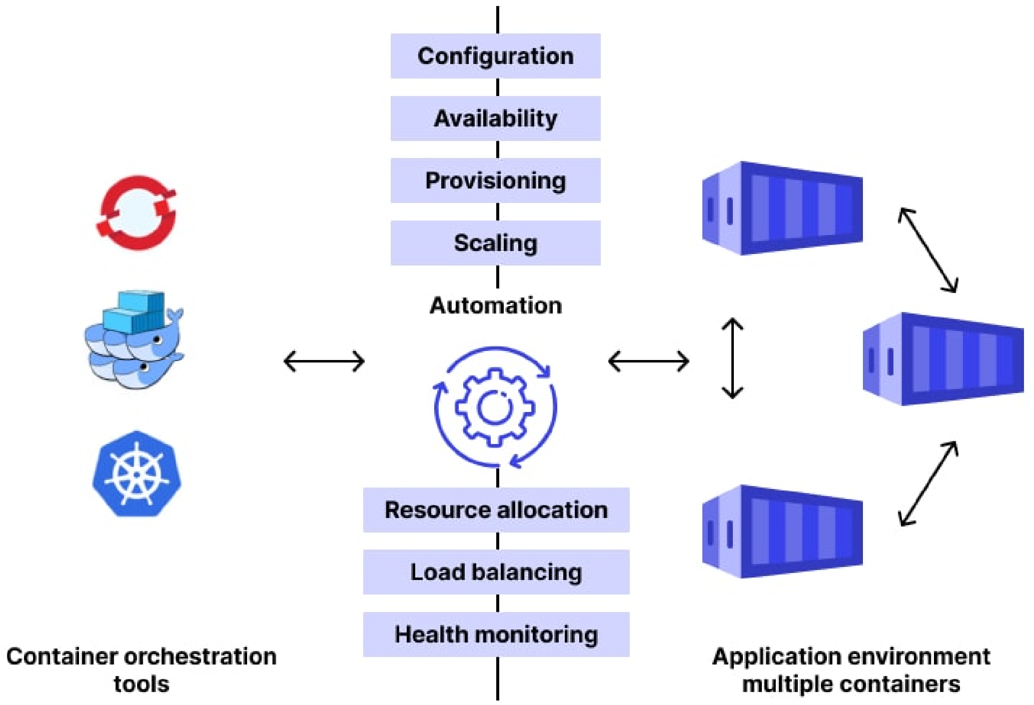
Container Orchestration

Managing Containerised Applications at Scale

What is Container Orchestration?

- Container orchestration automates the management, deployment, scaling, and networking of containers. It's crucial when dealing with a large number of containers running across multiple environments.
- **Why it’s needed?**
    - As the number of containers grows, manually managing them becomes unfeasible. Orchestration tools provide automation for deploying, scaling, and managing these containers in a controlled manner.

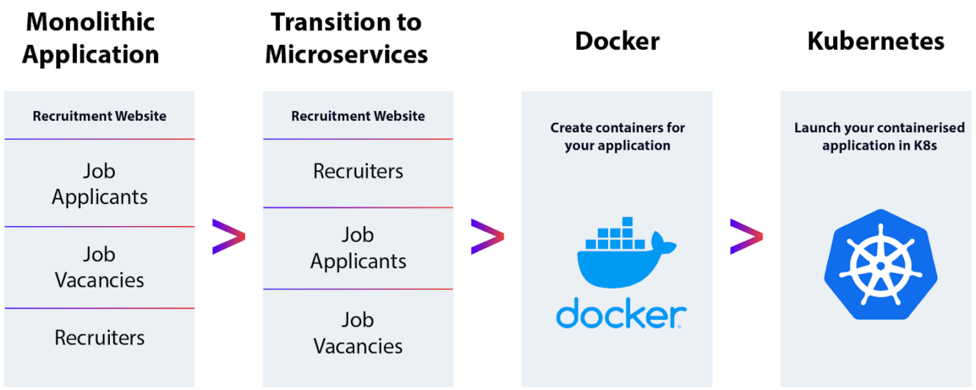
▼ Key Components of Container Orchestration



