Motivation for Cloud Computing

Prior to cloud computing users who are in need of computing are expected to invest money on computing resources such as hardware, software, networking, and storage
This investment is huge to the users as they have to buy these computing resources, keep these in their premises, and maintain
This is very expensive to the enterprises that require enormous computing power and resources, compared with classical academics and individuals
This is a huge expenditure to the enterprises and it grows with the requirement of more computing
It is easy and handy to get the required computing power and resources from some provider (or supplier) as and when it is needed and pay only for that usage
This would cost only a reasonable investment or spending, compared to the huge investment when buying the entire computing infrastructure
Therefore, cloud computing is a mechanism of bringing–hiring or getting the services of the computing power or infrastructure to an organizational or individual level to the extent required and paying only for the consumed services
Therefore, cloud computing is needed in getting the services of computing resources.
Cloud computing is very economical and saves a lot of money.

What is Cloud

The term Cloud refers to a Network or Internet. In other words, we can say that Cloud is something, which is present at remote location.
Cloud can provide services over public and private networks, i.e., WAN, LAN or VPN.
Applications such as e-mail, web conferencing, customer relationship management (CRM) execute on cloud.

What is cloud Computing

The practice of using a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server or a personal computer .
Cloud Computing refers to manipulating, configuring, and accessing the hardware and software resources remotely. It offers online data storage, infrastructure, and application.

Types of Clouds

1.Public Cloud
Computing services, such as servers, storage, networking, and applications, are provided over the internet by a third-party cloud service provider.
These services are made available to the general public, and multiple organizations or individuals
Examples of well-known public cloud providers include Amazon Web Services (AWS),Microsoft Azure, Google Cloud Platform (GCP), IBM Cloud, and Oracle Cloud.
2.Private Cloud

A private cloud refers to a cloud computing environment that is used exclusively by a single organization.
Unlike public clouds, which are shared by multiple tenants, a private cloud is dedicated to a specific business or entity.
This cloud deployment model provides greater control, customization, and security for the organization's computing resources and services.
They are suitable for industries such as finance, healthcare, and government, where regulatory compliance and data privacy are critical considerations.

3.Hybrid Cloud

A hybrid cloud is a computing environment that combines elements of both public and private clouds, allowing data and applications to be shared between them.
In a hybrid cloud model, organizations can leverage the advantages of both public and private clouds to meet their specific business requirements.
Hybrid clouds are well-suited for organizations that have a mix of workloads with varying requirements for performance, security, and compliance

4. Community Cloud

A community cloud is a cloud computing model that is shared by multiple organizations with common interests, concerns, or compliance requirements.
This model is designed to meet the specific needs of a community of users while providing more control and customization than a public cloud.
Community clouds are particularly well-suited for industries or sectors that have shared regulatory requirements or face similar challenges. Examples include healthcare, finance, government, and research institutions

Five essential components of Cloud Computing

1.On-demand self-services:The Cloud computing services does not require any human administrators ,user themselves are able to provision,
2.Broad network access:The Computing services are generally provided over standard networks and heterogeneous devices.
3.Rapid elasticity:The Computing services should have IT resources that are able to scale out and in quickly and on as needed basis.Whenever the user require services it is provided to him and it is scale out as soon as its requirement gets over.
4.Resource pooling:The IT resource (e.g., networks, servers, storage, applications, and services) present are shared across multiple applications and occupant in an uncommitted manner.Multiple clients are provided service from a same physical resource.
5.Measured service:The resource utilization is tracked for each application and occupant, it will provide both the user and the resource provider with an account of what has been used.This is done for various reasons like monitoring billing and effective use of resource.

Amazon Lex

- Amazon Lex V2 is an AWS service for building conversational interfaces for applications using voice and text.
- Amazon Lex V2 enables any developer to build conversational bots quickly.
- With Amazon Lex V2, no deep learning expertise is necessary—to create a bot, you specify the basic conversation flow in the Amazon Lex V2 console.
- Amazon Lex V2 manages the dialog and dynamically adjusts the responses in the conversation.
- Using the console, you can build, test, and publish your text or voice chatbot.
- You can then add the conversational interfaces to bots on mobile devices, web applications, and chat platforms (for example, Facebook Messenger).
- Amazon Lex V2 provides integration with AWS Lambda, and you can integrate with many other services on the AWS platform, including Amazon Connect, Amazon Comprehend, and Amazon Kendra.

Amazon Lex V2 core concepts and terminology

1. Bot

- A bot performs automated tasks such as ordering a pizza, booking a hotel, ordering flowers, and so on.
- An Amazon Lex bot is powered by Automatic Speech Recognition (ASR) and Natural Language Understanding (NLU) capabilities.
- Amazon Lex bots can understand user input provided with text or speech and converse in natural language.
- You can create Lambda functions and add them as code hooks in your intent configuration to perform user data validation and fulfillment tasks.
- Each bot must have a unique name within your account.

2. Intent

- An intent represents an action that the user wants to perform
- You create a bot to support one or more related intents. For example, you might create a bot that orders pizza and drinks.
- For each intent, you provide the following required information:
    - Intent name– A descriptive name for the intent. For example, OrderPizza. Intent names must be unique within your account.
    - Sample utterances – How a user might convey the intent. For example, a user might say "Can I order a pizza please" or "I want to order a pizza".
    - How to fulfill the intent – How you want to fulfill the intent after the user provides the necessary information (for example, place order with a local pizza shop).

3. Slot

- An intent can require zero or more slots or parameters. You add slots as part of the intent configuration
- At runtime, Amazon Lex prompts the user for specific slot values. The user must provide values for all required slots before Amazon Lex can fulfill the intent.
- For example, the OrderPizza intent requires slots such as pizza size, crust type, and number of pizzas.
- In the intent configuration, you add these slots.
- For each slot, you provide slot type and a prompt for Amazon Lex to send to the client to elicit data from the user.
- A user can reply with a slot value that includes additional words, such as "large pizza please" or "let's stick with small."
- Slot type –
    - Each slot has a type.
    - You can create your custom slot types or use built-in slot types.
    - Amazon Lex also provides built-in slot types. For example, AMAZON.NUMBER is a built-in slot type that you can use for the number of pizzas ordered.
    - Each slot type must have a unique name within your account.
    - For example, you might create and use the following slot types for the OrderPizza intent:
    - Size – With enumeration values Small, Medium, and Large.
    - Crust – With enumeration values Thick and Thin.

