findings by using the SensitiveData:S3Object/Financial finding type.

* **Amazon CloudSearch -> Amazon OpenSearch Service:**

Managed Search service

Use to search, log analytics, real-time application monitoring, observability, **dashboard**.

* **Amazon Kinesis:**

Analyze **real-time** **video** and data **streams**

Uses:

Real-time apps (monitoring, fraud detection, live leaderboards)

From batch to real-time analytics

Analyze IoT device data: to send real-time alerts, thresholds

Video analytics apps: video playback, security monitoring, face detection, ML

**ProvisionedThroughputExceededException**: the request rate for the stream is too high, or the requested data is too large for the available throughput. You **must reduce the frequency** or **size** of your **requests**.

* **Amazon Kinesis Data Firehose:**

Real-time streaming delivery for any data, at any scale, at low-cost

Uses:

**Data Transformation**: invoke your **Lambda** function to **transform** incoming source data and deliver the **transformed data** to **destinations** (S3, HTTP endpoint)

* **Application Integration**
* **AWS Step Functions:**

**Visual** workflows for distributed applications

Create State Machine

A Step Functions execution receives a **JSON** text as input and passes that input to the first state in the workflow. Individual states receive JSON as input and usually pass JSON as output to the next state.

In the **Amazon States Language**, these fields filter and control the flow of JSON from state to state:

InputPath

OutputPath

ResultPath

Parameters

ResultSelector

Uses:

**Automate workflows without code**

**Orchestrate** **microservices**

Create data and machine learning (ML) pipelines

AWS Step Functions Local with **mocked** service integrations

**To state machine to reference the API Gateway endpoint**: Configure the CloudFormation template to reference the API endpoint in the **DefinitionSubstitutions** property for the **AWS::StepFunctions::StateMachine** resource.

* **Amazon API Gateway:**

Create, publish, maintain, monitor and secure **REST APIs** and WebSocket APIs.

Authorize access to your APIs with AWS Identity, Access Management (IAM), Amazon Cognito

**API types:**

HTTP API

WebSocket API

REST API

**Supports:**

**mock** integrations for API methods

REST APIs: OIDC, OAuth 2.0 authorization, CORS

HTTP APIs: AWS Lambda, HTTP endpoints

WebSocket APIs: stateful frontend for an AWS service (Lambda, DynamoDB), HTTP endpoint.

**REST APIs** and **HTTP** **APIs** support authorizers for AWS Lambda, IAM, Amazon Cognito.

**WebSocket** **APIs** support IAM authorization and **Lambda** **authorizers**.

Does **not** support **SOAP APIs**. Cannot process the XML SOAP data.

**Mapping Templates**: map the payload for integration request and response.

**API Gateway Stages and Deployments:**

Deployments must be created and associated with a **stage** for anyone to access the API.

A **stage** is a logical reference to a lifecycle state of your REST or WebSocket API (e.g. dev, prod, beta, v2)

**Stage Variables**: **environment** configuration (dev, test, prod). Used in Lambda function ARN, HTTP endpoint, Parameter mapping templates.

Stage variables are passed to the “context” object in Lambda.

Are used with **Lambda aliases**. Create a **stage variable** to indicate the corresponding **Lambda alias**.

**Caching:**

Allows you to cache the endpoint’s response for a specific amount of time (TTL min 0, max 3600).

Can reduce number of calls to the backend and improve latency of request to the API.

**Caches** are defined per **stage**.

You can **encrypt** caches.

Cache capacity between 0.5GB to 237GB

Clients can **invalidate the cache** with the header: **Cache-Control: max-age=0**

**Throttling:**

If you go over 10,000 requests per second or 5,000 concurrent requests, you will receive a **429 Too Many Requests error response**.

**Access Control:**

**Resource-based policies: JSON,** use ARN

**Identity-based policies**

IAM Tags

Endpoint policies for interface VPC endpoints

**Lambda authorizers**: Lambda function to control access to your API

A token/based: JWT, OAuth token

A request parameter-based: headers, query string parameters, stageVariables, $context variables.

Amazon Cognito user pools

* **Amazon EventBridge:**

Serverless event bus for SaaS apps & AWS services

**EventBridge (formerly Amazon CloudWatch Events):** help you to respond to **state changes** in your **AWS resources**. When your resources change state, they automatically send events into an **event stream**.

<https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-cwe-now-eb.html>

**Amazon EventBridge rule:**

runs on a regular schedule to invoke the Lambda function.

AWS Lambda function to be invoked every 10 minutes

Uses:

**event-driven architectures**

create point-to-point integrations

* **Amazon Simple Notification Service - SNS:**

Fully managed **messaging** service

Is a **publisher**/**subscriber** system

Exchange of messages between **distributed systems**, **microservices** or applications using **Pub/sub** (publish/subscribe), **SMS**, **email** and **mobile** **push** **notifications**

FIFO messaging

**Encrypt** messages with AWS Key Management Service (**KMS**)

**Amazon SNS topic:**

Multiple recipients can be grouped using **Topics**

A **topic** is an “**access point**” for allowing recipients to dynamically **subscribe** for identical copies of the **same notification**.

