27/02/2023

|  |  |
| --- | --- |
| **On Premises** | **On Cloud** |
| Higher pay and less scalability | Scale up-pay more / scale down-pay less |
| Allocate huge space for server | No Server space required |
| Appoint team for hardware and software maintenance | No team is required |
| Very less data security | High data security |
| Data recovery is very less | Disaster recovery (stores in 3 places) |
| Lack of Flexibility | High Flexibility |
| No automatic software update | Automatic software update |
| Data cannot extend or accessed remotely | Data can be shared or accessed anywhere over the internet |
| Takes longer implementation time | Takes rapid implementations |

**Definition**:

CLOUD COMPUTING is a delivery of on demand computing service over the internet on a pay as you go basis.

Types of cloud computing:

**Deployment model**

1-Public cloud

2-Private cloud

3-Hybrid cloud

**Service model**

1-IAAS-> Infrastructure as a Service

2-PAAS-> Platform as a Service

3-SAAS-> Software as a Service

**Deployment model:**

**Public Cloud:**

The cloud infrastructure made available to the general public over the internet and is owned by a cloud provider.

ex-> AWS, Microsoft azure, IBM, blue cloud, sun cloud.

**Private Cloud:**

The cloud infrastructure exclusively operates by a single organisation. It can be managed by an organisation or a third party and exist on premises or off premises

ex->AWS and VMware

**Hybrid Cloud:**

It consists of the functionality of both public and private.

ex-> Federal agencies, opt for private cloud when sensitive information is involved also, they use the public cloud to share datasets with general public or other government department.

**Service model:**

**IAAS**: only space-platform

1) IAAS is a cloud service that provides basic computing infrastructure

2)Service are available on pay for what you use model

3)IAAS provider includes AWS, Microsoft azure, google compute engine

Users are IT administrators.

**PAAS**: only space and requirements-platform and software

1)provides a cloud platform and runtime environment for developing, testing and managing application.

2)It allows software developers to deploy application without requiring all required infrastructures.

3) -

Users are software developer.

**SAAS**: only utilize-platform, software,

1)All software’s and hardware’s are provided and managed by the vendors. So, you don't have to maintain anything.

Users are customers.

|  |  |  |  |
| --- | --- | --- | --- |
| **On premises** | **IAAS** | **PAAS** | **SASS** |
| Application  Data  Runtime  Middleware  OS  Virtualisation  Server  Storage  Network | Application  Data  Runtime  Middleware  OS  **Virtualisation**  **Server**  **Storage**  **Network** | Application  Data  **Runtime**  **Middleware**  **OS**  **Virtualisation**  **Server**  **Storage**  **Network** | **Application**  **Data**  **Runtime**  **Middleware**  **OS**  **Virtualisation**  **Server**  **Storage**  **Network** |

**AWS**- Amazon Web Service

It provides service over internet

The major use is to create and deploy application and cloud.

AWS use subscription pricing model (pay for what you use).

1.Hardware - EC2, lambda, elastic container service

2.Storage – S3, EFS, Glacier

3.Network – Router55, direct connect, VPC

4.Security - IAM, KMS, Cognito

5.Management process tool – Cloud Watch, Auto Scaling, Cloud Formation

6.Testing - Code Star, Code Build, Code pipeline

7.Analytics – Athena, ENA, cloud service.

---------------------------------------------------------------------------------------------------------

28/02/2023

|  |
| --- |
| Virtualisation  Server  Storage -> EBS – Elastic block storage  Network |
| EC2 – Elastic compute cloud |

**AMI** – Amazon Machine Image

**Instance types** – 5 types

1. General purpose
2. Compute optimized
3. Memory optimized
4. Accelerates computing
5. Storage optimized

M5D.X Large

M -> Instance family 🡺 A, C, D, I, F, G, H, M, P, R, T, X

5 -> Which Generation 🡺 1,2,3,4,5,6

D -> Additional capability 🡺 nano, micro, small, medium, large, extra-large, 2X large, 4X large, 8X large, 10X large, 16X large,32X large

X -> Instance size

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| General  Purpose | Compute  Optimized | Memory  Optimized | Accelerated  Computing | Storage  Optimized |
| A | C | R | P | H |
| T |  | X | G | I |
| M |  | Z | F | D |

**General purpose:**

Use Case:

1. Web servers and code repository
2. Compute memory and networking

T2.micro free tire

**Compute optimized:**

Use Case:

1. Intensive task with high performance like Machine learning, batch processing workload, Dedicated gamin server, high performing web servers.

