**Introduction to Tekton**

**What is tekton:-**

* Tekton is an open-source framework for creating and running continuous integration and continuous delivery (CI/CD) pipelines. It was created by Google, in collaboration with several other companies, as a way to standardize CI/CD workflows and make them more portable across different environments.
* Tekton provides a set of building blocks for creating pipelines, such as tasks, pipelines, and workspaces, which can be combined to create customized workflows for building, testing, and deploying applications. It is designed to be modular and extensible, allowing developers to integrate Tekton with other tools and platforms.
* One of the key features of Tekton is its ability to run pipelines as **Kubernetes-native resources**, which makes it easy to manage and scale pipelines using Kubernetes orchestration. Tekton is also designed to be cloud-native, which means it can be used in any cloud environment or on-premises.

**Use of Tekton:-**

1. **Building and testing applications:** Tekton provides building blocks for creating tasks that can build and test applications. This can help developers catch errors and bugs early in the development cycle, and ensure that code is properly tested before being deployed.
2. **Continuous Integration:** Tekton can be used to create CI pipelines that automatically build, test, and validate code changes as they are pushed to a code repository. This helps ensure that code changes are validated quickly and efficiently, without requiring manual intervention.
3. **Continuous Delivery:** Tekton can also be used to create CD pipelines that automatically deploy code changes to production environments. This can help streamline the deployment process and reduce errors and downtime.
4. **Kubernetes-native Pipelines:** Tekton is designed to run pipelines as Kubernetes-native resources, which means that it can be easily integrated with Kubernetes-based environments. This makes it easy to manage and scale pipelines using Kubernetes orchestration.
5. **Cloud-native:** Tekton is cloud-native, which means it can be used in any cloud environment or on-premises. This makes it easy to deploy and manage pipelines in a variety of different environments.

**Difference between Jenkins and Tekton:**

**Architecture:** Jenkins is a traditional CI/CD tool that runs on a centralized server, whereas Tekton is a Kubernetes-native tool that runs on a container orchestration platform. Tekton is designed to be highly scalable and cloud-native, while Jenkins is more suited for traditional on-premise environments.

**Extensibility**: Jenkins is highly extensible through the use of plugins, which can be used to integrate with a wide range of other tools and technologies. Tekton also supports extensions through its "Task" API, but it is designed to be more lightweight and modular than Jenkins.

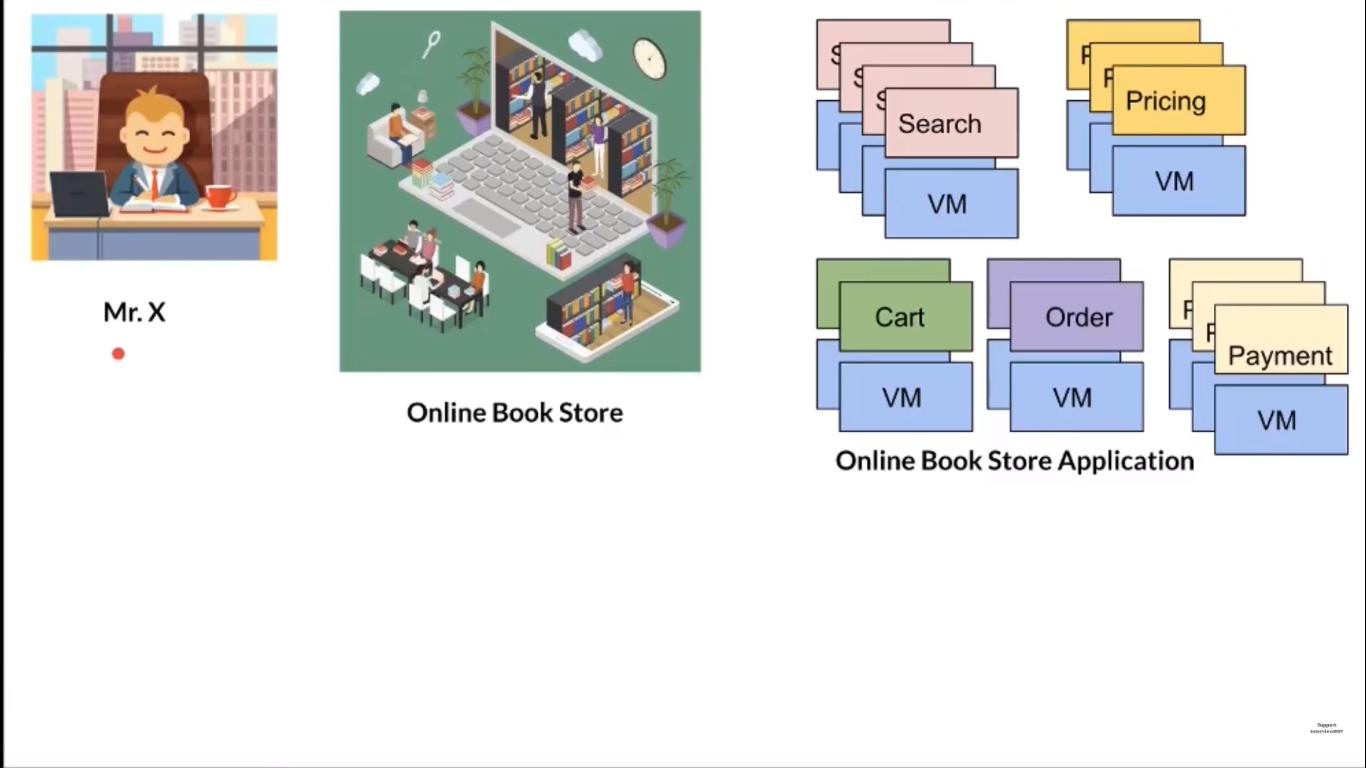
**Pipeline Configuration:** Jenkins uses a graphical user interface (GUI) to configure pipelines, while Tekton uses a YAML-based configuration language that is more code-centric. This can make Tekton more flexible and easier to manage for developers who are familiar with code-based configurations.