* **Amazon Simple Queue Service - SQS:**

Managed message **queues** for **microservices**, distributed systems, serverless apps

**FIFO** queues

Is **pull-based** (polling/checking the queue), not push-based (like SNS)

Messages are up to **256KB** in size. Can be kept in the queue from 1 minute to 14 days.

Security with AWS Key Management (**KMS**), HTTPS/TLS

Supports **dead-letter queues** (**DLQ**), which other queues (**source queues**) can target for messages that can't be processed (consumed) successfully.

**Amazon SQS Queue Types:**

Standard Queue

**FIFO** Queue: require the Message Group ID and Message **Deduplication** ID parameters to be added to messages.

**Delay Queue**: delay in processing

**SQS Long polling**: to retrieve messages from SQS queues, waits for messages to arrive. Can lower costs.

**SQS Short polling**: polling returns immediately (even if the message queue is empty).

**Key attributes**:

**DelaySeconds**: configures a delay queue. 0/s to 900/2 (default 0/s).

**ReceiveMessageWaitTimeSeconds**: sets short/long polling. 0/s to 20/2 (default 0s).

**VisibilityTimeout**: give you a period of time in which the message is invisible. 0/ to 43,200/s (12 hours) (default 30/2).

**ReceiveMessage**:

**WaitTimeSeconds:** enable long-poll

**SendMessage**:

**DelaySeconds**: delays a message

**MessageDeduplicationId**: for **FIFO** only

**MessageGroupId**: adds a tag for a message group (FIFO only)

**Message Deduplication**: ensure that **duplicate messages** are **not** **processed** **multiple** **times**. For FIFO queues.

* **AWS AppSync:**

Fully-managed, scalable **GraphQL APIs**

Connect applications to events, data, AI models

to access data from **multiple databases**, **micro-services**, and AI models with a **single** GraphQL API request

**GraphQL** for **DynamoDB** and **Aurora** databases

Server-side data caching capabilities reduce the need to directly access data sources.

* **Database**
* **Amazon Aurora:**

Manage **relational** database for **MySQL** and **PostgreSQL**

has 5x the throughput of MySQL and 3x of PostgreSQL

data durable across 3 AZs (customers only pay for 1 copy)

**Replication**:

Global Database: best replication performance

**binlog-based replication**: with **external** MySQL databases

* **Amazon DynamoDB:**

Managed **NoSQL** database

**key-value** and document database

serverless

data synchronously replicated across 3 AZs in a region

used for **storing session data**

All data is stored on **SSD** storage

Horizontal scaling

**DynamoDB Supported Data Types:**

Scalar Types: one value (number, string, binary, Boolean, null)

Document Types: JSON, list and map.

Set Types: multiple scalar values.

**DynamoDB Table Classes:**

DynamoDB Standard

DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA)

**DynamoDB Access Control:**

DynamoDB supports **identity-based policies**, doesn’t support **resource-based policies**.

**DynamoDB** **Streams**:

build serverless **event-driven** applications

captures a time-ordered sequence of item-level modifications in any DynamoDB table and stores this information in a log for up to 24 hours.

**AWS Lambda triggers**: code automatically **respond** to **events** in **DynamoDB Streams**

**DynamoDB Accelerator** (**DAX)**: Full managed **in-memory cache** for DynamoDB that **increases** **performance and reduce latency** of **DynamoDB** tables and **offload read requests**. **Cannot** used in front of an Amazon **RDS** database.

Provides managed **cache** **invalidation**, data population, and cluster management

Used to improve **READ** **performance** (**not writes**).

**DynamoDB** provides low latency (**milliseconds**). **Microsecond** latency can be achieved with DynamoDB Accelerator (**DAX**).

**Partition** **key**

**Composite** **key**: Partition key + Sort key

**DynamoDB** evenly distributes provisioned throughput – read capacity units (RCUs) and write capacity units (WCUs) among partitions.

**DynamoDB Consistency Models:**

**Eventually consistent reads**: when you read data from a DynamoDB table, the response might not reflect the results of a recently completed write operation.

**Strongly consistent reads**: DynamoDB returns a response with the **most up-to-date data**.

**Local Secondary Index (LSI)**: querying with **same** **partition** **key**, **different** **sort** **key**. **Strongly consistent reads** (**retrieves the most-up-to-date data**). Can be added during the **table** **creation**.

**Global Secondary Index (GSI)**: querying with a **different** **partition** **key and sort key**. **Eventually** consistent reads (**not return the latest data**). Can be added to **existing tables**.

**DynamoDB Provisioned Capacity:**

Read Capacity Units (RCUs)

Write Capacity Units (WCUs)

**Throttling** occurs when the configured RCU or WCU are exceeded. You may receive the following error: **ProvisionedThroughputExceededException**. The AWS **SDKs** for DynamoDB **automatically** **retry** requests that receive this exception.

