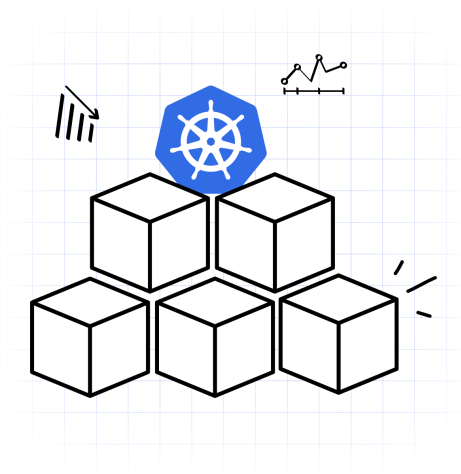
⁉️

Kubernetes Explained - What is Kubernetes & How it works?

Why is Kubernetes so Popular? 🤩



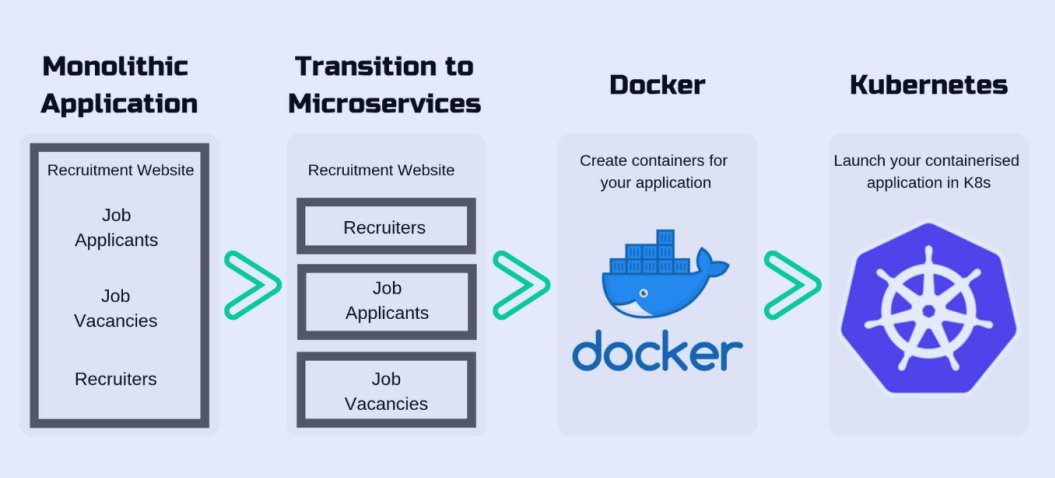
Kubernetes is amazing, it simplifies complex tasks, making it easier for DevOps engineers to manage applications seamlessly. So big companies like NASA, ChatGPT, Microsoft, Google, Spotify, Pinterest almost everyone rely on Kubernetes.

https://www.airplane.dev/blog/companies-using-kubernetes

💡 • Virtually every job description for DevOps or cloud-related roles includes Kubernetes skills.

So this is the right time to learn and master Kubernetes!

What Problem does Kubernetes solve? 🤔



Kubernetes addresses several challenges associated with deploying and managing containerized applications in a distributed environment. Here are some key problems that Kubernetes solves:

1. **Container Orchestration:**
    - **Challenge:** Manual deployment and management of individual containers for services is complex.
    - **Kubernetes:** Automates the deployment, scaling, and management of microservices, ensuring seamless operation.
2. **Declarative Configuration:**
    - **Challenge:** Defining configurations for complex applications can be error-prone.