Amazon Lex: How It Works

Following are the typical steps you perform when working with Amazon Lex:

- Create a bot and configure it with one or more intents that you want to support. Configure the bot so it understands the user's goal (intent), engages in conversation with the user to elicit information, and fulfills the user's intent.
- Test the bot. You can use the test window client provided by the Amazon Lex console.
- Publish a version and create an alias.
- Deploy the bot. You can deploy the bot on platforms such as mobile applications or messaging platforms such as Facebook Messenger.

Creation of AWS Account

- To create an AWS account
    - Open https://portal.aws.amazon.com/billing/signup.
- Follow the online instructions.
    - Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.
- Amazon Lex V2 provides API operations that you can integrate with your existing applications
- AWS SDK — When using the SDKs your requests to Amazon Lex V2 are automatically signed and authenticated using the credentials that you provide.

- AWS CLI — You can use the AWS CLI to access any Amazon Lex V2 feature without having to write any code.
- AWS Console — The console is the easiest way to get started testing and using Amazon Lex V2

Bot creation

- You create an Amazon Lex V2 bot to interact with your users to elicit information to accomplish a task.
- To build a bot, you need the following information:
- The language that the bot uses to interact with the customer.
- The intents, or goals, that the bot helps the user fulfill.
- The information, or slots, that you need to gather from the user to fulfill an intent.
- The type of the slots that you need from the user.
- The user interaction flow within and between intents

Lambda function

A Lambda function, in the context of cloud computing, typically refers to an AWS Lambda function. AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). It allows you to run your code without the need to provision or manage servers. You can use AWS Lambda to execute your code in response to events, such as changes to data in an Amazon S3 bucket, updates to a DynamoDB table, or HTTP requests via API Gateway.

Here are some key characteristics of AWS Lambda:

- Serverless:
  - With AWS Lambda, you don't need to manage servers.
  - The service automatically scales and provisions the required compute resources for your functions.

- Event-Driven:
  - Lambda functions are often triggered by events, such as changes in AWS services or HTTP requests.
  - You define the events that trigger the execution of your function.
- Support for Multiple Runtimes:
  - AWS Lambda supports multiple programming languages, allowing you to write your functions in languages such as Python, Node.js, Java, C#, and more.
- Scalability:
  - Lambda functions can automatically scale in response to incoming traffic.
  - Each function runs in isolation, and
  - multiple instances of the same function can run concurrently to handle increased load.

Here's a simple example of a Lambda function written in Python:

```
def lambda_handler(event, context):
  # Your code logic goes here
  return {
    'statusCode': 200,
    'body': 'Hello, this is a Lambda function!'
  }
```

This function responds with an HTTP status code of 200 and a simple body message when triggered. You can deploy this function and configure it to be triggered by various AWS services or events.

The **lambda_handler** function is the entry point for Lambda functions written in Python.

Here's a simple example of a Lambda function in Python:

```
def lambda_handler(event, context):
    # Your code logic goes here
    # You can access input data from the 'event' parameter
    # Perform your desired operations and return a response

    response = {
        'statusCode': 200,
        'body': 'Hello from Lambda!'
    }

    return response
```

- In this example, **the lambda_handler** function receives two parameters: **event and context**.
  - The **event** parameter contains the input data that triggered the Lambda function, and the **context** parameter provides information about the runtime environment.
  - Remember that you can customize the logic within the function based on your specific use case.

- The **response** should include a status code and a body.
  - Adjust the content of the response according to your requirements.

When deploying this Lambda function on AWS, make sure to set up the necessary permissions, triggers, and configurations based on your use

---

Code for lambda function

```
import json

def prepareResponse(event, msgText):
    response = {
        "sessionState": {
          "dialogAction": {
            "type": "Close"
          },
          "intent": {
            "name": event['sessionState']['intent']['name'],
```

```
            "state": "Fulfilled"
          }
        },
        "messages": [
        {
          "contentType": "PlainText",
          "content": msgText
        }
        ]
      }
    return response


def createIceCreamOrder(event):
    firstName = event['sessionState']['intent']['slots']['name']['value']['interpretedValue']
    iceCreamFlavor = event['sessionState']['intent']['slots']['flavor']['value']['interpretedValue']
    iceCreamSize = event['sessionState']['intent']['slots']['size']['value']['interpretedValue']

    discount = event['sessionState']['sessionAttributes']['discount']
    # Your custom order creation code here.
    msgText = "Your Order for, " + str(iceCreamSize) + " " + str(iceCreamFlavor) + " IceCream has been
placed with Order#: 342342"
    return prepareResponse(event, msgText)


def cancelIceCreamOrder(event):
    # Your order cancelation code here
    msgText = "Order has been canceled"
    return prepareResponse(event, msgText)


def lambda_handler(event, context):
    intentName = event['sessionState']['intent']['name']
    response = None

    if intentName == 'CreateOrderIntent':
        response = createIceCreamOrder(event)
    elif intentName == 'CancelOrderIntent':
        response = cancelIceCreamOrder(event)
    else:
        raise Exception('The intent : ' + intentName + ' is not supported')
    return response
```

Explanation of code lines

1) **intentName = event['sessionState']['intent']['name']**

    - This line of code is extracting the intent name from the event object.

- The event object likely contains information about the current session state, and within that,
- it's accessing the intent name.

---

2) firstName = event['sessionState']['intent']['slots']['name']['value']['interpretedValue']

- In an AWS Lambda function, you might receive an event object as an argument. The structure of this object can vary depending on the trigger or service invoking the Lambda function.
- This code snippet extracts the interpreted value of the 'name' slot from an event object.

---

3) return prepareResponse(event, msgText)

- we are using a function named prepareResponse to generate a response, and returning the result of this function.
- The prepareResponse function likely takes the event and msgText as parameters.

---

4) **prepareResponse** function is for constructing a response object with a structure suitable for a conversation interface or chatbot.

```
def prepareResponse(event, msgText):
response = {
        "sessionState": {
         "dialogAction": {
           "type": "Close"
         },
         "intent": {
           "name": event['sessionState']['intent']['name'],
              "state": "Fulfilled"
         }
        },
        "messages": [
         {
           "contentType": "PlainText",
           "content": msgText
         }
        ]
    }
```

- In the context of a Lambda function handling a conversation or interaction, an intent of type "Close" typically signifies that the conversation or session is ending or reaching a conclusion.
- The "Close" type is commonly used when you want to fulfill a user's request, provide a final response, and indicate that the interaction is complete.
- You have to understand few key points about an intent of type "Close" in a Lambda function:

Dialog Action Type:

- The **"type": "Close"** under "**dialogAction"** suggests that the system should close the current dialog or session.