Adding a Time To Live (**TTL**) to Items: used to automatically **expire** items.

Attribute **Expire** based on Unix **Epoch** time, is a number of time, used when removing storing **session** **state** data.

* **Amazon ElastiCache:**

**In-memory** **database** **cache** used in front of Amazon **RDS**

Is a **key/value store**

web service to deploy and run **Memcached** or **Redis** server nodes (replication) in the cloud.

**low latency** (in-memory database) than **DynamoDB**

is not durable

**does not** offer the same durability and **consistency** guarantees as **DynamoDB**.

**2 types of ElastiCache engine:**

**Memcached**: simplest model, scaled in and out, can cache objects such as DBs, **not** support **encryption** or high availability. **Multithreaded** **caching** solution that can be used to **offload heavy read traffic** from Amazon RDS instances. For **temporary** data, **not** for **persistent** **idempotency** checks. **Not replication**.

Add nodes to a cluster.

Scale vertically (node type), must create a new cluster manually.

**Redis**: complex model, master/slave **replication**, cross AZ (AH), automatic **failover** and backup/restore, for **encryption** and **high availability** use ElastiCache **Redis** with **cluster mode** enabled.

**Cluster mode disabled**: add replica or change node type.

**Cluster mode enabled**: online resharding, vertical scaling. Offline resharding.

A screenshot of a computer

Description automatically generated

**Uses:**

For **storing session state data (key/value)**, low latency, high performance

Lowest latency

Real-time application **data caching: Real-time performance** for **real-time applications**

Real-time session stores: session data for gaming, e-commerce, social media, online apps

<https://digitalcloud.training/amazon-elasticache/>

**Caching strategy**:

**TTL strategy cache**: Time-To-Live. By **expiring** items

**write-through cache**: most **recent** value for each data item. The cached data must populate **real-time** dashboards.

**lazy loading cache**: data is loaded into the cache only when it is **requested** by the application

**read-through cache**: to cache frequently read data

**write-behind cache:** for improving write performance but with a tradeoff in consistency

* **Amazon RDS - Relational Database Service:**

Managed relational database service for PostgreSQL, MySQL, MariaDB, SQL Server, Oracle, Db2

Amazon **Aurora**: MySQL, PostgreSQL

Runs on Amazon EC2 instances

RDS uses Amazon **EBS** **volumes** for storage

* **Developer Tools**

code, build, test and deploy app

SDKs, code editors, CI/CD

A screenshot of a computer

Description automatically generated

* **AWS Cloud9:**

Write, run, and debug code on a cloud IDE

Is an integrated development environment (IDE).

* **AWS CloudShell:**

Browser-based shell environment

AWS CLI in browser, **Linux**

<https://aws.amazon.com/cloudshell/?nc2=h_ql_prod_dt_cs>

* **AWS Command Line Interface - CLI:**

Unified tool to manage AWS services

<https://aws.amazon.com/cli/>

<https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-sso.html>

* **AWS CodeBuild:**

Compiles, build and test code, and produces software packages that are ready to deploy.

Is a fully managed continuous integration (CI) service.

Specify a local cache in CodeBuild. Add the **cache folder path** to the **buildspec.yaml** file for the build project

**Components:**

**Build project:** defines how CodeBuild will run a build

**Build environment:** the operating system, language runtime, tools for the build

**Build Specification:** a YAML file (**buildspec.yml**) that describes the collection of commands and settings for CodeBuild to run a build.

A screenshot of a computer

Description automatically generated

* **AWS CodeCommit:**

Managed **version control** service

host private Git repositories

commit packages

Store code in private Git repositories

IAM supports CodeCommit with 3 types of credentials:

Git credentials

SSH keys

AWS access keys

* **AWS CodeDeploy:**

Automate code deployments

**2 deployment type**:

**in-place**: **cannot** used by **ECS.** The code updated while it’s actually running, means the application is likely to experience downtime.

**blue/green**: AWS Lambda, Amazon **ECS.** When CodeDeploy have your live traffic going to target group one and then is going to actually update target group two, and then it’s going to take out target group one.

**AWS Lambda deployment type**: can manage the way in which traffic is shifted to the updated Lambda function versions during a deployment by choosing a **canary**, **linear**, or **all-at-once** configuration.

**Amazon ECS deployment type:** used to deploy an Amazon ECS containerized application as a task set.

**Blue/Green Traffic Shifting:**

**Canary:** traffic is shifted in two increments.

**Linear:** Traffic is shifted in equal increments.

**All-at-once:** All traffic is shifted from the original Amazon ECS task set.

**run order of the hooks for in-place deployments:**

ApplicationStop -> BeforeInstall -> AfterInstall -> ApplicationStart

* **AWS CodePipeline:**

Delivery pipeline to deliver application

Release software using **continuous delivery**

Integrated with:

AWS **CodeBuild:** to **test** and **build**

AWS **CodeDeploy:** to **deploy** the app

AWS **CodeCommit** repository: is triggered by changes to the **main branch**

* **AWS CodeGuru:**

Provides intelligent recommendations for improving application performance, efficiency and code quality.