**Memory optimized:**

Use Case:

1. High performance relational and non-relational database
2. Distributed web scaled cache store
3. Application performing real time big unstructured data
4. Business Intelligence

**Accelerated Computing:**

Use Case:

1. Machine learning and deep learning
2. Graphic intensive and streaming
3. Hardware Acceleration

**Storage optimized:**

Use Case:

1. OLTP – Oline Transaction Processing

2. Relational and NoSQL databases

3. Cache in the memory database

4. DFS – Distributed File System

**Storage:**

EBS - Elastic Block Storage

S3 - Object level Storage

EFS - File level Storage

**EBS**:

Two types:

1. HDD - Hard Disk Drive
2. SSD - Solid State Drive

**Backup of EPS:**

Snapshot

🡺A single instance consists of multiple EBS where A Multiple instance cannot have a same EBS

* 1 Gib(gibibytes) = 1.07374 GB(Gigabyte)

**EIP** - Elastic Ip address

* Static IP address

**Region and Zones:**

**Commands:**

**[for ubuntu]**

Root user 🡺 sudo su

Update🡺 apt-get update

Apache 🡺 apt install apache2

Enable 🡺 systemctl enable apache2

Start🡺 systemctl start apache2

Next🡺 curl -sL https://deb.nodesource.com/setup\_14.x | sudo -E bash -

Node 🡺 sudo apt-get install -y nodejs

Npm 🡺 apt-get install npm

Package 🡺 cd /var/www/html

React 🡺 npx create-react-app myapp

{

sudo su

**#Update ubuntu instance**

apt-get update

**#Upgrade ubuntu system**

apt-get upgrade

**#Install apache2**

apt-get install apache2

systemctl enable apache2

systemctl start apache2

apt-get install curl

curl -sL https://deb.nodesource.com/setup\_14.x | sudo -E bash -

sudo apt-get install -y nodejs

cd /var/www/html

npx create-react-app myapp

cd myapp

npm run build

npm start

}

IAM 🡺 Identity and Access Management

1. User
2. User Group
3. Roles - designation
4. Policies - access

MFA 🡺 Multi Factor Authentication

User defined Policy

1. arn
2. aws
3. service
4. region
5. account-id
6. resource-type
7. resource-name

ex: -

arn:aws:ec2:::ec2:mywebapp

**S3** 🡺 **Simple Storage Service** – Object Based Storing

To store large sized files

1. Buckets: directory or drive in a system
2. Objects: every saved file
3. Key: unique links of an individual objects
4. Versions:
   * 1. backtrack or restore the previous version of the object
     2. You can retrieve the object even after deleting
5. Encryption: for security purpose of the object
6. Storage classes:
7. Static website hosting: in order to host and deploy a static webpage

**Encryption:**

🡺Server-side encryption

-SSE-S3 -> AES 256

-SSE-KMS-> key management encryption standard

-SSE-C -> Customer provided key based encryption

🡺Client-side encryption

Any standard can be used by us before uploading

🡺SSL or TLS encryption

It happens on network-based encryption

**Autoscaling:**

Autoscaling is the service that automatically monitors and adjust compute resources to maintain performance for application hosted in AWS.

As the demand spikes, AWS autoscaling services cab automatically scale those resources and if demand drops scale them back down.

The process of scaling up and scaling down is called as fleet management.