Intent State:

- The **"intent"** block contains information about the intent, including its name and state.
- In our example, the "**state**" is set to **"Fulfilled,"** indicating that the intent has been successfully processed or fulfilled.

Session Completion:

- Using a type of **"Close"** often means that the Lambda function has completed the necessary actions based on the user's intent, and the conversation is ready to be concluded.

Response Messages:

- The response may include one or more messages (in the "messages" array) to provide information back to the user.
- In our example, there is a plain text message with the content specified by msgText.

In the lambda_handler function, the **return response** statement will return the response object to the entity that invoked the Lambda function.
This entity could be another **AWS service**, **an API Gateway**, or any other mechanism triggering the execution of the Lambda function.

What is virtual network

- A virtual network refers to a logically isolated network created within a physical network infrastructure.
- Virtual networks are commonly used in cloud computing environments and data centers to provide flexibility, scalability, and isolation for various applications and services.
- virtual network closely resembles a traditional network that you'd operate in our own data center, with the benefits of using the scalable infrastructure.

What is Private cloud

- A private cloud refers to a cloud computing environment that is used exclusively by a single organization.
- Unlike public clouds, which are shared by multiple tenants, a private cloud is dedicated to a specific business or entity. .
- This cloud model provides greater control, customization, and security for the organization's computing resources and services.

Virtual Private Cloud

- A private cloud, however, is single-tenant. A private cloud is a cloud service that is exclusively offered to one organization.
- A virtual private cloud (VPC) is a private cloud within a public cloud; no one else shares the VPC with the VPC customer.
- A virtual private cloud (VPC) is a secure, isolated private cloud hosted within a public cloud.

Amazon VPC

- VPC customers can run code, store data, host websites, and do anything else they could do in an ordinary private cloud, but the private cloud is hosted remotely by a public cloud provider.
- This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS.
- With Amazon Virtual Private Cloud (Amazon VPC), you can launch AWS resources in a logically isolated virtual network that you've defined.
- The following diagram shows an example VPC. The VPC has one subnet in each of the Availability Zones in the Region, EC2 instances in each subnet, and an internet gateway to allow communication between the resources in your VPC and the internet.

Features

1. **Virtual private clouds (VPC)**

   A VPC is a virtual network that closely resembles a traditional network that you'd operate in your own data center. After you create a VPC, you can add subnets.

2. **Subnets**

   A subnet is a range of IP addresses in your VPC. A subnet must reside in a single Availability Zone. After you add subnets, you can deploy AWS resources in your VPC.

3. **IP addressing**

   You can assign IP addresses, both IPv4 and IPv6, to your VPCs and subnets. You can also bring your public IPv4 addresses and IPv6 GUA addresses to AWS and allocate them to resources in your VPC, such as EC2 instances, NAT gateways, and Network Load Balancers.

4. **Routing**

   Use route tables to determine where network traffic from your subnet or gateway is directed.

5. **Gateways and endpoints**

   A gateway connects your VPC to another network. For example, use an internet gateway to connect your VPC to the internet. Use a VPC endpoint to connect to AWS services privately, without the use of an internet gateway or NAT device.

Who Needs a VPC?

Organizations that benefit most from VPCs are companies that need a private cloud environment but also want public cloud resources and savings.

Benefits of Using a VPC

1. Minimize downtime

   Although it's not always possible, customers expect 100% uptime and have little patience for any downtime – not even ten minutes. VPC environments provide the redundancy and other features required to meet near-100% uptime expectations.
   With nearly 100% uptime, your customers will experience a high level of reliability that will strengthen loyalty and trust in your brand.

2. Reduced risk

   A VPC will provide you with high security at the instance and subnet level.

3. Flexibility

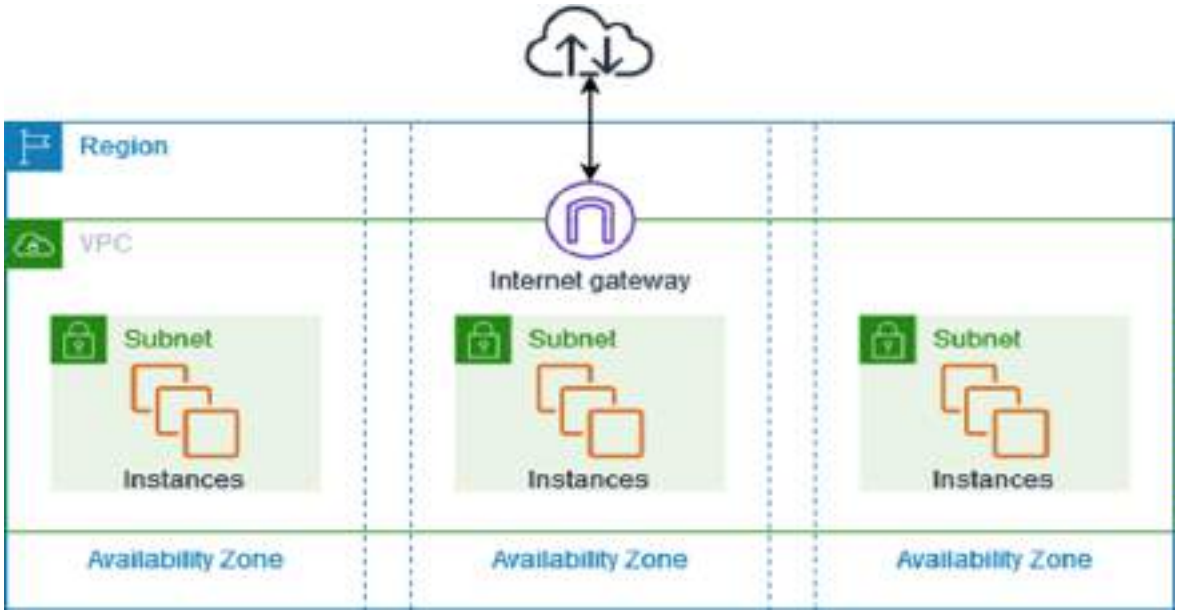
   Whether your business is growing or changing, VPCs are flexible enough to move with your business as needed. Cloud infrastructure resources are deployed dynamically, which makes it easy to adapt a VPC to your changing needs.

4. Cost savings

   Because of the elastic nature of public clouds, you only pay for what you use. Wth a VPC, you won't need to pay for hardware or software upgrades and you'll never pay for maintenance.

