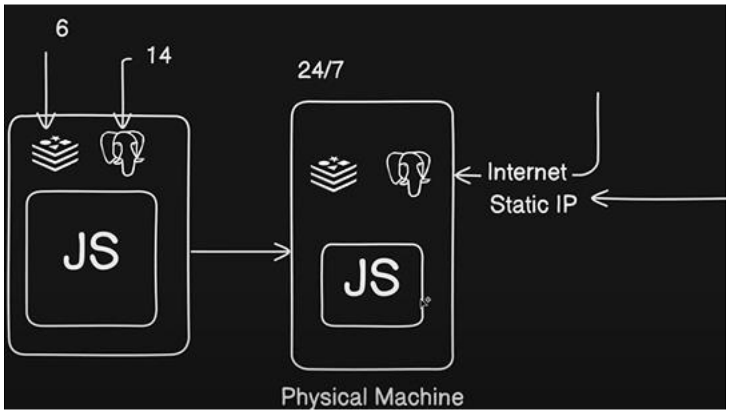
What is Kubernetes

Kubernetes is an open-source, cloud-native platform that allows you to deploy, scale, and manage your applications across multiple nodes and clusters.

Understanding of code deployment on cloud: Suppose we write code in any language (JS/C++/PYTHON) to build the application, it will require the other libraries (Like: Redis, RabbitMQ, SQL etc.) as well. Basically, it will be developer's stack on his machine and same should be replicated on server machine, so that others can also access the application. Server is nothing but it's physical machine where we need to replicate the same environment as developer's stack. Server machine needs public internet access and Static IP. A server needs public internet access to perform its function of storing, sending, and receiving data. If you're running servers, hosting services, or need a stable connection for specific applications, a static IP may be beneficial.

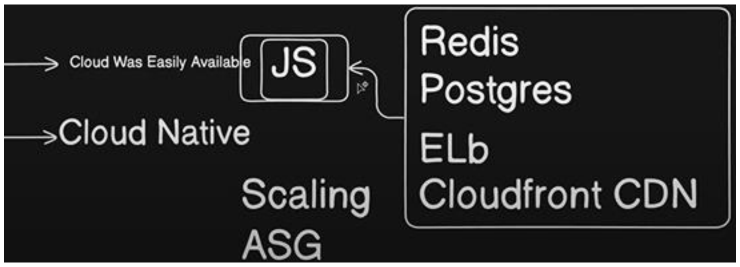
Someone can buy domain, let's say **example.com** and application can be rooted on this domain, so user can access this application using this domain. If more users start using, then we need to do scaling so that all requests to server are processed and service is provided. To scale the server, we might increase number of CPUs and RAM but that's not easy.

Solution: AWS made cloud easily available. Any user can create AWS account, and they can start using servers. AWS provided cloud native environment,



Cloud native environment provides all required services: Redis, Postgres, ELB, CloudFront CDN and just we need to copy or deploy our code to cloud service. These services can be easily configured to my application and connected. Scaling will be easy by just creating ASG(auto scaling group).

What is cloud native:
Cloud native is a software development approach that uses the cloud's distributed computing model to build, deploy, and manage applications.



Problem: If local environment is on windows and cloud environment is on Linux then there would be issue.

Solution:

Virtualization: we can install whatever image would be required or installed on local environment. This virtualization is very high due to complete os installation.

Solution:

Containerization: Very light weight. Containers ensure that code on local will run in similar way using containerization. Containers are:

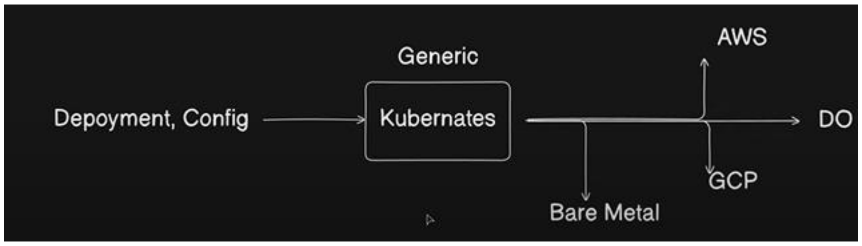
easily scalable, easily sharable and light weight

Containers tasks/operations: Run, monitor, start, destroy, restart, health check

Container orchestration: the process of automating the management of containerized applications.