* **AWS X-Ray:**

Analyze and **debug** your applications

Analyze the behavior of their production, distributed applications with **end-to-end tracing capabilities**. To identify performance **bottlenecks**, **edge case errors**, **detect issues**.

Instrument the application by using the X-Ray SDK for Python. Install the X-Ray SDK for all the services that the application uses.

If you add **annotations** with the **X-Ray SDK**, you can also **filter** based on the presence of an annotation key or the value of a key.

Install **X-Ray daemon** on the **on-premises servers** (**local**)

* **Management & Governance**
* **Amazon CloudWatch:**

**Monitoring** service for AWS cloud **resources** and **applications**, on premises, and on other clouds

Used to trigger actions based on changes in the state of AWS services, collect and track **metrics**, collect and monitor **log files**, and set **alarms**. Provides **insights** into operational **health.**

A diagram of a cloud watch

Description automatically generated

**Lambda** sends **metrics** to Amazon **CloudWatch** for performance monitoring, like Duration, Invocations, ConcurrentExecutions, Errors, Throttles.

CloudWatch observability

AWS X-Ray (**traces**) > Amazon CloudWatch (**metrics, logs**)

AWS X-Ray is useful for **tracing** requests across AWS services, but it doesn’t focus on detailed execution and access **logs** for **API Gateway**.

**CloudWatch Metrics**: services send time-ordered data points to CloudWatch.

**CloudWatch Alarms**: monitor metrics and initiate actions.

**CloudWatch Logs**: centralized collection of system and application logs.

can use **filter expressions. Metric filters** can be used to **trigger CloudWatch alarms**.

only publishes **metric data** for **events** **no** **count** **exceptions** in the application.

The **awslogs** **log driver** allows containers running in Amazon **ECS** to send their logs directly to **Amazon CloudWatch Logs.**

**CloudWatch Events**: stream of system events describing changes to AWS resources and can trigger actions.

**Unified CloudWatch Agent:**

Collect internal system-level metrics from Amazon EC2 instances across operating systems.

Collect system-level metrics from on-premises servers.

Can be installed on:

Amazon EC2 instances

On-premises servers

Linux, Windows Server, MacOS

* **AWS CloudFormation:**

Create and manage **resources** with **templates**

**IaaC**, manage infrastructure with **DevOps**: automate, test, and deploy infrastructure templates with continuous integration and delivery (**CI/CD**) **automations**, **Rollback** Support.

Use **helper scripts** for software installation and application setup.

**CloudFormation template**: a template file **(.yaml or .json)** to define infrastructure (**S3** bucket and **EC2** instance).

Creates Amazon VPC security groups, Amazon EC2 security groups

A screenshot of a computer

Description automatically generated

Add the relevant key and bucket to the **S3Bucket** and **S3Key** properties in the CloudFormation template

**Intrinsic Functions:**

**Ref**: to reference **resources** within the **same** **stack,** return value of parameter/resource

**GetAtt**: returns value of attribute from a resource in the template

**ImportValue**: importa el valor de un recurso de otro template

**Resources (mandatory)**: declare resources

**Parameters (optional):** input custom values to your template

**AllowedValues**: to create a parameter with the list of EC2 instance types

**Mapping (optional)**: for **static** configuration values, not for **dynamically.** Matches a key to a corresponding set of named values.

**Outputs (optional)**: to **export** reference in response, to import into other stacks.

**Conditions (optional)**: statements

**Transform (optional):** specifies macros to process your template. Can be used to reference additional code stored in S3, such as Lambda code.

**AWS**::Serverless: Can define **AWS SAM** syntax.

**StackSets:** to **manage** **resources** across multiple **AWS Regions**.

**AWS CLI** **create-stack-set** command to create/deploy and run a stack set in **multiple** **Regions**

**AWS CLI deploy** command to deploy in a **single** **Region**

**CloudFormation stack**: is created when deploy the template.

Are deployed through the Management **Console**, **CLI** or **APIs**.

**drift detection: detect** **changes** of security groups

A screenshot of a computer program

Description automatically generated

You create, update and delete stacks using templates.

* **AWS Cloud Development Kit (AWS CDK):**

open-source software development framework to define cloud **infrastructure** using programming languages (TypeScript, JavaScript, Python, Java, C#, Go). Use these languages to create reusable **Constructs**.

Simplifies the process of creating, managing and deploying AWS **resources** through code.

**CDK code** is converted into **CloudFormation templates** for provisioning **AWS resources**.

**CDK assertions module** for **unit testing**.

**CDK Aspects class** to enforce **security** **rules**.

**Commands:**

**cdk synth**: to convert the code into a **CloudFormation template** (JSON or YAML).

**cdk bootstrap**: to set up **prerequisite infrastructure** in your AWS account.

**cdk deploy**: to deploy the incremental changes.

* **AWS CloudTrail:**

Track **user activity**, **API usage**, **API calls**

service that enables **governance**, compliance, operational **auditing**, and auditing of your AWS account.

