**KUBERNETES**

**----------------------------**

KUBERNETES ---GREEK-----PILOT/NAVIGATOR

Why we called as pilot and Navigator we will see

APPLICATION ARCHITECHTURE:(FACEBOOK/HDFC)

There are 3 terms

1 Tier – Webserver –Static content—HTML

2 Tier –Webserver(HTML)+DATABASE (USER DATA)

3 Tier – Webserver(HTML)+APPSERVER(JAVA)+DATABASE (USER DATA-MYSQL)

**Key differences of static and Dynamic content:**

| **Feature** | **Static Content** | **Dynamic Content** |
| --- | --- | --- |
| **Content Type** | Fixed, unchanging | Customized, changes based on input |
|  |  |  |
| **Server Processing** | No server-side processing required | Server-side processing required |
| **Performance** | Faster | Slower due to computations |
| **Examples** | Images, plain HTML pages | User dashboards, search results |

**Here we take INSTAGRAM the Example for Webserver, App server and Database.**

* **1-Tier --Webserver (Static HTML) -- Instagram only shows a login page (no user data)- same for everyone.**
* **2-Tier--Webserver + Database--After you log in, Instagram shows your profile and posts from the database- (user-specific content).**
* **3-Tier: When you like a post, comment, or share something, the application server processes these actions. The database stores your actions and updates.**

**1-Tier Architecture**

* **What it does**: The **web server** just shows **static pages** (unchanging).

**Instagram Example**:

* + When you open Instagram, you only see the **login page**.
  + It’s the same for everyone and doesn’t change based on who you are.

**2-Tier Architecture**

* **What it does**: The **web server** shows HTML pages and gets **user data** from a **database**.

**Instagram Example**:

* + When you log in, Instagram shows your profile page.
  + It gets your posts, followers, and photos from the **database**.

**3-Tier Architecture**

* **What it does**: The **web server** shows HTML pages, the **app server** processes the actions, and the **database** stores the data.

**Instagram Example**:

* + When you like a post or send a message, the **app server** processes what you did.
  + Your profile, posts, and followers are still stored in the **database**.

**Application code:**

1. Monolithic Architecture
2. Micro service Architecture
3. Server less Architecture
4. **Monolithic:**

**Monolithic was run by single code.**

-Single piece of code

Eg: We Take Amazon (E-commerce site)

If I want to buy one shoe, steps are below

Step 1: **Login** to the amazon e-commerce website.

Step 2: Select the **product** and Select the size & color – (Product is shoe).

Step 3: After that we added the shoe in **cart**. **Added in cart.**

Step 4: And then we need to select the **shipping address**.

Step 5: After that we will do a **payment** using **COD/UPI/CARD**.

Step 6: Some may use logout or someone not else.

**Monolithic Code:** I started writing code in **2015**, and it was **100 lines** and which consist of all the above s 1-6 steps. The below step wise we are started creating code.

**Monolithic code: V1- 100mb**

1.Login

2.Product

3.Cart

4.Shipping address

5.Payment

6.Logout.

In 2017, we made some changes and added extra features to the code.

What are the features?

In 2017, the previously developed code was 100 lines. After that, we developed the following additional features:

* OTP (Features): 1**50 lines**
* Credit Card payment method: **150 lines**
* Prime (One-day delivery): **100 lines**

**Totally – 500 lines and MB also increased into 560 MB**

Notes: Features are added in Red color font.

**Product is shoe:**

Step 1: **Login** to the amazon e-commerce website. ---OTP

Step 2: Select the **product** and Select the size & color –

Step 3: After that we added the shoe in **cart**. **Added in cart.**

Step 4: And then we need to select the **shipping address**. ---Prime (one-day delivery)

Step 5: After that we will do a **payment** using **COD/UPI/CARD**.—CREDIT

Step 6: Some may use logout or someone not else.

**Monolithic code: V2- 500mb**

1.Login (OTP Features added)

2.Product (One-day delivery features added)

3.Cart

4.Shipping address

5.Payment (Credit Card Features added)

6.Logout.

* After adding the new version, we need to update it on the system or mobile. Once updated it will restart automatically on the mobile or computer.

Unnecessary downtime occurs for the Cart, Shipping, and Logout features. During this time, customers might switch to Flipkart to buy the item. In a monolithic service architecture, the entire application will restart, which can result in losing customers.

Notes: In such cases, customers might have trouble logging in during an update, causing them to switch to other shopping platforms. In a monolithic architecture, we may face regular downtime every time a new update is released.

**2. Micro services**

-Loosely coupled code.

-No Downtime.

-Containerization.

-600 LINES -6 SEGEMENTS-100 LINES.

I have used the same scenario here as an example.

