**What is Jenkins**

* Jenkins automates the **build** and **deployment** process.
* It is developed using the Java programming language.
* Build means taking the code, compiling it, packaging it, and creating docker image.
* Deployment means creating the container is called deployment of the application.
* Using the Jenkins we can implement the **CI-CD.**
* Jenkins can create and run automation pipelines.
* It integrates with tools like Git, Maven, Gradle, and Docker.

Code will be available. It is a source code repository server. All the developers code will be integrated in the GitHub.

**What is the role of Maven?**

It is a build tool. Take the latest code and compile that code, execute the test cases and package that code.

**What is the role of Docker?**

Create a Docker image, create a Docker container.

**who will do these activities?**

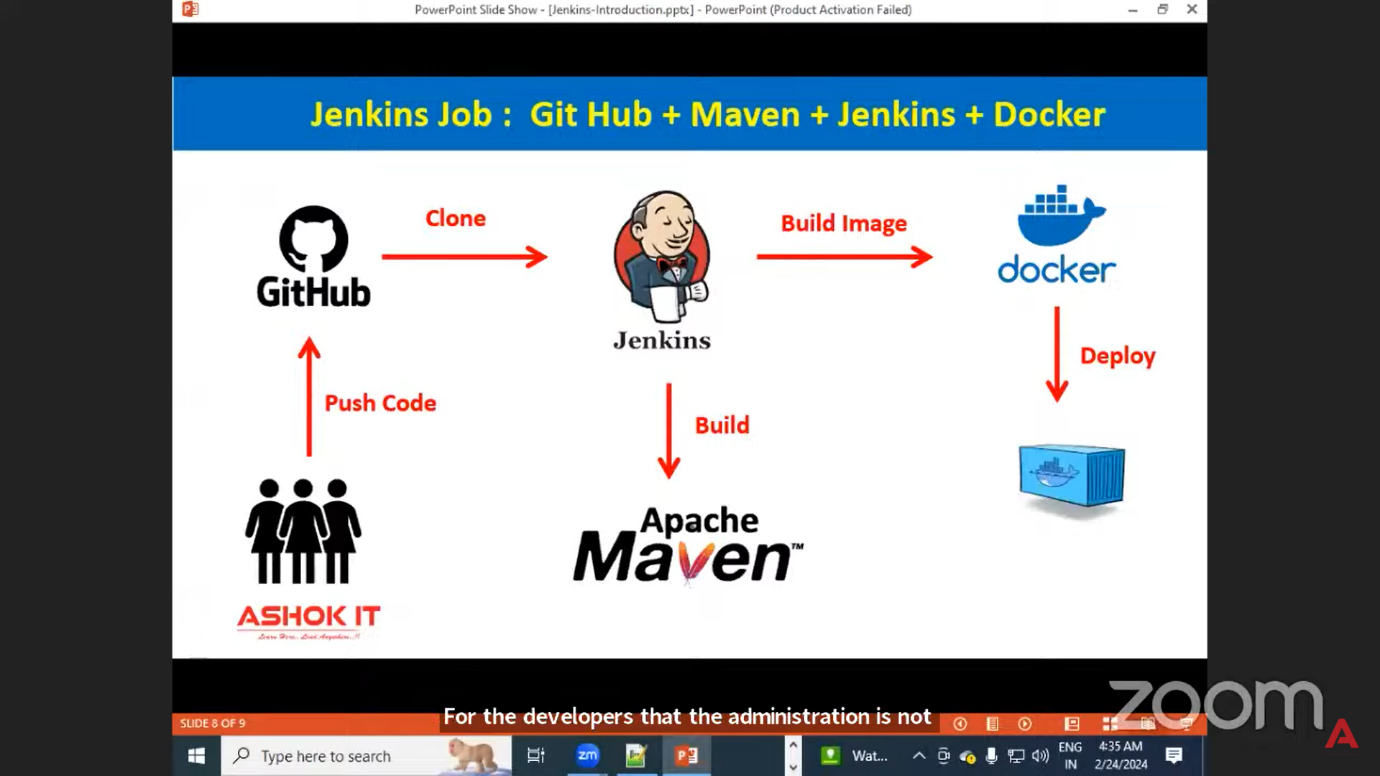
Earlier people used to do manually, but today Jenkins will do these activities.