Launch Configuration

Autoscaling group

1. Launch configuration

-AMI

-Instance type

-Key pair

-Security group

-Block device mapping

1. Autoscaling group

-Minimum instances

-Maximum instances

load balancing and autoscaling

for autoscaling (AMI Id instance)

**Introduction to AWS**

* Introduction to cloud computing
* Services provided by AWS
* The future of AWS
* Creating an AWS account

**AWS EC2**

* Introduction to AWS EC2
* Creating an EC2 instance
* EBS snapshots
* Creating an AMI / my AMI
* EIP

**Amazon S3**

* Types and benefits of cloud storage
* Bucket and object permissions
* Static web hosting and object versioning
* Storage classes

**IAM**

* Introduction to security services
* IAM users, groups, roles, and MFA
* Types of policies in IAM

**AWS Autoscaling**

* Types of scaling policies
* How autoscaling works
* Launch configurations
* Autoscaling groups

**VPC**

* Networking Fundamentals
* VPC and its Components
* Creating VPC components
* Public and private subnets

**VPC** -> Virtual private cloud

(Maximum IAM user – 1000 users)

IP Address map from 0 to 255

192.1.0.0 to 192.1.0.255

192.1.1.0 to 192.1.1.255

255.255.255.255

255.173.121.5

255.256.121.6 – wrong

192.2.2.0 to 192.2.2.255

CIDR – Classless Inter Domain Routing

192.2.2.0/32 -> IP assigned 1,

192.2.2.0/31 -> IP assigned 2,

192.2.2.0/30 -> IP assigned 4,

192.2.2.0/29 -> IP assigned 8,

192.2.2.0/28 -> IP assigned 16,

192.2.2.0/27 -> IP assigned 32,

192.2.2.0/26 -> IP assigned 64,

192.2.2.0/25 -> IP assigned 128,

192.2.2.0/24 -> IP assigned 256

**FORMULA:**

32-30 = 2^2 = 4

32-24 = 2^8 = 256

192.2.2.0/23 🡺 192.2.2.0 to 192.2.2.255 **and** 192.2.3.0 to 192.2.3.255

Internet gateway

10.0.0.0/21 –> 2048

10.0.1.0/23 -> 512 🡺 10.0.2.255

10.0.3.0/23 -> 512 🡺 10.0.4.255

10.0.5.0/23 -> 512 🡺 10.0.6.255

Load Balancer -> separate load for multiple instance

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#! /bin/bash

apt-get install apache2

systemctl start apache2

systemctl enable apache2

echo “<h1> This is my application $(hostname -f)</h1>” > /var/www/html/myapp/index.html

#! /bin/bash

apt-get install apache2 -y

systemctl start apache2

systemctl enable apache2

mkdir /var/www/html/myapp echo "<h1>Hello World from $(hostname -f)</h1?">/var/www/html/myapp/index.html

1. Create a load balancer with a target group – application load balancer
2. Create an auto scaling with desired two instance
3. Load the load balancer to the Auto Scaling

**Dockers:**

1. Dockers is a software development platform to deploy apps
2. Apps are package in contains that can run on any OS
3. Apps run the same regardless of where they run
   1. They run on any machine
   2. No compatibility issues
   3. Predictable behavior
   4. Less work and easier to maintain and deploy
   5. Works with any OS, any technology and any language

NODE

Docker



MYSQL

EC2 Instance -> Ubuntu

Where are Docker image stored?

Docker images are stored in docker repositories

**Types**:

1. Docker hub 🡺 Public Repository -> (Find base images of many technologies and OS)
2. Amazon ECR (Amazon Elastic Container Registry)🡺 Private Repository -> (Amazon ECR public gallery)

JAVA FULLSTACK

REACT PROJECT

KALLI LINUX

UBUNTU

Hypervisor

WINDOW 11

DELL – 8GB RAM – INTEL5

--------------------------------------------------------------------------------------------------------

CONTAINER

CONTAINER

CONTAINER

CONTAINER

DOCKER

EC2 INSTANCE

INFRASTRUCTURE

DOCKER CONTAINER

DOCKER IMAGE

DOCKER FILE

DOCKER REPOSITORY

DOCKER HUB AND   
AMAZON ECR

**Docker Container Management**:

4 types:

1. Amazon ECS (Amazon Elastic Container Service) – AWS own container platform.
2. Amazon EKS (Amazon Elastic Kubernetes Service) – Open Source
3. AWS far gate – It is called as own serverless container platform
4. Amazon ECR (Amazon Elastic Container Registry) – Store container image

Create instance

apt-get update

apt-get install nginx

apt-get install docker.io

docker –version

mkdir myproject

cd myproject

nano index.html

<h1>hi hello </h1>

Ctrl o & ctrl x

nano Dockerfile

nano Dockerfile

FROM nginx:alpine

COPY ./index.html /usr/share/nginx/html/

Sudo docker build -t myproject .

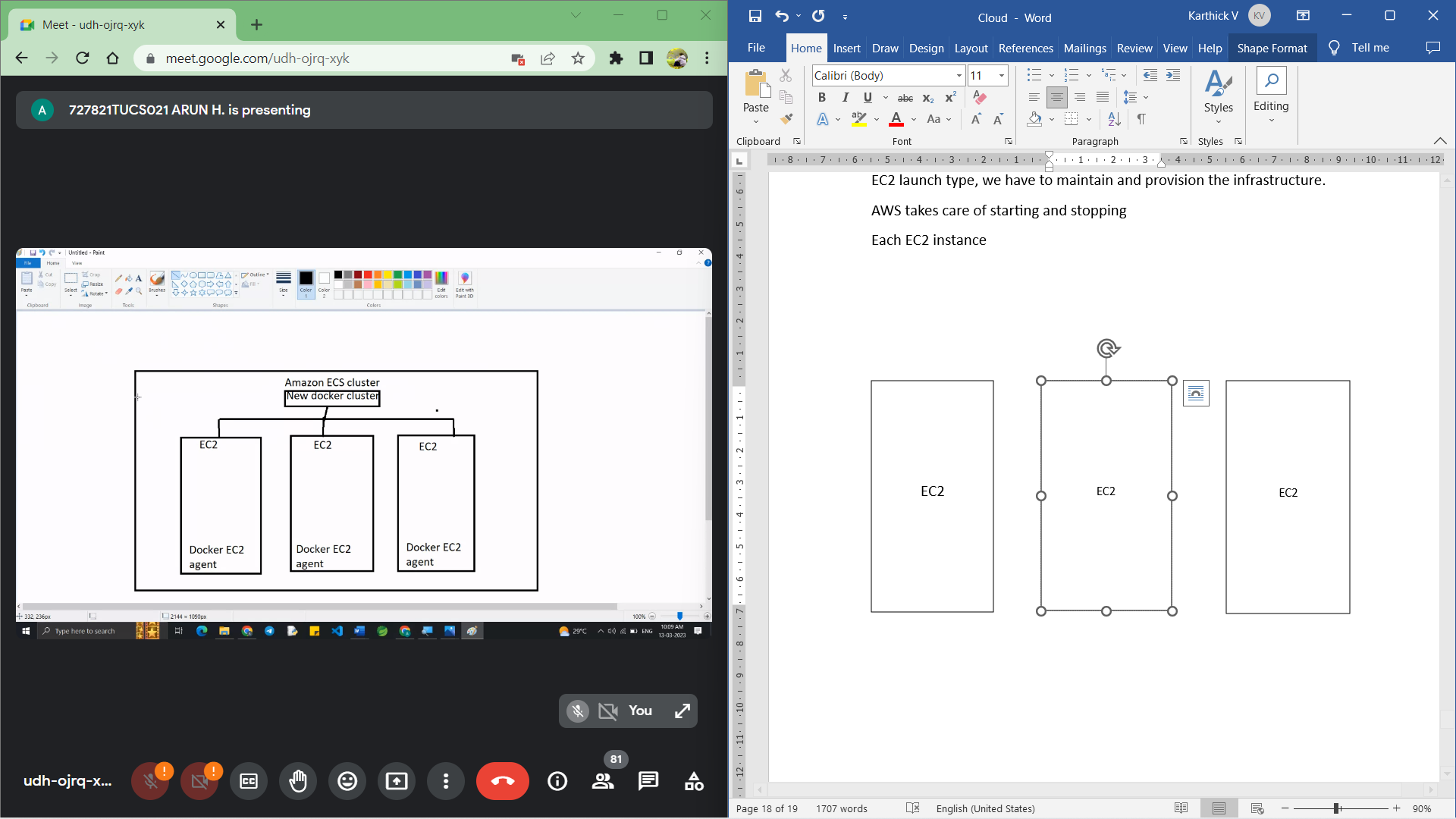
sudo docker run -p 80:80 myproject

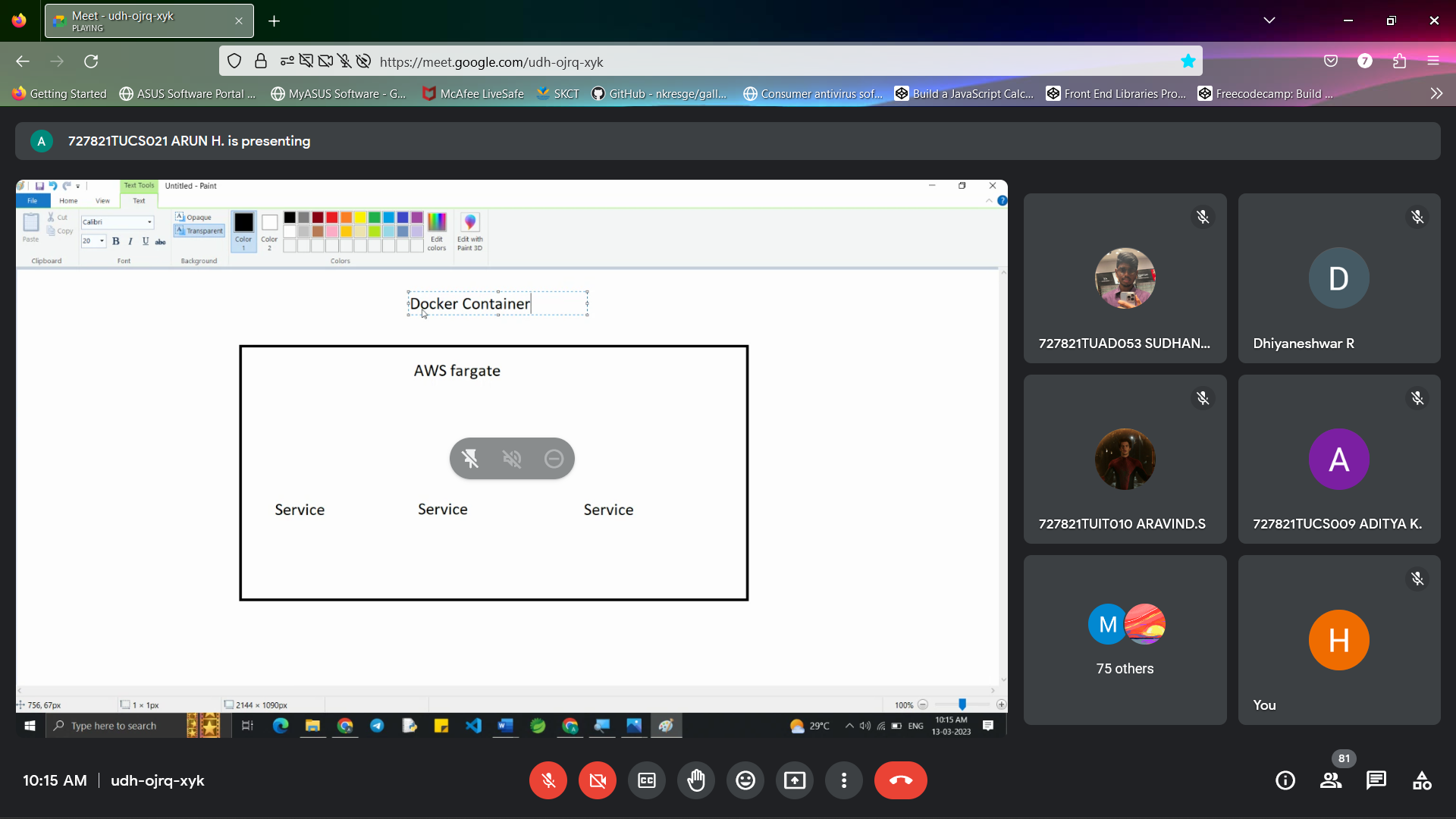
[ sudo lsof -i :80

sudo systemctl stop nginx ] if only means of error