Amazon Elastic Compute Cloud (Amazon EC2)

- Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers.
- An Amazon EC2 instance is a virtual server in Amazon's Elastic Compute Cloud (EC2) for running applications on the Amazon Web Services (AWS) infrastructure
- AWS is a comprehensive, evolving cloud computing platform; EC2 is a service that enables business subscribers to run application programs in the computing environment
- It can serve as a practically unlimited set of virtual machines (VMs).
- With Amazon EC2, you can set up and configure the operating system and applications that run on your instance
- Amazon provides various types of instances with different configurations of CPU, memory, storage and networking resources to suit user needs. Each type is available in various sizes to address specific workload requirements
- Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment.
- Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change.
- Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use.
- The following diagram shows a basic architecture of an Amazon EC2 instance deployed within an Amazon Virtual Private Cloud (VPC).



- In this example, the EC2 instance is within an Availability Zone in the Region.
- The EC2 instance is secured with a security group, which is a virtual firewall that controls incoming and outgoing traffic.
- A private key is stored on the local computer and a public key is stored on the instance. Both keys are specified as a key pair to prove the identity of the user.
- The VPC communicates with the internet using an internet gateway.

- **Benefits**

1. Elastic Web-Scale Computing
  - Amazon EC2 enables you to increase or decrease capacity within minutes, not hours or days.
  - You can commission one, hundreds, or even thousands of server instances simultaneously

2. **Completely Controlled**
  - You have complete control of your instances including root access and the ability to interact with them as you would any machine.
  - You can stop any instance while retaining the data on the boot partition, and then subsequently restart the same instance using web service APIs.
  - Instances can be rebooted remotely using web service APIs, and you also have access to their console output

3. Flexible Cloud Hosting Services
  - You have the choice of multiple instance types, operating systems, and software packages.
  - Amazon EC2 allows you to select a configuration of memory, CPU, instance storage, and the boot partition size that is optimal for your choice of operating system and application.
  - For example, choice of operating systems includes numerous Linux distributions and Microsoft Windows Server.

4. Integrated
  - Amazon EC2 is integrated with most AWS services such as Amazon Simple Storage Service (Amazon S3), Amazon Relational Database Service (Amazon RDS), and Amazon Virtual Private Cloud (Amazon VPC) to provide a complete, secure solution for computing, query processing, and cloud storage across a wide range of applications.

5. Reliable
  - Amazon EC2 offers a highly reliable environment where replacement instances can be rapidly and predictably commissioned.
  - The service runs within Amazon's proven network infrastructure and data centers.
  - The Amazon EC2 Service Level Agreement commitment is 99.99% availability for each Amazon EC2 Region.

6. Secure
  - Cloud security at AWS is the highest priority.
  - As an AWS customer, you will benefit from a data center and network architecture built to meet the requirements of the most security-sensitive organizations.
  - Amazon EC2 works in conjunction with Amazon VPC to provide security and robust networking functionality for your compute resources.

1. Elastic Block Store (EBS)

- We will focus on three important topics which comes under EBS
    1. EBS volume
    2. Amazon EBS snapshot
    3. Data life cycle manager
- EBS stands for Amazon Elastic Block Store. It's a block-level storage service provided by Amazon Web Services (AWS) that's designed for use with Amazon Elastic Compute Cloud (EC2) instances
- EBS is hard disks which have a root volume. Root volumes are storage in which an operating system is already present inside
- As the name suggests it stores the data in a block form which are suitable for databases that requires frequent reads and write
- These block store volumes are external block storage means they are not directly attached storage
- EBS volumes are attached to your EC2 instance to the AWS network like virtual private cloud and both EBS volume and EC2 instance must be in the same availability zone

Features of EBS

1. **Block-Level Storage:** EBS provides persistent block-level storage volumes for use with EC2 instances. These volumes behave like raw, unformatted block devices that can be attached to EC2 instances and used as storage drives.
2. **Elasticity and Scalability:** EBS volumes can be easily created, attached, and detached from EC2 instances. You can also scale the size and performance characteristics of your EBS volumes dynamically to meet the changing needs of your applications.
3. **Data Persistence**: EBS volumes are designed for durability and data persistence. They are replicated within their Availability Zone to protect against component failure, and you can also create snapshots of your volumes to back up your data to Amazon Simple Storage Service (S3) for long-term storage.
4. **Durable snapshots :** EBS offers durable snapshot capabilities. It is possible because EBS volumes are placed in a specific availability zone where they are automatically replicated to protect you from the failure

Amazon EBS Volume Types

There are different types of EBS volumes. The first type is

1. Solid State Drive (SSD) Volume
2. Hard Disk Drive (HDD) Volume
3. Magnetic Standard (MS) Volume

1. **Solid State Drive (SSD) Volume**
    - They are categorized into 2 parts a) **General Purpose SSD** b**) Provisioned IOPS SSD**
    - **General Purpose SSD :**
        - Provides the balance between both pricing and performance means you can get balance in both iops and throughput
        - Here volume size will be between 1 gigabytes to 16 gigabytes with a throughput of aroud 1000 megabytes per second
        - They are by default commonly used
    - **Provisioned IOPS SSD**
        - It provides high performance for mission critical low latency or high performance throughput. Throughput is your mb per seconds or megabytes per second which show you how fast your data is being transferred
        - Their volume size is in between 4 gigabytes to 64 terabytes with a through put of around 4000 megabytes per second. You will get the highest iops here. Iops are generally used when you require a lot of read and write operations or transactions

2. **Hard Disk Drive (HDD) Volume**
    - They are non-bootable drives which means you can install an operating system in this drive
    - They are useful when you require an extra storage like a d drive , e drive or an f drive
    - They are categorized into a) **Throughput Optimized HDD or (ST1)** b**) Cold HDD or (SC1).** Both are thoroughput intensive now but st1 is more optimized than sc1 intensive
    - **Throughput Optimized HDD (ST1)**
        - They are non bootable

- It is backed by hard didk drive and is ideal for frequent access to put intense workload with large datasets
- St1 volumes deliver high performance in terms of throughput and they are also measured in mb per second
- Volume size of an HDD st1 is 125 MB to 16 gigabyte with a maximum iops of 500 per volume and maximum throughput per volume is around 500 megabytes per second

- **Cold HDD (SCI)**
  - It is also backed by an HDD or a Hard Disk Drive now
  - It is also non bootable
  - They provide the lowest cost per gigabytes of all EBS Volume types but they are used when data Is not accessed frequently
  - It is used in file servers
  - Volume size is around 125 GB to 16 terabytes with maximum iops of 250 per volume and maximum throughput of around 250 megabytes per second

3. **<u>Magnetic Standard (MS) Volume</u>**
   - Of all the bootable volumes Magnetic standard is the cheapest among others
   - They are ideal for workload when data is accessed infrequently and application where the storage cost is more important
   - Volume size is around 1GB to 1 terabytes of space with a maximum iops of 40 to 200 per volume and the maximum throughput of around 40 to 90 megabytes per second

2. Amazon EBS snapshot

Why do we need snapshot and what is snapshot?