AWS ECS(Elastic container service): Developed by AWS: If we write project configuration according to AWS ECS then in future, we can not move to other cloud service provider with same configuration. It's not easy to move or change cloud service provider.

Solution: **Kubernetes**: it will allow application to run with any cloud provider service. Kubernetes provides abstraction layer so application will not be depend on particular cloud service provider.



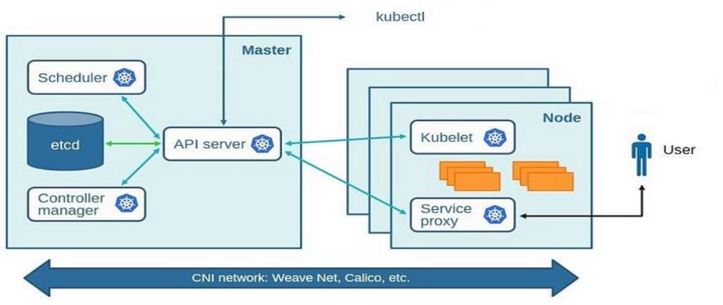
Benefits of using Kubernetes

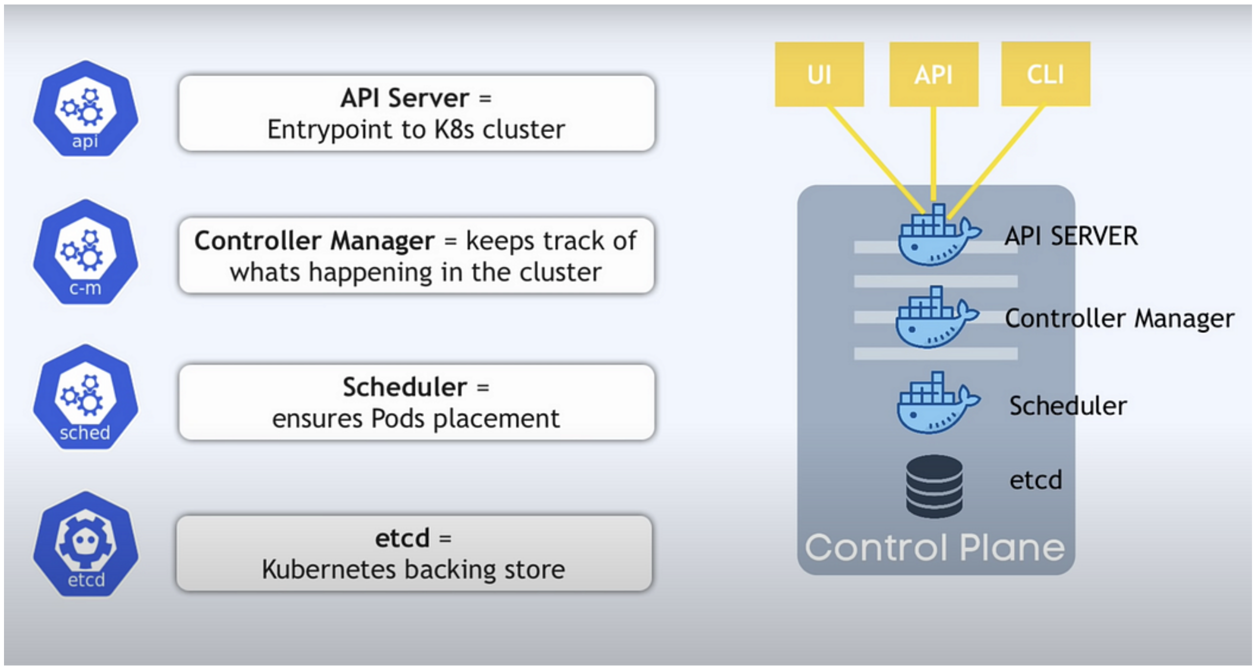
Kubernetes provides many benefits for developers and operators who want to run their applications in containers. Some of the benefits are:

- **High availability**: Kubernetes ensures that your applications are always up and running by automatically restarting failed containers, rescheduling them to different nodes, or scaling them to meet demand.
- **Load balancing**: Kubernetes distributes the traffic among your containers by exposing them as services with their own IP addresses and DNS names. It also supports service discovery and routing mechanisms to connect your services with each other.
- **Scalability**: Kubernetes allows you to easily scale your applications up or down by changing the number of replicas or using horizontal pod autoscaling. You can also scale your cluster by adding or removing nodes as needed.
- **Portability**: Kubernetes enables you to run your applications on any platform that supports Kubernetes, such as public clouds, private clouds, hybrid clouds, or bare metal servers. You can also migrate your applications across different environments without changing your code or configuration.
- **Automation**: Kubernetes automates many tasks that are otherwise tedious or error-prone, such as deployment, rollback, configuration management, secret management, service discovery, resource allocation, health monitoring, etc.
- **Extensibility**: Kubernetes is designed to be modular and pluggable, so you can customize it to suit your needs. You can use various tools and plugins that integrate with Kubernetes, such as Helm, Istio, Prometheus, etc. You can also extend Kubernetes by adding your own custom resources and controllers.

Architecture of Kubernetes

Kubernetes



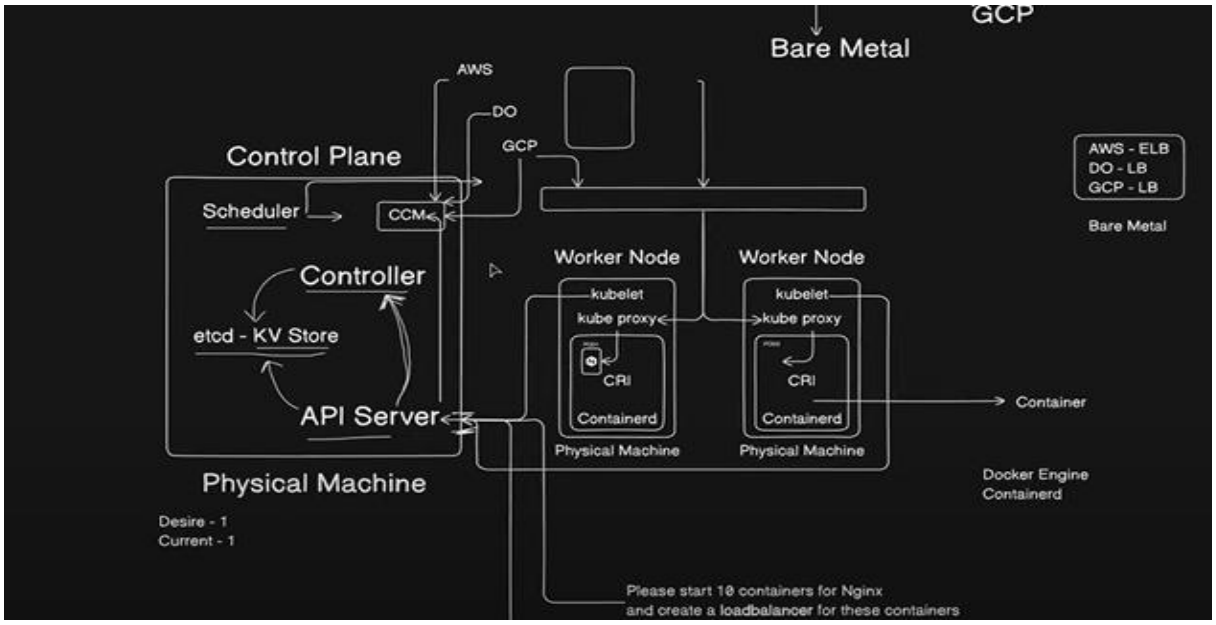


API Server = Entrypoint to K8s cluster

Controller Manager = keeps track of whats happening in the cluster

Scheduler = ensures Pods placement

etcd = Kubernetes backing store



How it works:

First, Kubernetes require physical machine (**control plane**). Control plane has four components inside it:

1) API Server
2) Controller
3) Etcd kv store
4) Scheduler

1) All instructions and configurations are given to API server. API server talks to controller, basically API server exposes the API to developer/user. API server and controllers are connected to etcd data base for reading and writing data.
2) Worker nodes are also important, and these are the physical machines. Actual applications run on worker nodes. Worker nodes contain some components inside it:
    a) Kubelet
    b) Kubeproxy: It handles networking when users send request to server.
    c) CRT(container run time): Container needs run time.
3) Actual container runs inside CRI, worker nodes(through kubelet) are connected to API server.