**Amazon ECS:**

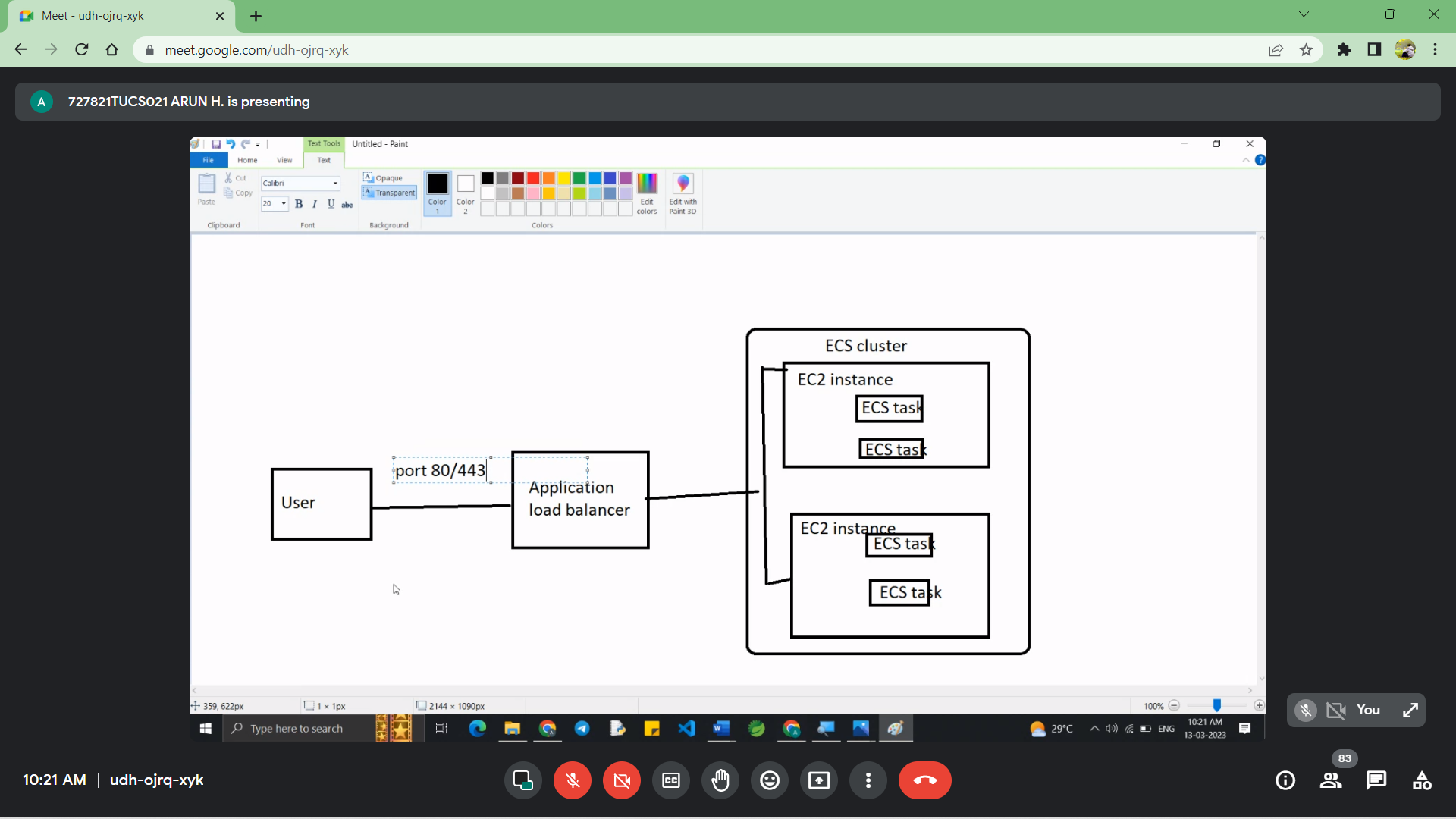
1. ECS🡺 Elastic Container Service
2. We launch a docker by launching ECS task on ECS clusters.
3. EC2 launch type, we have to maintain and provision the infrastructure.
4. AWS takes care of starting and stopping containers.
5. Each EC2 instance must run the ECS agent to register in the ECS cluster.