- **Scheduling**: Automatically assigns containers to host machines based on resource availability.
- **Scaling**: Dynamically adds or removes containers based on demand.
- **Networking**: Manages the communication between containers and ensures they can interact securely.
- **Load Balancing**: Distributes traffic across multiple containers to optimize resource usage.
- **Service Discovery**: Automatically detects and connects services running in different containers.

▼ Popular Container Orchestration Tools

- **Kubernetes**:

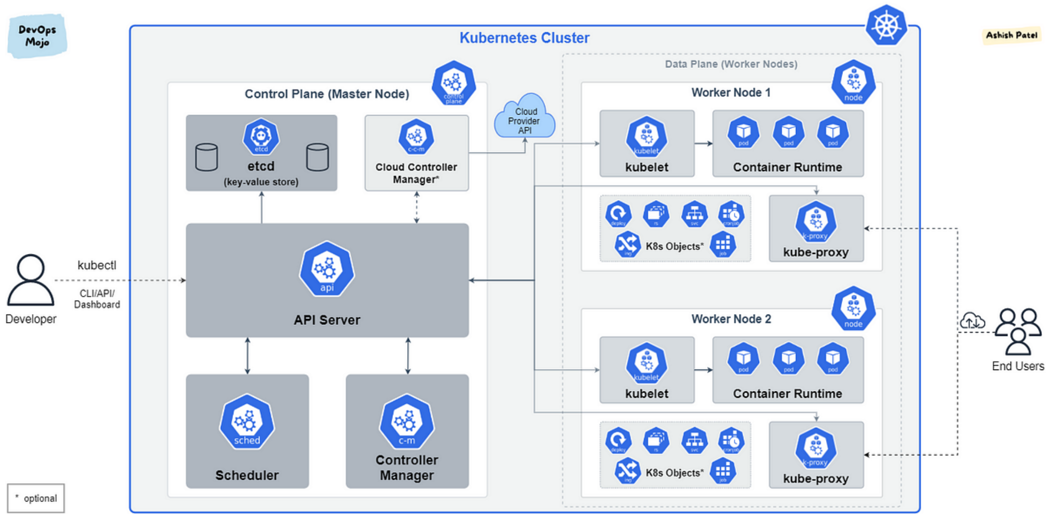


  - Most widely adopted container orchestration platform.
  - Manages containerized applications across clusters of machines.
  - Handles self-healing, automated rollouts, and scaling.
- **Docker Swarm**:
  - Built-in Docker tool for orchestration.
  - Easier to set up but less feature-rich compared to Kubernetes.
  - Ideal for smaller setups with Docker-native capabilities.
- **Apache Mesos**:
  - General-purpose distributed systems platform that supports container orchestration.
  - Suitable for large-scale environments requiring both container and non-container workloads.