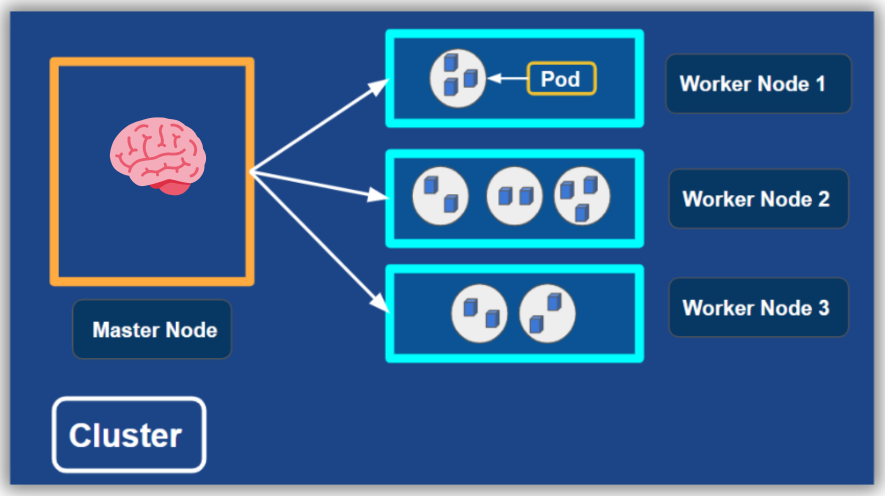
- **Kubernetes:** Uses manifests or declarative configurations, enabling you to specify microservices' desired states. Kubernetes ensures actual states match defined ones, minimizing configuration errors.

3. **Scalability:**
    - **Challenge:** Scaling microservices efficiently becomes difficult as the recruitment platform grows.
    - **Kubernetes:** Automatically scales services based on demand, optimizing performance across cluster nodes.
4. **Fault Tolerance and High Availability:**
    - **Challenge:** Ensuring continuous availability for all the services is challenging.
    - **Kubernetes:** Ensures fault tolerance by automatically rescheduling failed containers, maintaining the desired state, and ensuring high availability.
5. **Service Discovery and Load Balancing:**
    - **Challenge:** Discovering and connecting to dynamic services is challenging.
    - **Kubernetes:** Automates service discovery and load balancing, facilitating seamless communication between service interfaces.
6. **Rolling Updates and Rollbacks:**
    - **Challenge:** Updating or rolling back applications without downtime is risky.
    - **Kubernetes:** Supports seamless updates without disrupting user activities and provides the capability to rollback changes too.

Kubernetes Architecture in Depth 🔎

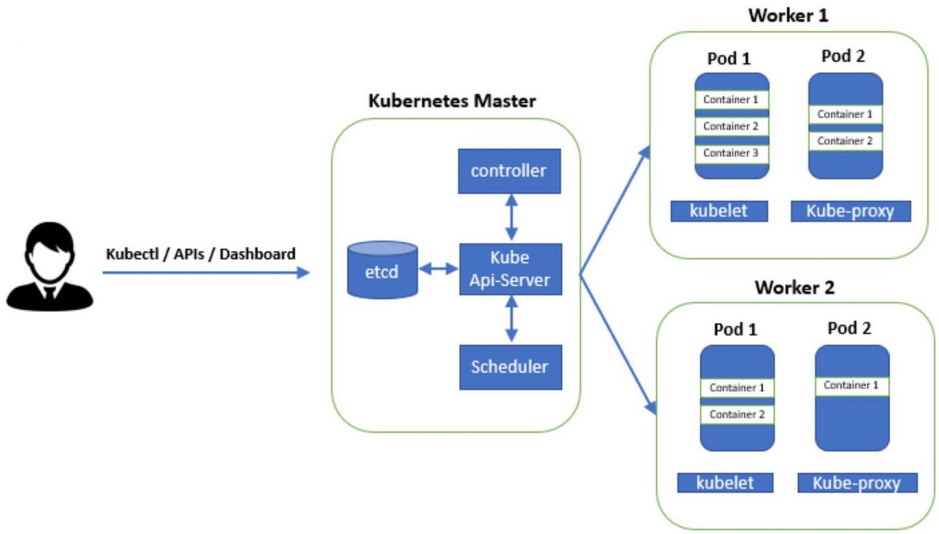
High-Level Overview:

Kubernetes architecture consists of two main components: the **Master Node (Control Plane)** and **Worker Nodes**. Let's understand their roles at a high level.



Feature	Master Node	Worker Node
Responsibility	Manages the entire cluster and is crucial for cluster-wide decisions.	Receives instructions from the control plane and manages local execution of workloads.