**What is the role of Jenkins here?**

CICD server.

**Why we are using this Jenkins here?**

To automate the build and deployment process. Whatever the work that we do manually, now I want to automate that work.



* Developers will write the code. Developers will keep the code into GitHub. So, when the code is pushed to the GitHub, then Jenkins will trigger. Jenkins will download the code from the GitHub. Jenkins will communicate with Maven to package our application, compile our application, and package our application. Once the Maven build is completed...
* Jenkins will communicate with Docker to create a Docker image for the project. Once a Docker image is created for our application, we are going to create a Docker container by using that Docker image, which is called CI-CD pipeline. Now, once this Jenkins job is created, when I run that job, it will automate my build and deployment process.

1. Can you tell me what is the CI-CD pipeline in the project?
2. can you tell me how your CI-CD pipeline is developed?
3. Can you tell me what are the steps available in your project CI-CD pipeline?
4. Do you have any experience in writing the CI-CD pipeline?
5. Do you know how to execute the CI-CD pipeline available in the Jenkins?
6. how to install Jenkins.
7. how to create the pipeline.

**How to Jenkins Backup and Restore?**

Using Thin Backup Plugin you can Backup and Restore your Jenkins job.

* Go to Manage Jenkins → Manage Plugins → Available.
* Search Thin Backup and install it.
* Configure backup location:  
  Manage Jenkins → ThinBackup → Settings.
* Create a backup manually or schedule it.

1. Backup directory [where want store backup file in the path].
2. Backup schedule for full backups [H \*\*\*\*]
3. Maximum schedule [10] than how many backups maintain.
4. When 11th backup created then 1st backup deleted.

click on [Save]

**Note**: if you give \*\*\*\*\*: every minute take the backup  
 if you give H\*\*\*\*: every hour takes the backup

* Go to Manage Jenkins, click on Thin Backup, and click on Restore.
* And select which file you need to backup, and click on that Restore button.
* and then restart Jenkin. Once restarted, your backup file will be displayed.

**1️⃣ Open Jenkins Dashboard**

* Go to your Jenkins URL in your browser (e.g., http://localhost:8080).
* Log in with your Jenkins credentials.

**2️⃣ Create a New Job**

* On the dashboard, click **“New Item”** and enter a **job name** (e.g., Build-Vehicle-Insurance-App).
* Choose a job type:
  + **Freestyle project** – good for simple builds.
  + **Pipeline** – good for scripted CI/CD flows (e.g., using Jenkinsfile).
  + **Multibranch Pipeline** – for Git repositories with multiple branches.
* Click **OK**.

**How to create jobs in Jenkins?**

**1. Open Jenkins** → Go to http://localhost:8080 → Login.

**2. Create Job** → Click **“New Item”** → Enter job name → Select **Pipeline**→ **OK**.

**3. Configure Job**

* **Source Code Management** → Select **Git** → Add repo URL & credentials.
* **Build Triggers** → Select trigger (e.g., “Poll SCM” or “Build periodically”).
* **Write pipeline script like below**



**4. Save & Run → Click Save → Click Build Now → Check Console Output.**

**Note:**  Build file = Check build history.

If you n?” eed Install maven=Manage Jenkins- tools. Click on. MavenInstall and Add maven.

If any tool you have installed then give like below

Tools {

Maven 3.8.3

}

When you have installed docker the restart our Jenkins

**65.0.92.247:8080/restart then login again**

**NOTE:**

* when I have committed new code in the git and I run the Jenkins using build now.But new code is not deployed.
* So before creating the new container. stop and remove the existing container to deploy the new code in the Jenkins.

**How to remove container:**

* Sh ‘docker stop ‘srikanth’
* Sh ‘docker rm ‘srikanth’
* Sh ‘docker run -d -p 9090:8080 --name ‘svsrikanth’

**NOTE:**

* **Use** **Build Periodically [\*\*\*\*\*]**: Every minute Automatically deploy started.
* **Use Poll SCM [\*\*\*\*\*]:** If anyone code committed in the GitHub Automatically deploy started.