**Resource Management:** Tekton provides built-in support for managing Kubernetes resources such as pods, secrets, and ConfigMaps, while Jenkins relies on external plugins to integrate with Kubernetes resources.

**What is Kubernates:**

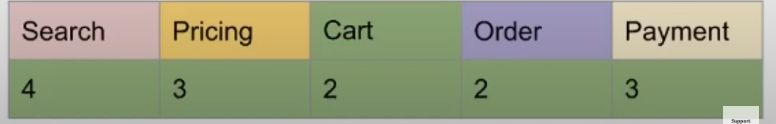
**Problem:** Online bookstore which is low in availability.

**Example:** Online bookstore application which was build using microservices architecture.



**Current scenario:**

* To handle the traffic online bookshop has 14 different microservices



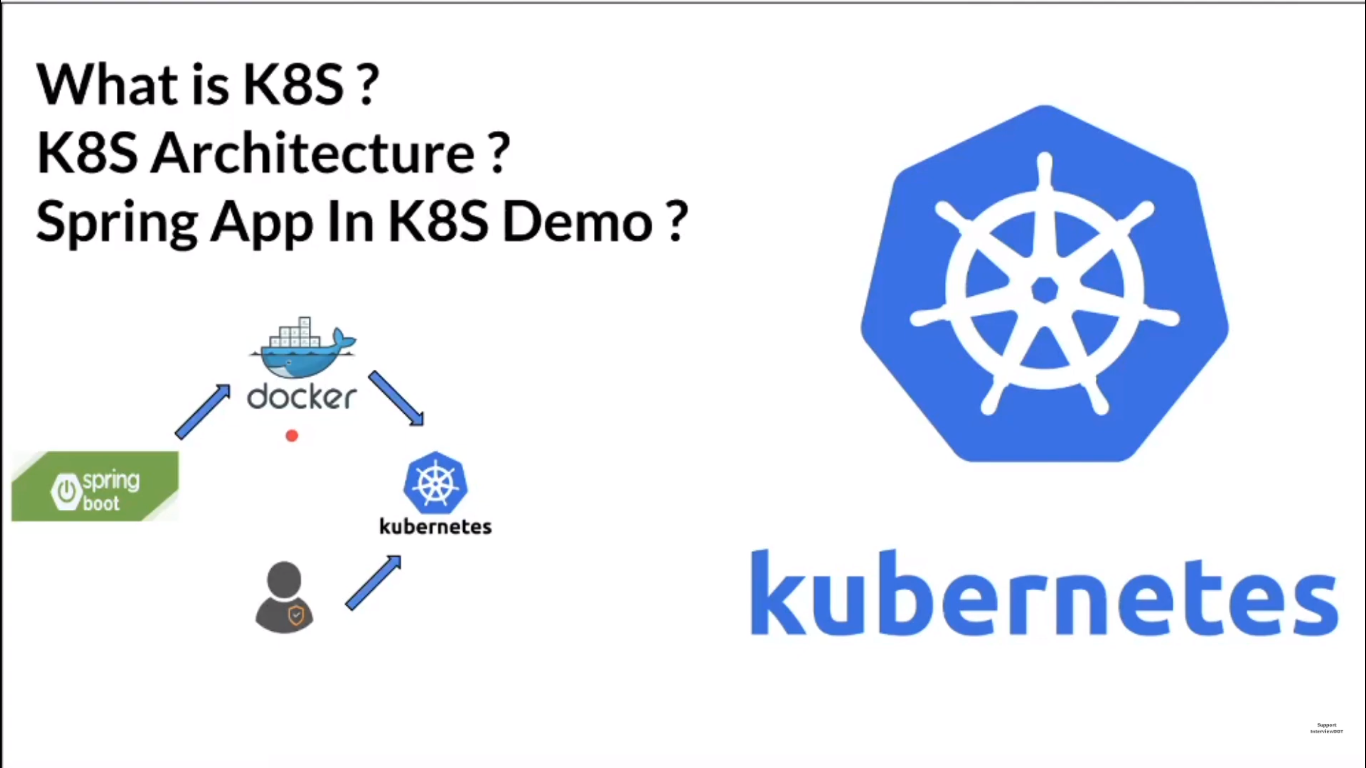
* An application goes down due to many reasons
* No availability for the customers
* Which cannot be monitored 24/7 by an individual