CloudTrail logs, CloudTrail Insights

* **AWS AppConfig:**

Managing application configurations for large-scale configuration.

Assess, audit, and evaluate **configurations** of your **resources**

Track resources inventory and changes

**Feature Flags** configuration profile: to require **selective activation** of specific **features**

* **Networking & Content Delivery**
* **Amazon Virtual Private Cloud - VPC:**

Isolated cloud resources

Define and launch AWS resources in a logically isolated virtual network

Customize your virtual network by choosing your own **IP address range**, creating **subnets**, and configuring **route tables**.

VPC -> EC2 + RDS, VPCs across accounts, Availability Zones, or AWS Regions

**Public subnets** are accessible to the outside world (**internet**). **Private** **subnets** **not**.

We control the **VPC router** using the **route table**

**NAT** instances and NAT gateways allow your instances in **private** **subnets** which only have private IP addresses to be able to connect to the internet.

**Security Groups** and Network ACLs: are firewalls to protect the network traffic that’s able to connect to our EC2 instances.

**NACLs** don’t apply to traffic within the subnet, is only the ingress and egress traffic.

Security Group are applied to the Elastic Network Interfaces that are attached to each of your EC2 instances.

Security Groups support allow rules only. **Stateful** (if traffic is outbound then any return traffic is the same).

Network ACLs also support deny rules. **Stateless** (need separate rule for outbound and the return traffic inbound)

Create a Custom VPC:

VPC Console, option Your VPCs, Create VPC

option Subnets, Route tables

GitHub-Code/code/amazon-vpc/custom-vpc.md/Create Public and Private Subnets

EC2 Dashboard, in Network setting choose your VPC

* **Amazon CloudFront:**

Global Content Delivery Network (**CDN**) with **HTTPS** communication

**IaaS** (Infrastructure as a Service)

Distribution of **static and dynamic content of web service (.html, .css, .php, graphics files).** Distribute media files using HTTP or HTTPS.

Add, update, or delete objects, and submit data from web forms.

used for **caching content** for better global performance

**Behaviors**:

Path Pattern

**Viewer Protocol Policy**

Cache Policy

**Origin Request Policy**

**CloudFront Functions** for **customizing** **URL behavior**.

**CloudFront Signed URLs:** provide more control over access to content. Can specify beginning and expiration date and time, IP addresses.

**CloudFront Origin Access Control (OAC, formerly OAI):**

S3 bucket policy to allow CloudFront OAC with read and write access -> Origin Access Control (**OAC**) in **CloudFront**

Use **OAI** to restrict users so they cannot access the content directly using the S3 URL, they must connect via **CloudFront**.

**Lambda@Edge:**

is a feature of Amazon **CloudFront** that allows developers to **run** **AWS** **Lambda** **functions** closer to the **end user**, at AWS edge locations worldwide.

Integrates with AWS Security Token Service (AWS **STS**)

dynamically **optimizes photos** based on the device's request

**Uses cases:**

Custom Authentication and Authorization

Content **Personalization**: viewer characteristics (geolocation, device type, or cookies)

A/B Testing

* **Amazon Route 53:**

Scalable domain name system (**DNS**)

Can perform **domain registration**

Route end users to Internet apps

provides highly available and scalable [Domain Name System (DNS)](https://aws.amazon.com/route53/what-is-dns/), [domain name registration](https://aws.amazon.com/blogs/networking-and-content-delivery/benefits-of-domain-registration-with-amazon-route-53/), and [health-checking](https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/dns-failover.html) cloud services

* **AWS VPN:**

Securely access your network resources

Connect your on-premises networks and remote workers to the cloud

* **Elastic Load Balancing (ELB):**

Distribute incoming traffic across multiple targets

Security: Secure your applications with SSL/TLS termination, integrated certificate management, and client certificate authentication.

**Automatic scaling**

Create an Auto Scaling Group:

EC2/Load Balancers

Types:

**Application Load Balancer**: operates at the request level. Used in web apps (HTTP/HTTPS), microservices (Docker), Lambda. **Layer 7**.

**Network Load Balancer**: operates at the connection level. Used in TCP/UDP apps, static IP addresses, VPC endpoint services.

**Gateway Load Balancer**: Used in front of virtual appliances (firewalls). **Layer 4**.

Create a Scaling Policy

Create ASG and ALB using the AWS CLI

**Session state** data can be stored in databases such as **DynamoDB** and **ElastiCache**. Can include **temporary** **data**, **metadata**, **authentication** information.

**Sticky sessions** can be used on **ELBs** to bind a **session** to an **EC2** instance. It uses **cookies** that are generated at the ELB level. Are **not** supported with **TLS** listeners and TLS target groups (NLB).