- EBS Snapshot are similar to taking a snapshot of your screen in your phone or PC. The output of the snapshot will be nothing but your snap you have taken at that instance

Why do you want to take a snapshot?

- The moment you take snapshot then all the data inside the volume will come in a snapshot
- The different things you can do with snapshot like store or restore data then copy the data of your snapshot or you can even delete those snapshot which you have taken

What is EBS Snapshot?

- They **are point-in-time images or copies** of your EBS volume. Point-in-time means that only the data being snapped at that instance will be taken and store in your volume
- You can create upto **5000 volumes per aws** account and you can have up to **10000 EBS** snapshot per account
- EBS volumes are availability **zone specific** means EC2 instance and Volume will also be in same zone but EBS Snapshot are **region specific** which means that they can be in availability zone1 and availability zone 2 and availability zone 3 as Snapshot is in S3 which means all the region can access it

Features of snapshot

- With Amazon EBS snapshot shareabiltiy is made easier. We can share data with co-workers and others in aws community
- **Resizing Amazon EBS volumes**: If you create a new volume base on a snapshot you can specify a larger size for the new volume

3. EBS Data Lifecycle Manager

Next we will see the new service provided by Amazon EBS is data lifecycle Manager. It will make it easier for us to schedule your EBS Snapshot. Let us understand **why do we need to use an EBS Lifecycle Manager.**

You must have noticed that whatsapp backup your data every single day at sometime. So, if u ask why do they do that suppose you change your phone or your phone have been damaged or whatsapp crashes and you haven't scheduled your backup so all the data that is within that device will all be lost. So what is the solution. In order to prevent these data loss whatsapp recognize this issue and solve it by backing up your data daily or weekly or even monthly. Similarly EBS Lifecycle manager work in the same way by backing up your snapshots. The result that came out by backing up your data whatsapp user are now able to retrieve those data even if they change their phone or they lose their phone or even if the whatsapp crashes.

In the same manner EBS Lifecycle manager backs u huge data in big corporation where data are sensitive. If we take an example of some online course provider's website there may be hundreds of people enrolling the course in Aws in a day. What if the server crashes and organization didn't backup the data then they might have lost data and organization wont be able to find out which person have enrolled for the course

Now, that's why backing up the data is very important . With EBS lifecycle you can schedule and backup those data even if the server crashes your data wont be at loss

What is EBS Lifecycle

- With EBS Lifecycle you can back up or schedule your data of your screenshots
- Snapshots are cleaned up regularly
- EBS Lifcycle uses cloud watch events to monitor your policies

Amazon Elastic File System

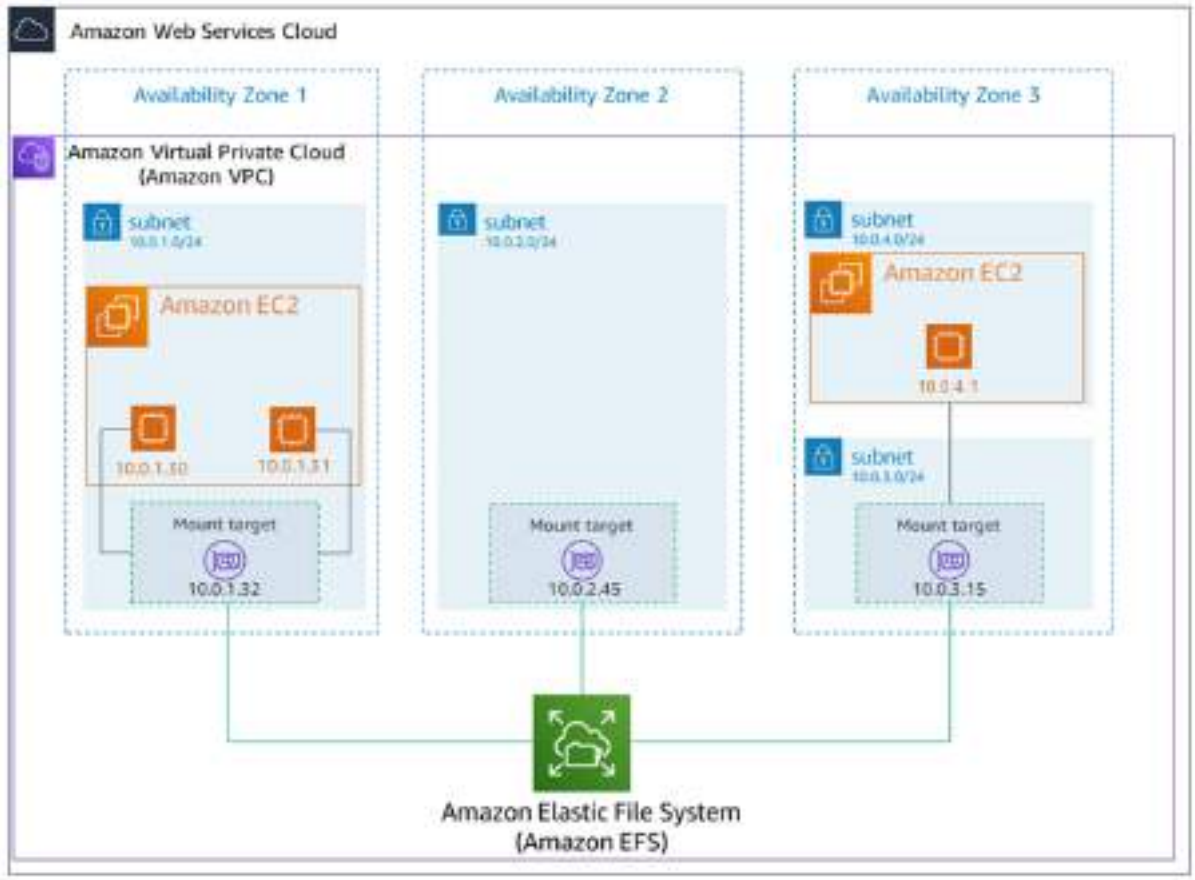
What is Amazon Elastic File System?