**Goal:** Online bookstore should be highly available.

**What is kubernates and its feature?**

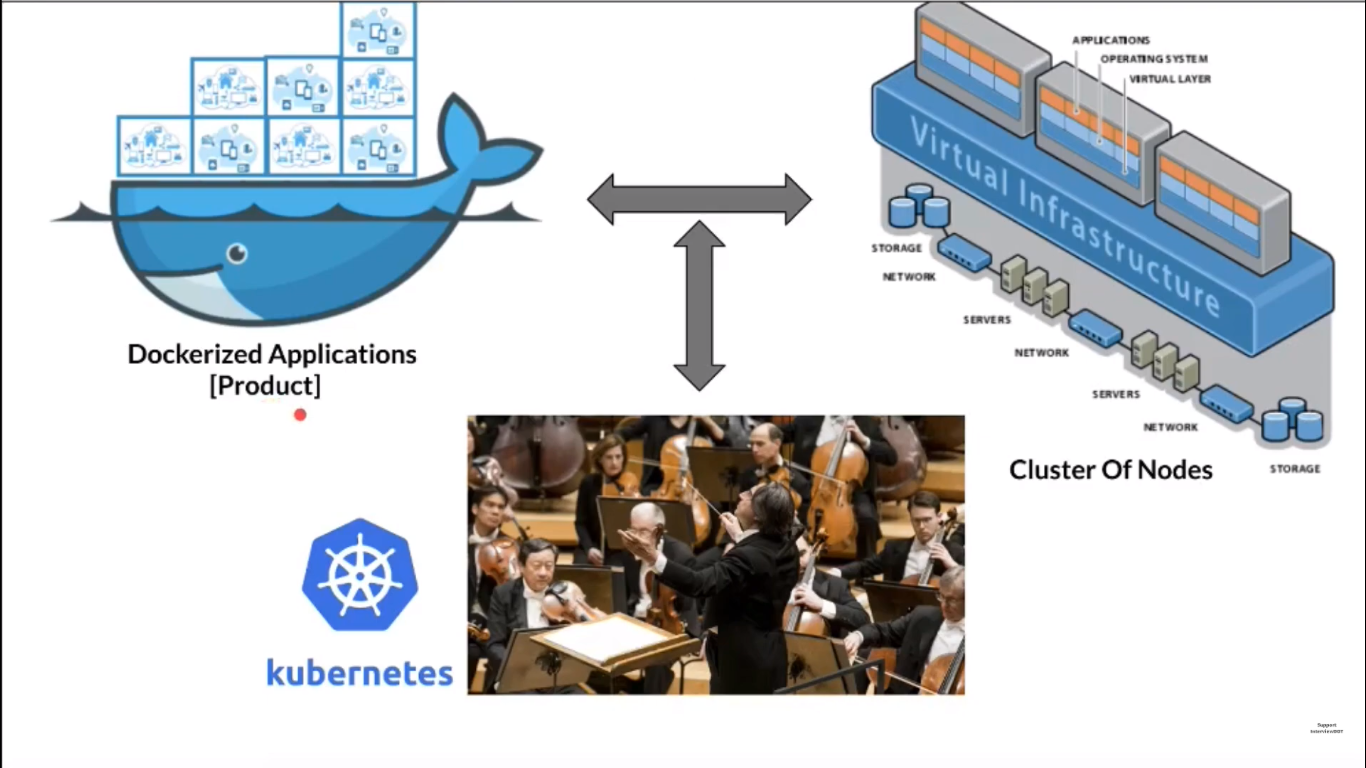
1. Kubernates used to run your dockerized application in cluster of Nodes
2. **Automatic scaling**: Kubernetes can automatically scale the number of containers running based on resource utilization and demand.
3. **Service discovery and load balancing**: Kubernetes has a built-in service discovery mechanism that enables containers to communicate with each other easily. It also provides load balancing for distributing incoming network traffic across the containers.
4. **Self-healing**: Kubernetes can detect and replace failed containers automatically, ensuring that the application is always available.
5. **Rolling updates and rollbacks**: Kubernetes can update an application without downtime by gradually replacing the containers with new ones. If the update fails, Kubernetes can roll back to the previous version.
6. **Resource management**: Kubernetes allows you to specify the resources (CPU and memory) that each container can use, ensuring that containers don't use more resources than necessary.
7. **Multi-cloud support:** Kubernetes can run on various cloud platforms, making it easy to move applications between different environments.
8. **Security**: Kubernetes provides several security features, including network policies, secrets management, and role-based access control (RBAC).