Actual working: First configuration (start 2 containers of EngineX) is ready by API server and before that user authentication is done by API server. API server will sends information to controller that create 2 container applications. Controller will create 2 pods and engineX containers are running inside it. Pod requires physical machine to run, scheduler will communicate with cubelet service to run pods to worked nodes.

For example, if we have to delete containers then API server will ask cubelet to delete containers from worker nodes.

Kube proxy redirects the traffic to specific pods. Kube proxy does networking work.

Kubernetes maintains 2 states:

Current: 2

Desired: 5

Now 3 more pods will be created. These states (current & desired are stored in etcd database)

CCM(cloud control manager): It has the configuration of particular cloud service provider. It is dynamic component.

----

1. High-Level View

Kubernetes architecture is divided into two main layers:

- **Control Plane**: Responsible for managing the Kubernetes cluster.
- **Nodes (Worker Nodes)**: Responsible for running application workloads in the form of containers.

---

2. Key Components of Kubernetes

Control Plane

The Control Plane is responsible for managing the overall cluster state. Its components include:

1. **API Server (kube-apiserver)**
    - The entry point for all administrative tasks.
    - Exposes the Kubernetes API over RESTful endpoints.
    - Validates and processes API requests and updates the cluster state in the etcd store.
2. **etcd**
    - A distributed key-value store that stores all cluster data.
    - Contains the state of the entire Kubernetes cluster, such as configuration data, secrets, and resource states.
    - Highly consistent and resilient to ensure reliability.
3. **Controller Manager (kube-controller-manager)**
    - Manages various controllers that regulate cluster state:
        - **Node Controller**: Monitors the status of worker nodes.

        - **Replication Controller**: Ensures the desired number of pod replicas are running.
        - **Endpoint Controller**: Updates Service and Pod endpoints.
        - **Others**: Includes Job, Namespace, and Garbage Collection controllers.

4. **Scheduler (kube-scheduler)**
    - Assigns newly created pods to suitable nodes based on resource requirements, policies, and constraints.
    - Takes into account CPU, memory, affinity rules, and taints/tolerations.
5. **Cloud Controller Manager**
    - Interacts with cloud providers to manage external resources.
    - Handles load balancers, node management, and storage integration in cloud environments.

Nodes (Worker Nodes)

Worker Nodes are responsible for running application workloads. Each node contains the following components:

1. **Kubelet**
    - An agent that communicates with the Control Plane.
    - Ensures containers in pods are running and reports node status.
2. **Kube Proxy**
    - Manages networking on the node.
    - Implements network rules to allow communication between pods and services using IP tables or IPVS.
3. **Container Runtime**
    - Executes containers in pods.
    - Supported runtimes include Docker, containerd, CRI-O, and others compatible with Kubernetes' Container Runtime Interface (CRI).
4. **Node OS**
    - The operating system where the container runtime and other node components run (e.g., Linux or Windows).

---

3. Objects and Resources in Kubernetes

Kubernetes operates through declarative objects, including:

1. **Pods**
   - The smallest deployable unit in Kubernetes.
   - A pod represents a group of containers that share networking and storage.
2. **Services**
   - Provides stable IP and DNS to expose pods.
   - Types include ClusterIP, NodePort, LoadBalancer, and ExternalName.
3. **ReplicaSets**
   - Ensures a specified number of pod replicas are running.
4. **Deployments**
   - Manages updates and rollbacks for applications.
5. **ConfigMaps and Secrets**
   - Manages configuration data and sensitive information, respectively.
6. **Ingress**
   - Manages HTTP and HTTPS routing to expose services.
7. **Volumes and Persistent Volumes**
   - Handles storage for containers.
8. **Namespaces**
   - Provides logical isolation within the cluster.