If I want to buy a pair of shoes, the steps are as follows:

* For our understanding, each service will be assigned to a different person.
* In a monolithic architecture, one developer will write 500 lines of code. But in a micro service architecture, I will assign different services to 5 developers.

Micro service architect:

* **Login and Product** – One developer **(This developer will maintain 2 services)**
* **Cart** - One developer.
* **Shipping** –One developer.
* **Payment**-One developer.
* **Logout**-One developer.

**For this type of splitting into developers will make downtime may less compare to pervious monolithic architect.**

**Micro services**

Login

Product

Cart

Shipping

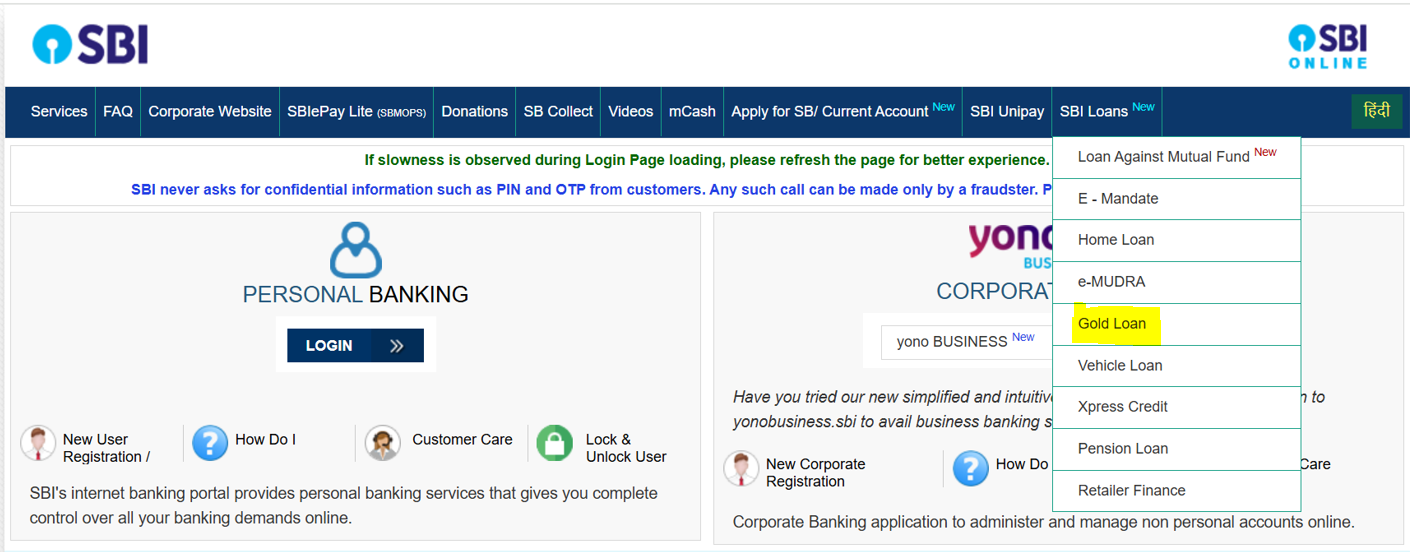
Payment

Logout

Here we have containerized the services. If I want to add a feature to the product service, we only need to update its code. This will cause downtime **only for the product service**, not the other 5 services. The product service also recovers in 1 minute. After that new features will be added in product services. That’s why I Highlighted the product service in yellow in color.

=====================================================================================

**Example we take SBI banking website:**



* The SBI banking website provides many services. I want to make a change in the SBI Loans section where they offer gold loans.
* The interest rate for gold loans needs to be changed from 6.5% to 7%. A developer will update the code to make this change.
* Then, as a DevOps engineer, we will deploy the updated code.

**Server less Architecture:**

-Managed by a cloud provider.

### Example:

If you upload an image to an S3 bucket in AWS, it can trigger an AWS Lambda function to process the image, like resizing it or adding a watermark. You don’t have to manage the server; AWS handles it all.

**How monolithic code runs.**

In a monolithic application running on multiple AWS instances, a **Load Balancer** helps manage incoming traffic. It makes sure the application works well and is always available by distributing the requests to different servers.

**How It Works:**

1. The **Load Balancer** receives requests from users.
2. It decides which **AWS instance** (server) should handle the request.
3. It sends the request to the chosen instance. If one instance is busy or down, it will send the request to another one.

This way, the application can handle more users and keep working even if one server stops.

How micro services codes runs.

Same example we take Amazon E-Commerce:

Example: Great Indian sale running in amazon website.

-Login----100 lakhs members

-Select Product----80 lakhs

-Add to cart-----40 lakhs

-Payment----card-10 lakhs

-Shipping-Fast delivery-5lakhs

-Logout-1 lakhs

Below is the container we split services using containers:

Login

Product

Log out

Login

Product

Login

**Docker: is containerization tool**

**Why we use Docker?**

There are many other tools like Docker, but Docker container tools are the best, and most organizations are using Docker and its free open-source platform.