- EFS stands for Amazon Elastic File System.
- It is a scalable, fully managed file storage service provided by Amazon Web Services (AWS).
- EFS is designed to provide simple, scalable file storage for use with AWS Cloud services and on-premises resources.
- EFS is a simple serverless and elastic file system that lets you share file data without provisioning or managing storage.
- Amazon Efs uses the industry standard Network File System for file access protocol. So the application you use works seamlessly

When to use EFS

- Amazon supports **File, Object** and **Block** level storage. For file storage you can use Elastic File System. For Object storage you can use AWS S3 and For Block storage you can use AWS EBS
- EFS can be used for application which require a shared File systems that can be accessed by multiple computers at the same time .
- It can be used by the application that can access data using the standard file system interface provided through the OS
- They can take advantage of the scalability and reliability of storage in cloud without writing any new code or adjusting the applications Which is not possible in other two storages types
- It can be used as a shared file system with EC2 Instances .Applications running on multiple EC2 instances can access the file system at the same time
- In EFS thousands of EC2 instances or on premises servers from multiple availability zones can concurrently access the file system
- But in EBS only a single EC2 instance in a single availability zone can access the data

In this illustration, the virtual private cloud (VPC) has three Availability Zones. Because the file system is Regional, a mount target was created in each Availability Zone. We recommend that you access the file system from a mount target within the same Availability Zone for performance and cost reasons. One of the Availability Zones has two subnets. However, a mount target is created in only one of the subnets.



Benefits

1. **Dynamic Elasticity:** Amazon EFS automatically scales the file system storage capacity up or down as u add or remove files without disrupting your applications. It dynamically provides the storage capacity u need. You can create your file system and start adding files without provisioning storage in advance
2. **Fully Managed:** It is fully managed by AWS. Amazon EFS provides a shared file system for linux workloads . It provides a simple interface allowing you to create and configure the file system quickly and Amazon manages the file storage infrastructure for you. It will remove the complexity of deploying , patching and maintaining the under planning of a file system

3. **Cost Effective:** With Amazon EFS you only pay for what you use. There is no need to provision in advance and there is no minimum commitment or upfront fees. With EFS life cycle management you can automatically move files access less frequently to a cost optimized storage class which will reduce the file storage costs by 92%
4. **Security and Compliance:** Amazon EFS allows you to securely access your file using an exsisting security infrastructure. You can control access to Amazon EFS file system with Amazon VPC and AWS IAM. You can secure your data by encrypting your data at rest and in transit

What is AWS S3

- S3 stands for Simple Storage Service
- S3 is an Object storage service provided by AWS that stores data as objects within buckets
- An object is a file and any metadata that describes the file . A Bucket is a container for objects
- Here basically files are stored in buckets. Bucket is nothing more than a folder in which files are stored. You can have more than one bucket in your AWS account
- Within a bucket you can break down to different sub directories . you can have as many as many different subdirectories as you want and then those subdirectories can then store the files and the files that you store in your bucket each file is going to be referred to as an object and so that's why it's called a object-based storage

- S3 supports a Max file or object size of 5 terabytes and you can upload unlimited amounts of files to S3 the only limitation really is that the file can't be larger than five terabytes
- S3 provides data protection it will actually replicate your objects or files on multiple devices across a availability Zone which is just a data center
    - In addition to that it's actually going to get replicated across multiple availability zones as well so you can actually handle losing an entire data center or the entire data center going down and your files will still be intact and you'll still be able to access them so that's the data protection that AWS provides with S3
- **Working:** To store your data in Amazon S3
    - First create a Bucket and specify a bucket name and AWS region
    - Upload your data to that bucket as objects

- **Bucket**
  - A Bucket is a container for objects
  - when you create a S3 bucket you're going to have to give it a name and the name has to be unique globally across all AWS accounts
  - when a S3 bucket gets created they'll actually designate and reserve a URL for you so that you can access the files within that bucket. So these are our URLs they're going to have to be unique
  - so once we actually create a bucket how do we **access the files within the bucket** how do we add new files to the bucket well there's three different ways to do
    - **first one** is to use the URL so this is going to be the format of the URL https://bucket-name.s3.region-code.amazonaws.com/key-name
    - The **second one** is you can always use the console so you can go in you can upload files you can download files and
    - Then **lastly** we could do it programmatically through code so you can use the S3 SDK to manipulate files add new files new folders and things like that
- **Object**
  - Amazon S3 is an object store that uses unique key- values to store objects
  - An **object comprised** of three things three pieces of information
    - key - represents the file name. Used to retrieve the object
    - value - file data. The content you are storing
    - version ID- It is a string that Amazon S3 generates when you add an object to bucket . Within a bucket , a key and version ID uniquely identify an object

- S3 offers a range of **different storage classes** that are going to provide us a varying level of data access resiliency with different levels of cost
  - **S3 standard:** this is the one that's most frequently used this is offers High availability and durability for frequently accessed data with low latency obviously anytime you need to get files very

quickly you're going to want to use S3 standard .It has a very high durability that's spread across multiple availability zones the overall availability is 99.99 over a given year

- **S3 intelligent tiering** :The most optimal storage class . This is the best storage class that's going to be for your application or your business you may want to use the intelligent hearing which will actually analyze how your data is being used in Access and will actually move you into the storage class that is the most optimal and cost effective for the way you retrieve your data so if you don't know what you want to do or you don't know which storage classes it's best to probably select intelligent tiering so that AWS can figure it out intelligently for you
- **S3 standard –IA(S3 standard Infrequent Access):** This is for data that doesn't have to be accessed frequently and because it doesn't have to be accessed frequently the overall fee is going to be less than standard S3.However if you do retrieve this data which doesn't have to be they're going to charge a retrieval free it's going to have the same latency and throughput as S3 standard and it has a 99.9 percent availability
- **S3 One Zone-IA (S3 one zone Infrequent Access**): This is the same as S3 standard IA but it doesn't require data to be replicated across multiple availability zones. So if you don't need that extra durability of having it across multiple availability zones then this is going to be a cheaper option because it's only spread across one data center
- **S3 Glacier Deep Archive:** This the cheapest storage class and this is meant for long-term storage retention that doesn't need to be retrieved quickly. So the retrieval time can take up to 12 hours. This is used when you have to store data and keep it for like seven to ten years you would want to just dump it on there so that you have the lowest cost

Migration of website to cloud using Docker

Developer develops the application(Frontend & Backend) and pushed to git hub. Git hub will have two repositories Frontend and backend.

These two repositories are going to be cloned into our EC2 instance as front end repository and backend repository. When we are cloning this we will be injecting two files named **Docker** which has six attributes and **.env** file.