- **Master Node:**
  - Centralized control plane, Manages the entire cluster.
  - Key components include the **API Server**, **Controller Manager**, **Scheduler**, and **etcd**.
- **Worker Nodes:**
  - Execute the actual workloads, Host containers that run applications.
  - Components on worker nodes include the **Kubelet, Kube-Proxy** and **Container Runtime** (e.g., Docker).



Master Node Components:

In a Kubernetes cluster, the master node manages and coordinates the cluster's operation. Key components include:

1. **etcd:**

   - **Role:** Etcd is a distributed key-value store that serves as the primary datastore for Kubernetes. It stores the configuration data that represents the state of the entire cluster, including information about nodes, pods, services, and more.
   - **Responsibilities:**
     - Stores and retrieves the configuration data for all cluster components.
     - Consistently replicates data across a cluster of etcd nodes, providing fault tolerance and high availability.

   Documentation: https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/#:~:text=etcd is a consistent and,etcd in the official documentation.

2. **kube-apiserver:**

   - **Role:** The API server is the central management entity for the entire Kubernetes cluster. It exposes the Kubernetes API, which allows users, administrators, and other components to interact with the cluster. All other components communicate with the API server to read or modify the cluster state.
   - **Responsibilities:**
     - Validates and configures data for the API objects.
     - Serves the Kubernetes API, handling requests from the users and components through Kubernetes CLI (`kubectl`) and other API clients.
     - Acts as a front-end for the etcd datastore, persisting cluster state.

   Documentation: kube-apiserver Documentation

3. **kube-scheduler:**

   - **Role:** The scheduler is responsible for placing pods onto nodes based on resource requirements, constraints, and other policies. It **makes decisions** on where to deploy pods in the cluster.
   - **Responsibilities:**

- Watches for newly created pods with no assigned node.
- Selects an appropriate node for each pod based on resource requirements, node capacity, and other constraints.
- Updates the pod's information in the API server, assigning it to a specific node.

Documentation: kube-scheduler Documentation

4. **kube-controller-manager:**

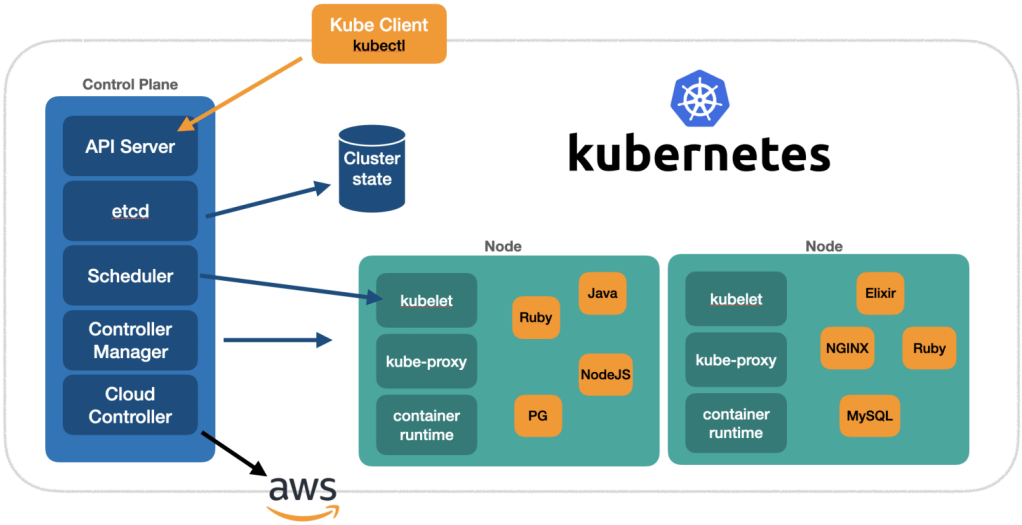
- **Role:** The controller manager runs controller processes that monitor the state of the cluster and respond to changes to drive the cluster towards the desired state.
- **Responsibilities:**
  - Node Controller: Monitors and responds to changes in the state of nodes.
  - Replication Controller: Maintains the desired number of pod replicas.
  - Endpoints Controller: Populates the Endpoints object (combining services and pods).
  - Service Account & Token Controllers: Manages the lifecycle of service accounts and API access tokens.