* **Security, Identity, & Compliance**

**AWS credentials**: use **IAM** (AWS Identity and Access Management)

**Encryption keys**: use **KMS** (AWS Key Management Service)

**AWS CLI** is not designed for direct file encryption, instead, use **Encryption SDK**.

**SSH keys**: use Amazon EC2 Instance Connect

**Private keys and certificates**: use AWS Certificate Manager (ACM)

**AWS Secrets Manager:** Rotate, manage, and retrieve secrets (**database credentials**, **API keys**, **token,** OAuth)

**AWS Systems Manager Parameter Store**: Secure String, for storing and retrieving non-sensitive configuration data (API URL) across environments (development, testing, production), credentials. **Less cost than AWS Secrets Manager**. **No** native **rotation**.

* **AWS Identity and Access Management (IAM):**

Securely manage access to services and resources

Grant **temporary security credentials** for workloads that access your AWS resources

IAM policies: east-privilege policies

* **Amazon Cognito:**

Management service for **authentication (sign-in)** and **authorization (sign-up)**

Customized sign-up and sign-in

Identity management for your apps

Leverages **IAM roles** to generate **temporary credentials** for your application’s users

**Amazon Cognito user pools**: for external identity providers

**User pool**: to **authenticate**

**Identity pool**: to **authorized** access to AWS services. Used to obtain **temporary**, limited-privilege credentials for AWS services.

* **AWS Key Management Service (KMS):**

**KMS** provides **audit** functionalities

Create and managed symmetric and asymmetric encryption keys.

**Symmetric Encryption** uses a **single** **key** for both **encryption** and **decryption**. Support **rotation**. Used by **KMS**.

**Asymmetric Encryption** uses a **data** **key pairs (public** and **private)** for **encryption** and **decryption**. Used by AWS **CloudHSM**.

used to create and manage the **encryption** **keys** used for encrypting data at **rest**.

The KMS keys are protected by hardware security modules (**HSMs**).

Used to be known as customer master keys (**CMKs**).

You cannot manage KMS keys, **rotate** them, or change their key policies.

Automatic **key** **rotation**:

Every year

Include key ID, key ARN, region, policies, permissions.

Is not supported on:

**Asymmetric** KMS keys

HMAC KMS keys

KMS keys in custom key stores

KMS keys with imported key material

* **AWS Secrets Manager:**

**Rotate**, manage, and retrieve secrets (**database credentials**, **API keys**, **token,** OAuth).

Used for storing secret information such as **database connections strings** and **passwords** with **API access**.

increase security, use this instead of **environment variables** to **store database credentials** (for **long-term credentials**) and other sensitive information, retrieve them at runtime **without** using **long-term credentials**.

Support **secret replication** for disaster recovery scenarios.

Use AWS Identity and Access Management (**IAM**) permissions **policies** to manage access to your **secrets**.

Integrate secrets with AWS logging, monitoring, and notification services.

Offer automatic rotation of credentials (built-in) for:

Amazon RDS (MySQL, PostgreSQL, Amazon Aurora)

Amazon Redshift

Amazon DocumentDB

For other services you can write your own AWS **Lambda** function for **automatic** **rotation**.

Components:

Automation

Run Command

Inventory

Patch Manager

Session Manager

Parameter Store

* **AWS Systems Manager Parameter Store**:

Secure String. for storing and retrieving non-sensitive configuration data (API URL) across environments (development, testing, production), credentials. **Less cost than AWS Secrets Manager**. No native rotation.

* **AWS Private Certificate Authority (CA):**

provides strong security and restrictive **access controls**.

allowing more permissive access and bulk certificate issuance for subordinate CAs lower on the chain.

**create** **private** **certificates** to identify resources and protect data.

create versatile **certificate** and CA **configurations** to **identify** and protect your **resources** (servers, applications, users, devices, and containers).

direct integration with **AWS IAM** (IAM policies)

Support:

Managed Public Key Infrastructure (**PKI**)

  IAM integration.

      Auditing with AWS CloudTrail.

Private certificates.

      Subordinate certificate authorities (CAs).

* **AWS Certificate Manager (ACM):**

used for issuing **SSL/TLS certificates**

Create, store, renew **SSL/TLS X.509 certificates**

Single domains, multiple domain, names and wildcards

Integrates with:

Elastic Load Balancing

Amazon CloudFront

AWS Elastic Beanstalk

AWS Nitro Enclaves

AWS CloudFormation

Public certificates are signed by the AWS public Certificate Authority

Create a Private CA with ACM

Issue private certificates

<https://digitalcloud.training/aws-certificate-manager/>

* **Environment variables:**

To **store secrets securely** and adjust your function’s behavior **without updating the code**.

* **Storage**
* **Amazon Simple Storage Service - S3:**

**object-based** storage: **key-value** pairs rather than in a file or block structure

Is a public service

**File Storage vs Object Storage**

Stores data as objects within **buckets**

**Bucket**: container for objects

Buckets cannot be created within other buckets.