**Spring boot application 🡪 Docker 🡪 Kubernetes**



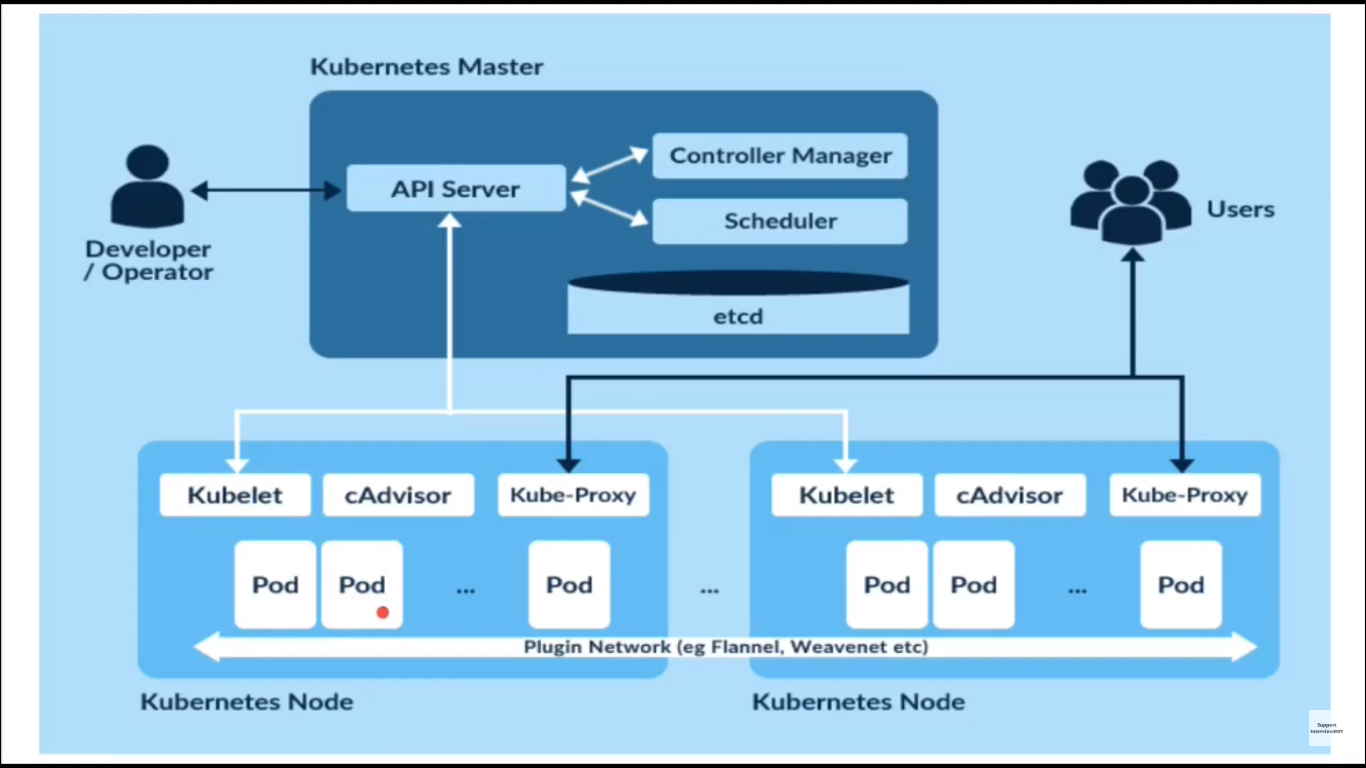
**Informations need to be provided:**

Dockerized application & Cluster of nodes



**Kubernates Architecture:**

1. Which contains Master & worker node
2. **Pod** is a container which is the deployment unit. Our Application will be running in the Pods(Recommended to run one application in one pod)
3. **Kubernetes Master** will be managing the worker node
4. Using **API server** only we can able to communicate with kubernates
5. **Controller manager** used to monitor & manage the status of worker node
6. **Etcd** is used to store deployment details
7. **Kubelet** is used for the communication between the master & worker node
8. **Kube-proxy** is used for the Users to communicate with the application
9. **cAdvisor**  is used to monitor resource usage and analyse the performance of the containers



**Configuration of kubernetes:**

In deployment.yml file we have to configure the instances we need for our application.