---

4. Networking in Kubernetes

Kubernetes networking includes:

- **Pod-to-Pod Communication**: Enabled by CNI (Container Network Interface) plugins like Calico, Flannel, or Weave.
- **Service Discovery**: Handled by kube-proxy and CoreDNS.
- **Ingress Controllers**: Facilitates external access to services.

---

5. Security in Kubernetes

- **Role-Based Access Control (RBAC)**: Manages permissions.
- **Network Policies**: Restricts communication between pods.
- **Pod Security Standards (PSS)**: Ensures pods follow security guidelines.

---

6. Kubernetes Cluster Workflow

1. **Developer Interaction**
    - Developers define applications using YAML/JSON manifests and send them to the API Server.
2. **Control Plane Processes**
    - The API Server validates the requests.
    - The Scheduler assigns the workload to a node.
    - Controllers ensure the desired cluster state.
3. **Node Operations**
    - The Kubelet receives instructions and ensures containers are running.
    - The Kube Proxy facilitates networking.
    - Pods run application containers managed by the container runtime.

---

7. High Availability (HA)

To achieve high availability:

- **Control Plane Redundancy**: Multiple instances of API Servers and etcd.
- **Load Balancers**: Distribute API requests across multiple API Servers.
- **Node Redundancy**: Distribute workloads across multiple nodes.

Basic Commands

1. **View cluster information:** kubectl cluster-info
2. **Check version:** kubectl version
3. **View API resources:** kubectl api-resources
4. **View component status:** kubectl get componentstatuses

Resource Management

1. **List resources:**
    1. kubectl get pods
    2. kubectl get services
    3. kubectl get deployments
2. **View resource details:**

```
kubectl describe pod <pod-name>
```

3. **Apply a configuration:**

```
kubectl apply -f <filename>.yaml
```

4. **Delete resources:**

```
kubectl delete pod <pod-name>
kubectl delete -f <filename>.yaml
```

Pods

1. **Run a pod:**

```
kubectl run <pod-name> --image=<image-name>
```

2. **View pod logs:**

```
kubectl logs <pod-name>
```

3. **Execute commands inside a pod:**

```
kubectl exec -it <pod-name> -- /bin/bash
```

Deployments

1. **Create a deployment:**

```
kubectl create deployment <deployment-name> --image=<image-name>
```

2. **Scale a deployment:**

```
kubectl scale deployment <deployment-name> --replicas=<number>
```

3. **Update a deployment:**

```
kubectl set image deployment/<deployment-name> <container-name>=<new-image>
```

4. **View deployment rollout status:**

```
kubectl rollout status deployment/<deployment-name>
```

5. **Rollback a deployment:**

```
kubectl rollout undo deployment/<deployment-name>
```

Services

1. **Expose a deployment as a service:**

```
kubectl expose deployment <deployment-name> --type=<service-type> --port=<port>
```

2. **List services:**

```
kubectl get services
```

Namespaces

1. **List namespaces:**

```
kubectl get namespaces
```

2. **Switch namespace:**

```
kubectl config set-context --current --namespace=<namespace>
```

3. **Create a namespace:**

```
kubectl create namespace <namespace-name>
```

Configuration and Debugging

1. **View current configuration:**

```
kubectl config view
```

2. **View logs of a node or pod:**

```
kubectl logs <pod-name>
```

3. **Debug a pod interactively:**

```
kubectl debug <pod-name> --image=<debug-image>
```

4. **Explain a resource:**

```
kubectl explain <resource-type>
```

Monitoring and Metrics

1. **Top nodes or pods (requires Metrics Server):**

   ```
   kubectl top nodes
   ```

   ```
   kubectl top pods
   ```

2. **List events:**

   ```
   kubectl get events
   ```

YAML Management

1. **Generate YAML for a resource:**

   ```
   kubectl get pod <pod-name> -o yaml
   ```

2. **Edit a resource:**

   ```
   kubectl edit <resource-type>/<resource-name>
   ```

3. **Replace a resource:**

   ```
   kubectl replace -f <filename>.yaml
   ```