**1. CI – Continuous Integration**

* Developers regularly merge (integrate) their code into a shared repository (e.g., GitHub, GitLab).
* Each commit triggers an automated build and test process.
* Goal: detect bugs early and ensure new code works with existing code.

**2. CD – Continuous Delivery / Continuous Deployment**

* Continuous Delivery → After CI, the code is packaged and made ready for release; deployment is still a manual decision.
* Continuous Deployment → After CI, the code is automatically deployed to production with no manual step.

**CI/CD Pipeline Steps (Example)**

* Code → Developer writes code and pushes to Git.
* Build → Compile code and prepare artifacts (e.g., JAR, Docker image).
* Test → Run unit tests, integration tests.
* Deploy → Push to staging or production environment.
* Monitor → Observe logs, metrics, errors after release.

<https://kubernetes.io/docs/concepts/overview/>

# Kubernetes

# What is Kubernetes?

* Kubernetes K8s is a orchestration platform. K8s is used to manage containers. K8s is developed by Google. Donated to CNCF.
* Kubernetes provides a framework for managing the complex tasks of deploying, scaling, and operating applications in containers.

# Why is Kubernetes?

* Multiple microservices will be available for one project. For every microservice, we need to create a docker container.
* one docker container is not sufficient. Multiple docker containers we need to create for one microservice.
* For example, for order microservice, I created one docker container. So, with one docker container, one server will be available to handle the request.
* If more number of requests are coming, then one container is not sufficient to handle the load.
* So, creating multiple containers, increasing the count of the containers, decreasing the count of the containers is called management. That management work will be taken care by Kubernetes.

NOTE: it used to manage containers, like create the containers, delete the containers, scale up the containers, scale down the containers. Complete containers management will be taken care by Kubernetes. Kubernetes is called as orchestration platform. Docker is called as containerization platform.

# Advantages of Kubernetes?

**1. container orchestration:**

Kubernetes is it is providing a platform to manage the content. That is called container orchestration. Kubernetes is used to orchestrate our content.

**2.Scalability in Kubernetes:**Scalability means automatically increasing or decreasing the number of containers based on demand. If requests increase, Kubernetes adds more containers; if requests decrease, it reduces them, ensuring optimal infrastructure usage.

**3.Self-healing in Kubernetes:**  
If any container stops or gets damaged, Kubernetes automatically replaces it with a new container to keep the application running without interruption. This automatic recovery process is called self-healing.

**4.Load Balancing in Kubernetes:**  
When an application runs in multiple containers, Kubernetes **distributes incoming requests evenly** among them using a **round-robin** approach, preventing any single container from being overloaded.

# Kubernetes Architecture

**K8S works based on cluster architecture. Cluster means group of servers.**

**1. K8s control plane will contain below components**

1. **API Server:** API server will receive incoming requests and it will store into etcd.
2. **Scheduler:** Scheduler will check pending tasks in etcd and it will schedule those tasks in Worker Nodes.

**Note:** In K8s, our project will be executed as a pod. Inside pod containers will be created or executed.

1. **Controller Manager:** will monitor all k8s resources and functionality.
2. **ETCD:** It is K8S cluster database.

2. **K8S Worker node Container, will contain Below Components**

1. **POD:** Pod is smallest building block that we run in Kubernetes cluster. pod represents runtime instance of our application.
2. **Containers:** Lightweight environments that package an application with its dependencies to run anywhere.
3. **Docker Engine:** Software that creates, manages, and runs containers.
4. **Kublet:** scheduler, will get available worker nodes information by using kublets. Is also called as Worker node agent.
5. **Kube Proxy:** kube-proxy provides network for cluster communication.

**To communicate with K8S control plane, we have two options,**

1. **UI Dashboard**
2. **Kubectl-CLI.**

# Kubernetes Cluster Setup

**Kubernetes cluster we can setup in multiple ways.**

Self-managed K8S cluster

1. MiniKube (Single Node Cluster): only for Practice purpose
2. Kube-ADM (Multi-Node Cluster): download and install our own.