To make it accessible to customers this front end and back end we are going to convert these folders or project code into **Docker images** one is called as **client** the other one is called as **server**

In server our backend code will be converted whereas in Client our front end code will be converted. To make it accessible we'll run this Docker image in a **container**. Once it is running in docker environment the application can be accessible by the customers all the clients whoever wants to.

This is what the entire structure is where developer is developing some code he will be pushing into GitHub from GitHub we will be cloning onto our EC2 instance. In EC2 instance we will be installing our Docker and in the docker we will be converting the projects into images and in the docker containers we'll be running the applications. This is how a application running on a local server is been deployed or migrated into a cloud environment

so let's have an Hands-On session on how to migrate a website from local server to Cloud . We are in our Cloud AWS environment where the first thing is you need is to search for an EC2. Just click on EC2 instance . we will launch an EC2 instance I am naming this instance as **Dockerdemo** . Selecting the operating system as **Ubuntu** enhancing my **T2 micro** instance type to **T2 large.** Creating a keypair I am naming it as same **DockerDemoKey**

In network settings I will modify VPC subnet. **Auto-assign public IP** I will enable it and will add security group by clicking **Add security group rule** change **Type** to allowing **All traffic** and in **Source type** to **Anywhere.** In **Configuration storage** enhancing my storage to 30 GB of capacity and now click on **Launch instance**

Once the instance has launched just wait until it turns into running State. Once the instance state gets into running click on **Instance ID** and click on **Connect**. Under SSH will be copying the link to get connected to EC2 instance.

Open the command prompt move to downloads my pem key has got downloaded under downloads . Now continue connecting to EC2 instance . Got connected and could see at certain IP address

Now the first thing will do is cloning my backend repository from GitHub using the command **git clone** https://github.com/procareer3fwd/realgrandebackend.git which is URL of your GitHub repository. Once this has been done will be be cloning frontend from GitHub using command get clone **git clone** https://github.com/procareer3fwd/realgrandefrontend.git. Now you could see that your realgrande back end as well as realgrande front end has got downloaded

One thing before installing of Docker you need to update your ubuntu because as ubuntu is an open-source operating system it's keep on updating its libraries and packages so to be updated we need to update our operating system which is done using command **sudo apt update** .

Next it's time to install your Docker which is done using command **sudo apt -y install docker.io.** This command installs your Docker on your EC2 instance. Once Docker has been installed just cross verify whether Docker has been properly installed using command **sudo docker version**. If you could see **client** and **server** then it indicates that Docker has installed properly.

Now check whether you do have any images in your Docker or not using command **sudo docker images. D**on't have any docker images . To have docker image we need to inject **.env** and **Dockerfile** So the first thing is to get into your backend directory by typing **cd realgrandebackend/** and in backend you need to inject a **.env .** First create .env file by using command **nano .env** and paste these lines

MONGODBURL="mongodb+srv://fsd04.2hxrdca.mongodb.net/realgrande?retryWrites=true&w=majority"
DBUSERNAME=procareer3
DBPASSWORD=ISobjBDohsFqEAqg
FRONTENDURI="http:// 3.82.247.96"

and do remember you need to change your public IP address over here which you will get into from your EC2 instance. Get your public IP from EC2 instance which is **DockerDemo** for me here . After pasting in .env file please type ctrl+ X hit Y and hit enter to save and quit

So I was talking about the docker file just type command **ls** to see it. It is already present in your real grande backend as well as frontend this is the file. Now type **cat Dockerfile**. Here we do have six attributes all the attribute names will be in upper case

1. **FROM node** it means that to run your react applications you require some sort of environment where operating system as well as the middle Ware should be present.node which has nodejs in it with the operating system system , Alpine is the operating system and nodejs is the environment or server on which your application is going to run.So in order to just use this what I'm doing is **From node.**

   what Docker file will do is it will search its local machine where node image is present if it is not then it will get into dockerhub and it pulls this particular image dockerhub which has its own operating system named Alpine and above that at the middleware which is running that is nodejs
2. The second attribute creating my **own workspace** where all the files are going to be copied under this workspace
3. Then I will be installing my **npm** (node package manager)
4. In the fourth command I'm **exposing** this back end on the port number 5,000 and will be running
5. I will be starting this particular application using **npm start**

These are the six attributes which are going to be mentioned in my Docker file

Now how to build the image presently we don't have any images in our Docker how to build an image use command **sudo docker build -t backend .**

It means should build based on a Docker file which is present in the current directory hit enter you can see that it is pulling an image from dockerhub named node pull complete you can see this is step one of six let us wait till it completes all the Steps

Step two it has created its own working directory / app step two it is creating it has created yeah it has Exposed on port number and it has started its image

Let's see content image running in a container using command **sudo docker ps** .Presently not running. To make that image to run in this container I'll be using the command

sudo docker run -d -p 2001:5000 backend

-d means in detachable mode **-P** the port on which it has to be exposed I can say it as 2001 colon 5000 and which image you want to run in the container the name of the image is backend as which we had created just now the backend successfully tagged backend

Just check it out whether that is running in a container using command **sudo docker ps** . yes it has its **own container ID** and the name of the image which is running in the container is **backend** it is been exposed on port number 2001 and 5,000 is a port which is internal if I want to access then I need to use 2001 port number. It is Port binding concept

So let's check whether we can connect to our backend through a browser paste the public IP **3.82.247.96:2001/api** hit enter . Yeah we can see Json file it means that we could connect to our backend repository

Let's redirect to our frontend now to convert our front end into an image. Type **cd** and type **cd realgrandefrontend.** So first thing is we have to check whether we do have Docker file type **ls.** yes we do it also . Now type **cat Dockerfile** we see has the same six attributes but this will be exposed on port number 3,000 the frontend

Let's create a .env file in our front end and paste **REACT_APP_BACKEND_URL=http://3.82.247.96:2001/api** this line . I have changed you can see that I have been given 2001 my back end is been exposed on port number 2001 so if you are exposing any other Port then you need to mention over here . ctrl+ X and Y hit enter

once the .env file is been done then you can build your images .So Docker build

sudo docker build -t frontend .