▼ Kubernetes Architecture Overview

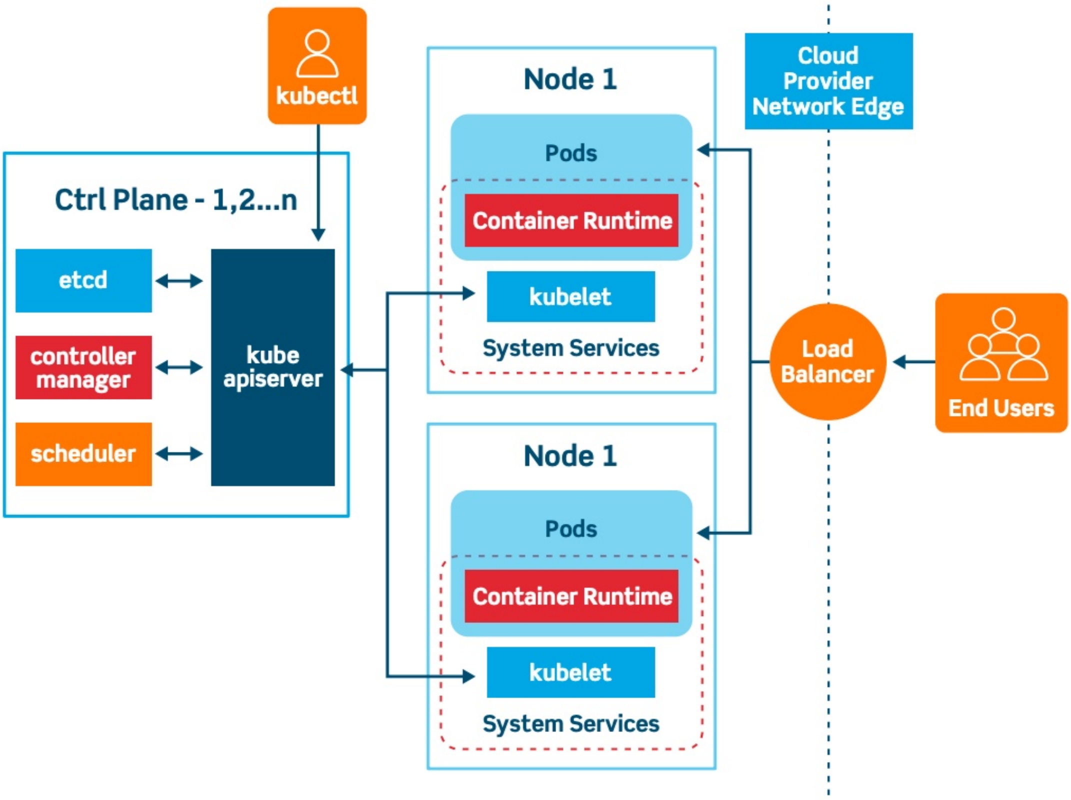
- **Master Node**: Manages the Kubernetes cluster.
  - **API Server**: Entry point for REST operations.
  - **Scheduler**: Assigns containers to nodes.
  - **Controller Manager**: Ensures the desired state of the system.
- **Worker Nodes**: Hosts running containerized applications.
  - **Kubelet**: Ensures containers are running on a node.
  - **Pod**: Smallest deployable unit consisting of one or more containers.

- **Kube-Proxy**: Handles networking within Kubernetes.



Key Kubernetes Concepts

- **Pod**: A group of one or more containers, with shared storage and network resources.
- **Service**: An abstraction that defines a logical set of pods and a policy for accessing them.
- **Deployment**: Manages pod scaling and rolling updates for your application.
- **Namespace**: Provides scope for resources within a Kubernetes cluster, helping organize and manage resources.



▼ How Orchestration Benefits DevOps

- **Automation**: Simplifies repetitive tasks such as deployment, scaling, and rollback.
- **High Availability**: Distributes workloads across different machines, ensuring that services remain available.
- **Fault Tolerance**: Automatically restarts or replaces failed containers and reroutes traffic to healthy containers.
- **Scalability**: Orchestrators can dynamically scale the number of running containers to handle increased traffic.

▼ How to Setup Kubernetes:

▼ On macOS

1. **Install Minikube**:
    - Minikube is a tool that runs a single-node Kubernetes cluster locally.
    - Command: `brew install minikube` (for macOS)

2. **Start Minikube**:
    - Command: `minikube start`
    - This will spin up a local Kubernetes cluster on your machine.
3. **Deploy an Application**:
    - Use `kubectl` to deploy a container to your Kubernetes cluster.
    - Example: `kubectl create deployment hello-world --image=k8s.gcr.io/echoserver:1.4`
4. **Expose the Application**:
    - Command: `kubectl expose deployment hello-world --type=NodePort --port=8080`
    - This exposes the application to the internet, allowing users to access it.
5. **Scale the Application**:
    - Command: `kubectl scale deployment hello-world --replicas=3`
    - This scales the application to run three instances of the container.