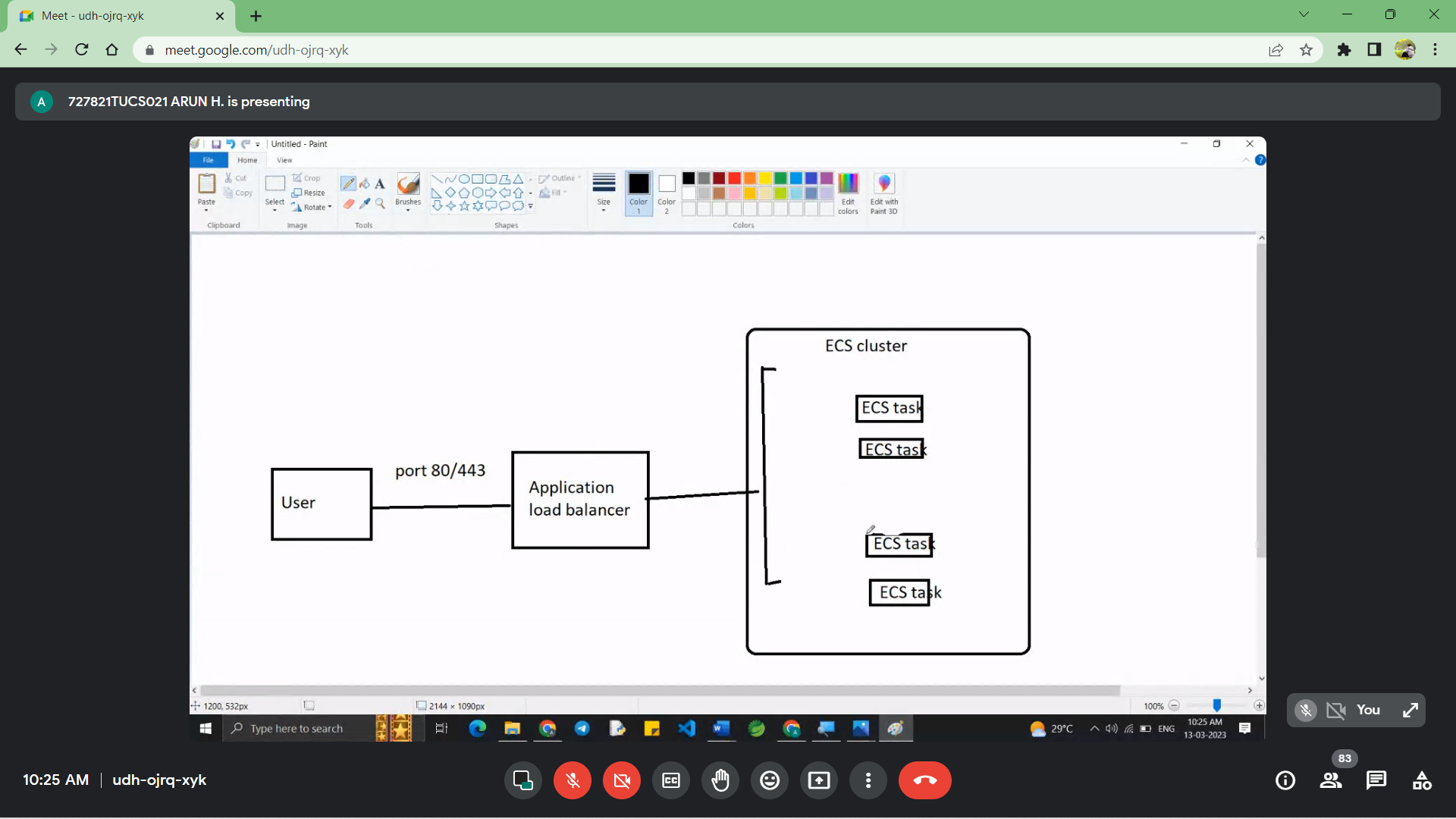


AWS far gate:

1. You do not provision the infrastructure (no EC2 instance to manage)
2. It is a serverless
3. You just create a task definition
4. AWS just runs ECS tasks for you based on the CPU and your RAM need.
5. To scale just increase the number of tasks (no more EC2 instance)



ECS with load balancer



ECS with far gate

EFS 🡺 Elastic File Storage

**DEVOPS:**

1. Dev – software development

Ops – IT operation

1. It uses SDLC model and provides the continues delivery of high-quality software.
2. The main goal of devOps is to improve the quality of the final product by reducing its time to develop, test and deploy new feature and update
3. It achieves through various tools
4. CI/CD – continuous integration and continuous delivery
5. Configuration management monitoring and logging tools

**ECR:**

1. Elastic Container Registry
2. With ECR you can create repositories to store docker image and manage access permissions to this repository.
3. ECR is tightly integrated with ECS and Kubernetes which make easy to deploy and manage docker containers on AWS.
   1. Fully Managed Service
   2. Security and access control
   3. High availability and Scalability
   4. Integration with other AWS services

**Amazon EKS:**

1. Amazon Elastic Kubernetes Service
2. Kubernetes is an open-source system for automatic deployment scaling and management of docker containers
3. It is an alternative to ECS
4. So EKS supports both EC2 instance and far gate

**EKS node types:**

1. managed node groups - create and manages nodes (EC2 instance)