Documentation: kube-controller-manager Documentation

5. **cloud-controller-manager (Optional):**

- **Role:** In cloud-based Kubernetes installations, the cloud controller manager integrates with cloud-specific APIs to manage external resources such as load balancers and storage volumes.
- **Responsibilities:**
  - Interacts with the cloud provider's API to manage resources.
  - Abstracts cloud-specific functionality from the core Kubernetes components.

```
--controllers stringSlice     Default: [*]
A list of controllers to enable. '*' enables all on-by-default controllers, 'foo' enables the controller
named 'foo', '-foo' disables the controller named 'foo'.
All controllers: attachdetach, bootstrapsigner, clusterrole-aggregation, cronjob, csrapproving,
csrcleaner, csrsigning, daemonset, deployment, disruption, endpoint, garbagecollector,
horizontalpodautoscaling, job, namespace, nodeipam, nodelifecycle, persistentvolume-binder,
persistentvolume-expander, podgc, pv-protection, pvc-protection, replicaset, replicationcontroller,
resourcequota, root-ca-cert-publisher, route, service, serviceaccount, serviceaccount-token, statefulset,
tokencleaner, ttl, ttl-after-finished
Disabled-by-default controllers: bootstrapsigner, tokencleaner
```



Worker Node Components:

1. **kubelet:**
    - **Role:** The kubelet is an agent running on each node, responsible for creating and maintaining the state of pods and ensuring they run as expected.
- **Responsibilities:**
    - Listens for pod-related instructions from the control plane (kube-apiserver).
    - Ensures containers within pods are running and healthy.
    - Reports node status and usage back to the control plane

Documentation: kubelet Documentation

CRI (Container Runtime Interface):

- **Role:** Interface for communication between the kubelet and container runtimes, defining how containers are managed within pods.

CNI (Container Network Interface):

- **Role:** Interface specifying how network plugins interact with container runtimes to provide networking for pods in a Kubernetes cluster.

CSI (Container Storage Interface):

- **Role:** Interface enabling Kubernetes to work with various storage systems and drivers for persistent storage volumes attached to pods.

kube-proxy:

- **Role:** The kube-proxy is responsible for network proxying and load balancing, ensuring communication between services and forwarding external traffic to the correct pods.
- **Responsibilities:**
  - Manages network rules to enable pod-to-pod communication within the cluster.
  - Implements service load balancing, distributing external traffic to service endpoints.
  - Facilitates network policies for secure communication between pods.

  Documentation: kube-proxy Documentation

What Happens when you create a POD?

User Requests Pod Deployment:

1. **Create Pod Configuration:**
   - Create a file named `my-pod.yaml` with the following content:

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
  - name: mycontainer
    image: nginx
```

1. **Submit Pod Configuration to kube-apiserver:**
    - Use `kubectl` to submit the pod configuration to the Kubernetes API server:

```
kubectl apply -f my-pod.yaml
```

1. **kube-apiserver and etcd:**
    - `kube-apiserver` validates the pod configuration and stores it in etcd.
    - View the pod status in etcd:

```
kubectl get pod mypod -o json | jq .status
```

1. **kube-scheduler:**

- Scheduler assigns the Pod to an available Worker Node.
- Check the scheduling decision made by the scheduler:

```
kubectl get pod mypod -o json | jq .spec.nodeName
```

1. **kubelet (on the worker node):**
    - Kubelet on the assigned Node creates and starts the Pod.
    - The `kubelet` on the selected worker node executes the instructions to create the pod:

```
kubectl get nodes # Identify the node where the pod is scheduled
ssh <worker-node>
```

```
docker ps # View running containers on the worker node
```

1. **Container Runtime (CRI - e.g., Docker):**
    - Inspect the container runtime to see the created containers:

```
docker ps
docker inspect <container-id>
```

1. **kube-proxy (on each worker node):**
    - Inspect the network rules set up by `kube-proxy`:

```
kubectl describe pod mypod | grep IP
iptables-save | grep <pod-ip>
```

1. **CNI (Container Network Interface):**
    - Inspect the network configuration set up by the CNI plugin:

```
kubectl describe pod mypod | grep CNI
```