Container 1

HTTPD

Container 2

SPLUNK

Container 3

Tomcat

Docker

EC2 INSTANCE

Amazon Linux

Docker install

OS-KERNEL

Container can communicate within a server but cannot communicate with one server to another server container.

Kubernetes: K8S

-ORCHESTRATION TOOL –MANAGEMENT & MONIORING.

-Launched by GOOGLE on 2014.

-CNF [Cloud Native Computing Foundation]

-KUBERNETES ARCHITECTURE – MASTER NODE & WORKER NODE.

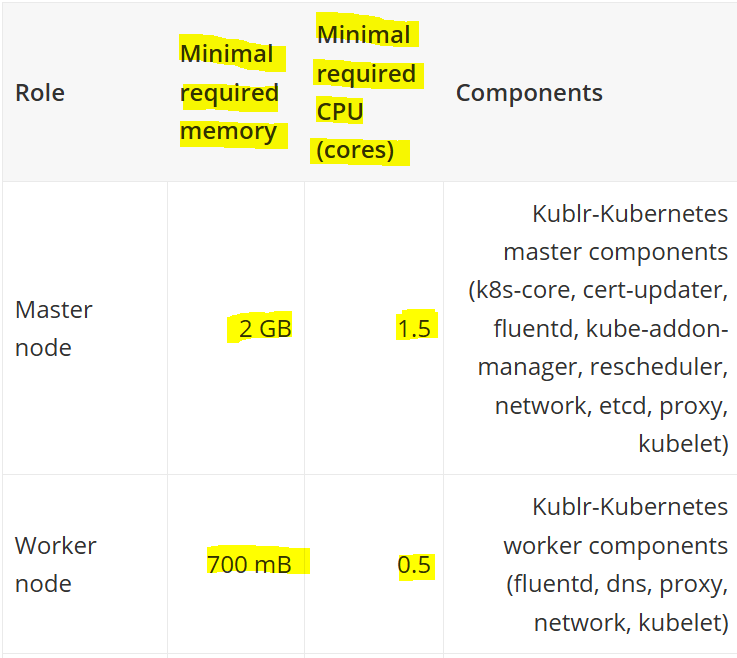
**Master Nodes----Components---T3 Medium (No T2 Medium)**

-4 Components- Controller Manager –API SERVER –SCHEDULER – ETCD

**Worker Node----Components**

**-KUBELET – KUBE PROXY – POD – CONTAINERS**

**Below is minimum requirement for Master node and Worker node**



* **Worker node is EC2 instance.**
* **Master node + worker node is together called cluster.**

**General meaning of cluster**

A **cluster** refers to a group of things or individuals that are closely gathered together, often sharing common characteristics.

In one cluster I can create master node and create multiple worker node in Ec2 instance.

MASTER NODE

Kubernetes Cluster

Worker node

Worker node

Worker node

* Inside worker node
* Please note: Worker node is instance
* POD –small unit of deployment

Worker node (Instance)

POD

Container

Day 2:

CLUSTER: MASTER AND WORKER NODE

-**EC2/STORAGE/VPC/LOAD BALANCER/AUTOSCALING**

**Kubernetes Installation Method:**

1.Single node Installation: Testing & Development

-MINIKUBE-using MINIKUBE we can master node and worker node

-MICROK8S

2.Manual cluster Installation:

-KUBEADM –KUBE ADMIN

3. AUTOMATIC CLUSTER INSTALATION:

-KUBESPRAY----ANSIBLE PLAYBOOK

-KOPS---TERRAFORM-CLOUD PROVIDER---INFRA (We are using this method)

4.MANAGED CLUSTER:(Now most of the organization using this method)

**CLOUD PROVIDERS:**

-**AWS----EKS-ELASTIC KUBERNETES SERVICE**

-AZURE---**AKS**-AZURE KUBERNETES SERVICE

-GCP-**GKE**-GOOGLE KUBERNETES ENGINE

**5.KUBERNETES HARDEST WAY**

-FROM THE SCRATCH We need to install it----START AND TILL END

**In an interview: Interviewer ask which method you are using a Kubernetes previous year I have learned in this method KOPS.**

KOPS –WE NEED TO INSTALL

KUBECTL-WE NEED TO INSTALL

**kubectl** is the essential tool for interacting with Kubernetes clusters. It helps with:

* Managing cluster resources.
* Deploying and scaling applications.
* Monitoring and troubleshooting the cluster.

Main thing: Kops only supports in AWS AND GCP:

CREATE A CLUSTER:

Master Node---Control Plane

Worker Node — Node

1.Linux Machine/Kops Binaries/Kubectl Binaries

2.KOPS BINARIES/KUBECTL BINARIES