Buckets are created **within** a **region**.

**Object**: file and **metadata** that describes the file (.doc, .pdf, .mp4, .jpg, etc). Files from 0 to 5TB.

An object consists of:

Key (name of objects)

Version ID

Value (actual data)

Metadata

Subresources

Access control information

A folder is a shared **prefix** for grouping objects.

**Best Practices:**

Create buckets in regions that are physically closest to your users to reduce latency.

Support versioning

**doesn’t** support **HTTPS** for website endpoints. The **HTTP** protocol is used with a **REST API** (GET, PUT, POST, DELETE).

S3 is accessed over **Internet**

S3 -> CodeDeploy

SQL queries

bucket: logs

**S3 Lifecycle Management:** There are two types of actions:

**Transition actions**: Define when objects transition to another storage class.

**Expiration actions**: Define when objects expire (deleted by S3).

**S3 Lifecycle rules** for **automating** object **deletion** based on **age**.

Uses:

**Data lakes**: centralized repository to store structured and unstructured data

**backup** (databases)

**archiving** (files)

**Content Delivery** (images, videos, documents)

**static websites** (HTML, CSS, JavaScript)

A screenshot of a computer

Description automatically generated

**S3 storage classes:**

**S3 Standard**: for frequently accessed data

**S3 Intelligent-Tiering**: for **automatic cost savings** for data with unknown or changing access patterns.

S3 Express One Zone: for your most **frequently** accessed data

**S3 Standard-Infrequent Access (S3 Standard-IA)** and **S3 One Zone-Infrequent Access (S3 One Zone-IA)**: for **less frequently** accessed data

**S3 Glacier Instant Retrieval**: for archive data that needs **immediate access**

**S3 Glacier Flexible Retrieval (formerly S3 Glacier)**: for **rarely** accessed **long-term** data that does not require immediate access

**S3 Glacier Deep Archive**: for **long-term** archive and digital preservation with retrieval in hours at the lowest cost storage in the cloud.

**S3 Glacier Deep Archive with standard retrieval**: **24**-hour access requirement

**S3 Glacier Deep Archive with bulk retrieval**: Can take up to **48** hours

<https://aws.amazon.com/s3/storage-classes/>

**Amazon S3** **Event Notifications:** Sends notifications when events happen in buckets. For **Real-Time Updates. Destinations** include:

Amazon Simple Notification (SNS) topics.

Amazon Simple Queue Service (SQS) queues

AWS Lambda functions: can be configured to trigger an AWS Lambda function when specific **events occur in an S3 bucket**, such as object creation or deletion.

**CORS with Amazon S3**: Configure **CORS** to **access the resources in other domains.**

Enabled through setting:

Access-Control-Allow-Origin

Access-Control-Allow-Methods

Access-Control-Allow-Headers

**S3 Object Lambda:**

is a new feature that allows the developer **to add custom code to S3 GET requests**. The developer can create an **S3 Object Lambda** **function** to **remove** the PII from the document and configure S3 to use the function whenever an object is requested from a specific access point.

**S3 Access Control Lists (ACLs):**

AWS recommends using **S3 bucket policies** or **IAM policies** rather than **ACLs**.

**S3 presigned URLs:** to grant **time-limited access** to objects in Amazon **S3** without updating your **bucket policy**

**Design Patterns for Optimizing S3 Performance:**

Use caching services to cache the latest content:

Amazon CloudFront (CDN)

Amazon ElastiCache (in-memory cache)

Horizontally scale requests across S3 endpoints

**Encryption options:**

Server-side encryption with S3 managed keys (SSE-S3)

Server-side encryption with AWS KMS managed keys (SSE-KMS)

Server-side encryption with client provided keys (SSE-C)

Client-side encryption

**Objects** are encrypted using **server-side encryption**.

* **Amazon Elastic Block Store (EBS):**

**EC2** **block storage** **volumes**

Block based storage system, not a File-based storage system

Have volumes

AZ = Availability Zone

**Instance Store volumes** are **ephemeral** (temporal, data is **non-persistent**). If the power is lost to this host server, all that data is gone forever.

Used for temporary data that can be recreated.

**EBS volumes** offer **persistent** storage. If you have long term data storage, you cannot afford to potentially lose your data.