2. Self-managed nodes - node created by us and register to EKS registry (managed by ASG)
3. AWS far gate - no maintenance required and no nodes managed

**EKS Storage Types:**

1. Amazon EBS
2. Amazon EFS

**CI/CD introduction:**

Steps:

1. AWS code command
2. AWS code pipeline
3. AWS code build
4. code deploys
5. code star
6. code artifact
7. code guru

**Steps involved in CI/CD pipeline:**

1. Planning
2. version control
3. build
4. test
5. deploy
6. monitor
7. feedback loop

**Stages of pipeline**

Planning - the tools and technology to be used

Version control - code base

Build

1. compiling the code
2. running unit test case
3. generating artifacts

Test

1. Integration test
2. Performance test
3. Security test

Deploy - deploying the application to the target environment

Monitor – monitoring the application for issue, performance issue, security, Vern ability and issues and errors

Feedback loops - gathering feedback from users, stakeholders and other systems

**CI – Continuous Integration:**

1. A Developer push the code to a code repository often like GitHub code commit with bucket.
2. A testing or build server as soon as the repository is pushed.

E.g., Jenkins code build

1. The developer gets the feedback about the test that were passed are failed.

Advantages:

1. Find bugs early and fix bugs.
2. Delivery faster as the code is tested
3. Deploy faster

**CD – Continuous Deployment:**

1. Ensure that software can be released reliably whenever needed.
2. Ensure deployment happen often and of quick

E.g., Instead of one release every 3-month, 5 release per day happens.

Tools = Code deploy, Jenkin cd, spinnaker