Managed K8S Cluster

1. AWS EKS,
2. Azure AKS,
3. GCP GKE,
4. IBM IKE

# What is Pod?

* Pod is a smallest building block that we can deploy in Kubernetes cluster.
* Our application will be deployed in Kubernetes cluster as a pod only.
* For one application, we can create multiple pod replicas for high availability.
* For every pod, one IP address will be generated.
* If pod got damaged, crashed, then Kubernetes will replace it (self-healing).
* To create pods, we will use manifest.yml files.
* **Note**, the default pods are accessible only within the cluster. (We cannot access outside).
* To expose pods for outside access, we need to use **Kubernetes service concept**.

# What is Service in K8S?

Kubernetes service is used to expose pods. We have three types of services.

**1. Cluster IP** (To access pods within the cluster).

**2. Node port** (To access pods using node public IP).

**3. Load balancer** (To distribute the traffic to pod replicas).

# Difference between Cluster IP, Node Port, and Load Balancer

|  |  |  |  |
| --- | --- | --- | --- |
| **Service Type** | **Description** | **Accessible From** | **Use Case** |
| **ClusterIP** (default) | Exposes the service **internally** within the Kubernetes cluster only. | Inside the cluster | When only internal communication between microservices is needed. |
| **NodePort** | Exposes the service **externally** on a specific port of each **worker node**. | Outside the cluster using <NodeIP>:<NodePort> | For accessing services from outside without a cloud load balancer. |
| **LoadBalancer** | Creates an **external load balancer** (usually via cloud provider) and routes traffic to the service. | Outside the cluster using a **public IP** | For production environments to handle **high traffic** with automatic load balancing. |

# Kubernetes Manifest YML Syntax

# Represent deployment for the pod

apiVersion: apps/v1

kind: Deployment

metadata:

name: javawebappdeployment

spec:

replicas: 2

strategy:

type: RollingUpdate

selector:

matchLabels:

app: javawebap

template:

metadata:

name: javawebapppod

labels:

app: javawebapp

spec:

containers:

- name: javawebappcontainer

image: svs/javawebapp

ports:

- containerPort: 8080

# Represent Service for the pod

apiVersion: v1

kind: Service

metadata:

name: javawebappsvc

spec:

type: LoadBalancer

selector:

app: javawebapp

ports:

-port: 80

containerPort: 8080

# # Commands

* # check pods running
* $ kubectl get pods
* # check pods running in which worker node
* $ kubectl get pods -o wide
* # check services created
* $ kubectl get svc
* # check deployments created
* $ kubectl get deployment
* # Execute k8s manifest yml
* $ kubectl apply -f <yml>
* #Check all
* $ kubectl get all
* #Delete all
* $ kubectl delete all --all

# Jenkins CI/CD Pipeline (Step-by-Step)

1. **Developers** write code and push it to the **Git repository**.
2. **Jenkins pipeline** is triggered automatically and **fetches the code** from Git.
3. Jenkins **builds and packages** the code (JAR/WAR).
4. Jenkins **creates a Docker image** for the application.
5. The **Docker image** is updated in the **Kubernetes manifest file**.
6. **Kubernetes** uses the Docker image to **deploy the application**.
7. This entire automated process is called a **CI/CD pipeline**.

# Steps to Create an AWS EKS Cluster

1. **Install prerequisites →** AWS CLI, kubectl, and eksctl.
2. **Configure AWS CLI →** aws configure.
3. **Create EKS cluster using eksctl:**

eksctl create cluster --name my-cluster --region ap-south-1 --nodegroup-name my-nodes --node-type t3.medium --nodes 2 –managed

1. **Update kubeconfig** to connect kubectl with the cluster:

aws eks --region ap-south-1 update-kubeconfig --name my-cluster

1. **Verify cluster**: kubectl get svc
2. **Deploy application** (optional):

kubectl create deployment nginx --image=nginx

kubectl expose deployment nginx --type=LoadBalancer --port=80

1. **Check resources**:

kubectl get nodes

kubectl get pods

kubectl get svc