This time your node has been already downloaded in the image node so it has not taken too much time it will check its local repository. node image is already present in its local repository. So it will not again get into the docker Hub and keep on downloading the node image because in our previous back end itself we had downloaded our node image we had pulled it from the docker Hub once this has been done then just run this frontend image and check whether you can accessyour front end or not by typing **sudo docker images.** Now my image has been successfully created . If you want you can check it it through images . You can see **frontend** image and **node** image and **backend** back end and **frontend** we had created node we had pulled it from the docker Hub

Just run it by using command **sudo docker run -d -p 80:3000 frontend** and port on which you are going to expose is 80:3000 is a port and which image you want to run , you want to run the frontend image hit enter. Just check it out whether that has been running using **Sudo docker ps** . You can see frontend has been running with a **container ID** and it is exposed on port number **80**

let's get back to our browser copy the public IP address **3.82.247.96:2001**hit enter as at is the default Port I need not to mention my port number you can see you can access your front end you can retrieve the data just like can't all these data are being retrivied fromthe mongodb database.You can login, you can sign up everything

Steps to Migrate a website from local server to Cloud

==========

1) Launch an EC2 Instance on AWS console
2) Open the command prompt and Connect to EC2 Instance

3) clone the git repos

backend

==========

tinyurl.com/cs1bekmit

git clone https://github.com/procareer3fwd/realgrandebackend.git

frontend

==========

tinyurl.com/cs1fekmit

git clone https://github.com/procareer3fwd/realgrandefrontend.git

4) Update the Ubuntu

sudo apt update

5) Install the Docker

sudo apt -y install docker.io

6) Check the docker images

sudo docker images

7) change the directory to backend

cd realgrandebackend/

8) create an .env file

nano .env

9) Copy Paste the lines in .env file

MONGODBURL="mongodb+srv://fsd04.2hxrdca.mongodb.net/realgrande?retryWrites=true&w=majority"

DBUSERNAME=procareer3

DBPASSWORD=ISobjBDohsFqEAqg

FRONTENDURI="http://3.82.247.96"

10) Build the docker image for backend

sudo docker build -t backend_server .

11) Check for any images running in the container

sudo docker ps

12) Now run the backend_server docker image in the container

sudo docker run -d -p 2001:5000 backend_server

13) Now try to access the backend_server on the browser

<EC2_PUBLIC_IP_ADDRESS>:2001/api

Eg: **3.82.247.96:2001/api** // 3.82.247.96 is the public ip of my EC2 instance

14) Now change the directory to frontend

ubuntu@ip-172-31-1-17:~/realgrandebackend$ cd

ubuntu@ip-172-31-1-17:~$ cd realgrandefrontend/

ubuntu@ip-172-31-1-17:~/realgrandefrontend$

15) Now create a .env file

nano .env

16) Copy paste the line in .env file

REACT_APP_BACKEND_URL="http:// 3.82.247.96 :2001/api"

17) Build the docker image for frontend

sudo docker build -t frontend .

18) Check for any images running in the container

sudo docker ps

19) Now run the backend_server docker image in the container

sudo docker run -d -p 80:3000 frontend

20) Now try to access the frontend on the browser

<EC2_PUBLIC_IP_ADDRESS>

Eg: **3.82.247.96** //This is public IP of my EC2 instance

Docker

- Docker is an open-source platform that allows developers to build and run applications in containers.
- Containers have become a de-facto standard for deploying applications in a microservice fashion.
- Containers are lightweight and portable, which encapsulate the application and its dependencies, such as libraries, frameworks, and runtime required to run the application making them platform agnostic.
- This enables developers to deploy the same application across different environments, such as development, QA, testing, and production.

Docker Architecture:

Docker consists of 3 main components:

- **Docker Engine**: the core runtime that runs containers and manages their lifecycle.
- **Docker Client:** the command-line interface (CLI) that interacts with the Docker Engine and sends commands and requests.
- **Docker Registry:** the central repository that stores and distributes Docker images, which are the building blocks of containers. A few examples are Docker Hub where all the public images are maintained, or cloud-based private docker registries like ACR (Azure Container Registry), GCR (Google Container Registry), etc.

What is Docker CLI?

- Docker CLI (Command-Line Interface) is a tool that allows developers and system administrators to interact with the Docker Engine.
- This includes building images from Dockerfiles, running containers from images, managing container networks and volumes, and interacting with the Docker registry.
- Docker CLI can be installed on both Linux systems and on Windows as a Docker desktop.

Let us start with the basic commands.

1) **docker pull**
    - The first command we will try is to pull a public docker image from Dockerhub.

      docker pull [IMAGE_NAME]:[TAG]
      Example: docker pull nginx:latest

    - This will pull the latest docker image from the DockerHub.
    - The image will be stored in our local system.
    - We can specify specific tags if we want to pull older versions of the image.
    - If no TAG is specified, it is considered at the latest.

2) **docker images**
    - We can see all the images in our local system by using the **docker images** command.
    - Every image gets an IMAGE_ID that can be seen in the output of this command.
    - The **-a** flag is used to view all the images available on the local machine.

      docker images -a

      Output:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	6efc10a0510f	11 days ago	142MB

3) **docker run**
    - Once the docker image is pulled we can run them as a container.

      docker run nginx
    - This shows that the Nginx container is running on the local machine using Docker Engine.

4) **docker stop**
    - We will now stop our running container.

- We will pass the CONTAINER_ID to stop the container.
- The container goes into an EXITED state after it is stopped.

docker stop [CONTAINER_ID]

Eg: docker stop 62814eb60c1c
Output: 62814eb60c1c

5) docker ps

- We can list the containers using the docker ps command.
- This will show us the state of the container — Running, stopped, exited, etc. Every container also gets a CONTAINER_ID as we get for an image.
- The **-a** flag is used to list all the containers on the local machine.

docker ps -a

Output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
62814eb60c1c	nginx	"/docker-entrypoint.…"	2 minutes ago	Up 2 minutes	80/tcp	boring_zhukovsky

6) docker run (as a daemon)

- We can run a container in the background as a daemon using the -d flag.
- This will make the container run in the background.
- On running the docker ps command we can see the container as up and running.

docker run -d [IMAGE_NAME/IMAGE_ID]

Eg: docker run -d 62814eb60c1c
Output:
15ab848abe7ac514135b2623dd0091087b37b96e7f1443f1f85afcf086025a51

docker ps -a

Output:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
15ab848abe7a	nginx	"/docker-entrypoint.…"	32 seconds ago	Up 31 seconds	80/tcp	relaxed_mendeleev

7) docker exec

- We can get inside a docker container and access the terminal of the container using the exec command.
- The -it command states we will use interactive mode to connect to the bash shell of the container.

docker exec -it [CONTAINER_ID] /bin/bash

eg: docker exec -it 15ab848abe7a /bin/bash
Output: root@15ab848abe7a:/#

The output shows we are accessing the shell inside the container. Use Ctrl+D or exit command to come out of the container.