▼ On Windows

1. **Install Docker Desktop for Windows**

- **Why?** Docker Desktop comes with a built-in Kubernetes option that allows for a simple installation and setup.
- **Instructions**:
    1. Download and install **Docker Desktop for Windows** from the official site.
    2. During the installation, make sure **"Enable Kubernetes"** is selected.
    3. Once installed, open Docker Desktop and navigate to **Settings** > **Kubernetes**.
    4. Enable **Kubernetes** and apply the changes.
    5. Wait for Kubernetes to start, which may take a few minutes.

2. **Install** `kubectl`

- `kubectl` is the command-line tool for interacting with Kubernetes.
    1. Download the **kubectl.exe** binary for Windows from the official Kubernetes site.
    2. Add the binary's path to your **system PATH** for easy access from the command line.

3. **Verify Installation**
    - Open a terminal (CMD or PowerShell).
    - Run `kubectl version` to check if Kubernetes is installed properly.
    - Run `kubectl get nodes` to see if the local cluster is running.
4. **Minikube (Alternative)**
    - If you don't want to use Docker Desktop, you can set up Kubernetes with **Minikube**:
        1. Download **Minikube** from the official site.
        2. Install Minikube using the installer.
        3. Run `minikube start` to set up a single-node Kubernetes cluster.

▼ On Linux:

1. **Install Docker**
    - On Linux, Docker needs to be installed to manage containers.

      Update the system:

      ```
      sudo apt-get update
      sudo apt-get install -y docker.io
      ```

      Enable and start Docker:

      ```
      sudo systemctl enable docker
      sudo systemctl start docker
      ```
2. **Install Minikube**
    - Minikube allows you to run Kubernetes on a single node.

      Download Minikube:

```
    curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
    sudo install minikube-linux-amd64 /usr/local/bin/minikube
    ```

    Start Minikube:

    ```
    minikube start
    ```

3. **Install `kubectl`:**

   ```
   sudo apt-get install -y apt-transport-https
   curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
   echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a /etc/apt/sources.list.d/kubernetes.list
   sudo apt-get update
   sudo apt-get install -y kubectl
   ```

   Verify the installation by running `kubectl version`.

4. **Running Kubernetes**

   - Use the following commands to start interacting with your Kubernetes cluster:

     ```
     kubectl cluster-info
     kubectl get nodes
     ```

   - You can deploy containers and pods using `kubectl apply -f <your-deployment-file>.yaml`.

5. **Manage Kubernetes with Helm (Optional)**

   - **Helm** is a package manager for Kubernetes that makes deployment easier.

     Install Helm:

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash
```

Start deploying applications using **Helm Charts**.

Deploying `song-suggester` App with Kubernetes

1. **Step 1**: Create a Docker image for the suggest-music app.
    - Command: `docker build -t song-suggester .`
2. **Step 2**: Deploy the Docker container in a Kubernetes pod.
    - Example: Use `kubectl create deployment song-suggester --image=song-suggester`
3. **Step 3**: Expose the app using a service to make it accessible outside the Kubernetes cluster.
    - Command: `kubectl expose deployment song-suggester --type=LoadBalancer --port=8080`
4. **Step 4**: Scale the application to run multiple instances.
    - Command: `kubectl scale deployment song-suggester --replicas=5`

Challenges with Container Orchestration

- **Complexity**: Orchestration platforms can introduce significant complexity, especially for small teams.
- **Learning Curve**: Tools like Kubernetes have a steep learning curve for new users.
- **Resource Overhead**: Orchestrators can consume considerable resources, particularly when managing large-scale systems.
- **Networking**: Configuring secure and reliable networking between containers can be challenging.