**Amazon EBS Snapshots**: to do **backups**

AMI = Amazon Machine **Image** are backed by a snapshot, is a template contains software configuration (operating system, app server, apps) required to launch your instance. Snapshot ID

Create and Attach an EBS Volume:

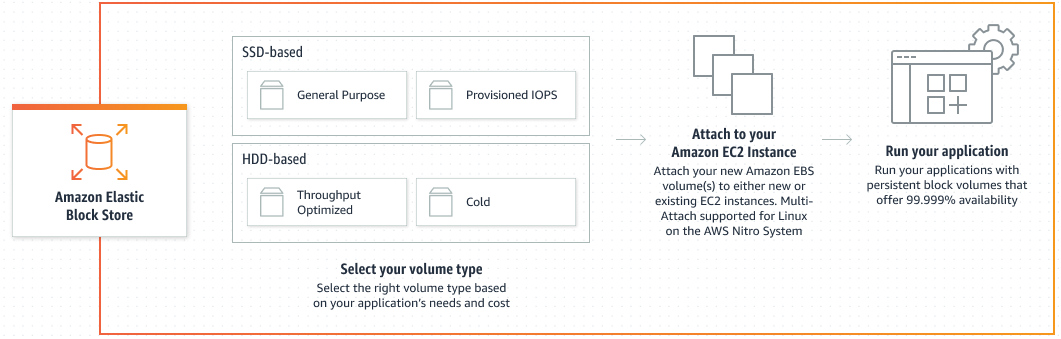
aws-dva-code\amazon-ebs\amazon-ebs-volumes.md

Snapshots are not stored in the availability zone, they’re store on Amazon S3 (regional service).

VPC -> ECS -> Instances -> Volumes, Snapshots

Uses:

Run relational or NoSQL databases



* **Amazon Elastic File System (EFS):**

Is **shared** file system, connect instances from multiple availability zones

**NFS** - Network File System: connection protocol

aws-dva-code\amazon-efs\working-with-efs.md

Create a File system using the Amazon elastic File system service.

EF2 = Managed File Storage for EC2

Fully managed file system for **EC2**

A screenshot of a computer

Description automatically generated

Uses:

Simplify DevOps

Accelerate data science: for machine learning (ML) and big data analytics workloads

Enhance Content Management Systems (CMS)

* **Others**
* **Amazon Inspector:**

Automated security assessment service that helps improve the security and compliance of applications deployed on AWS.

* **Amazon RDS Proxy:**

**Manage database** **connections** in serverless environments (AWS Lambda) using connection pool **reduces** the number of direct connections to the database.

* **Dead Letter Queue (DLQ):**

Configure this on **AWS Lambda** to give you more control over message handling for all **asynchronous** **invocations**, including those delivered via **AWS events** (**S3**, **SNS**, **IoT**). Saves **discarded events, unprocessed events** for further processing.

Set the dead-letter queue as an **event source** for the Lambda function.

Used for **debugging** your application or messaging system because they let you isolate unconsumed messages to determine why their processing doesn't succeed.

**DLQ** can be an **Amazon SQS queue** or an **Amazon SNS topic**.

* **AWS Serverless Application Model (AWS SAM):**

<https://aws.amazon.com/serverless/sam/>

**simplifies the development and deployment** of serverless applications.

Define and manage your **infrastructure** **code** with **AWS SAM templates (like CloudFormation but more simple and local)**. Define serverless resources in **YAML**.

Perform **real-time debugging** and **testing** in the cloud with **AWS SAM Accelerate**.

consists of two parts:

**AWS SAM templates**: IaC

**AWS SAM CLI**: create, develop, debug, and deploy serverless applications. Provides tools specifically for **local** **testing** of AWS Lambda functions.

**Commands**:

**sam package:** run a package.

**sam build** and **sam deploy**: to prepare your application for deployment and deploy to AWS.

**sam sync**: to watch for **local changes** and **quickly deploy** those changes to AWS. Then, use **sam remote invoke** to test your lambda functions in the cloud.

Deploy incremental changes to the AWS environment.

Avoid a full application deployment for every change.

Accelerate the feedback loop for testing.

**sam pipeline**: to create or modify pipelines for your continuous integration and continuous delivery (CI/CD) system.

**sam local invoke**: with the function's logical ID in the synthesized template to **test the function locally**.

**sam local start-lambda**: to **simulate** the Lambda **runtime** environment locally and is useful for integration testing with other services. However, it is not intended for invoking a single function directly for testing.

**sam local start-api**: to start a local API Gateway instance. Allow test your REST API locally before deploying to production.

**SAML directory:** You cannot provide access to an on-premises SAML directory using a VPC endpoint.

* **AWS Amplify Hosting:**

**build** and **deploy** an **application**

Tools and features for building full-stack applications on AWS.

Build web and mobile backends, and web frontend UIs.

Amplify **deploys** your app to the AWS global content delivery network (**CDN**).

is purpose-built for hosting static websites and **serverless** applications. Integrates with **version control** **systems** (AWS CodeCommit, Bitbucket, GitHub).

Manages deployment **workflows**, including support for **phased releases** (development, staging, UAT, production).

Add a **test** **phase** to the **amplify.yml** build settings for the application

Uses **HTTPS** for secure data exchange by default.

Amplify is used to build and host the WebStore application and create backend services.

**is an all in one** service

**AWS Amplify Studio** is a **virtual** **interface** for building **web** and **mobile** apps.

Using AppSync and Amplify to **simplify** access to **microservices**

* **AWS Application Integration Services:**

<https://digitalcloud.training/aws-application-integration-services/>

A screenshot of a computer

Description automatically generated

