🧭 Kubernetes Flow – A Deep Dive

🏗️ Kubernetes Architecture Overview

At a high level, Kubernetes has a **control plane** and a **set of worker nodes**:

🔧 Control Plane Components

Component	Purpose
`kube-apiserver`	Entry point for all Kubernetes comma
`etcd`	Key-value store for all cluster data (co
`kube-scheduler`	Assigns pods to nodes.
`controller-manager`	Manages controllers (replication, end

🧱 Node Components

Component	Purpose
`kubelet`	Runs on each worker node, ensures co
`kube-proxy`	Maintains networking rules and forwar
`container runtime`	(e.g., containerd or Docker) Runs conta

🔄 Lifecycle of a Pod Creation (End-to-End Flow)

Let's see what happens when a user runs:

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

1️⃣ `kubectl` → `kube-apiserver`

- `kubectl` is the CLI tool.
- It serializes `pod.yaml` into a REST API request.
- It sends a POST request to the API server, e.g., `POST /api/v1/namespaces/default/pods`.

2️⃣ `kube-apiserver` → `etcd`

- The `kube-apiserver` is the **front controller** of the control plane.
- It **validates** the pod spec (authentication, authorization, admission control).
- If valid, it **writes the Pod object** into `etcd`.

etcd stores:

yaml Copy Edit

```yaml
/api/v1/namespaces/default/pods/mypod
```

3️⃣ `kube-scheduler` Watches Unscheduled Pods

- The **scheduler** continuously **watches the API server** for Pods with `spec.nodeName: null`.
- It selects a node based on:
  - Resource availability
  - Affinity/anti-affinity
  - Taints/tolerations
  - Node selectors, etc.

It then **updates the Pod spec** with:

yaml Copy Edit

```yaml
spec: nodeName: "worker-node-2"
```

This change is saved back to `etcd` via the API server.

4️⃣ `kubelet` Watches Assigned Pods

- Each node runs a **kubelet**, which watches the API server for **Pod objects scheduled to its node**.
- When it sees a new pod, the kubelet:
  - Pulls the container image
  - Creates container via **container runtime**
  - Mounts volumes, sets up networking
  - Starts the container(s)

5️⃣ Pod is Running 🎉

- The kubelet continuously:
  - Probes liveness/readiness
  - Sends status updates to API server

6️⃣ Controllers in Background

- **ReplicaSetController**: Watches Deployments and maintains number of replicas.
- **JobController**, **DaemonSetController**, etc., all use the same pattern:
  - Watch → Reconcile → Update desired state.

🧠 Detailed Example: What Happens Internally?

Imagine this Pod spec:

```yaml
apiVersion: v1 kind: Pod metadata: name: hello-world spec: containers: - name: hello image:
busybox command: ["echo", "Hello Kubernetes!"]
```

🔄 Internals (Step-by-step)

1. **User** runs: `kubectl apply -f pod.yaml`
2. **kubectl** sends HTTP request to API server.
3. **API server** does:
   - Authentication (e.g., via TLS certs, tokens).
   - Authorization (e.g., RBAC).
   - Admission Control (validations, limits).
   - Writes to **etcd** (desired state).
4. **Scheduler** detects unscheduled Pod and assigns it to `node-x`.
5. **API server** updates etcd with assigned node.
6. **kubelet on node-x** detects new Pod spec via API.
7. **kubelet** starts the Pod using `containerd`.
8. **Pod is created and started.**
9. **kubelet** reports status to the API server, which stores it in etcd.
10. **kubectl get pod hello-world** shows:

    ```sql
    NAME          STATUS    AGE
    hello-world   Running   10s
    ```

📈 Kubernetes Flow Summary (Diagram)

```lua
+---------+       HTTP       +---------------+       etcd         +-----------+
| kubectl |  ─────────────→  | kube-apiserver |  ─────────────→  |  etcd     |
+---------+                  +---------------+                   +----------+
                                  ↓
                         +-----------------+
                         | kube-scheduler  |
                         +-----------------+
                                  ↓
                            Pod Assigned
                                  ↓
                        +----------------+
                        | kubelet (Node) |
                        +----------------+
                                  ↓
                        +------------------+
                        | container runtime|
                        +------------------+
```

🧪 Additional Notes for Students

- `kubectl get pod -o yaml` lets you see how the Pod looks in etcd.
- Every component talks only to `kube-apiserver`.
- All reconciliation loops are controller-driven: **"desired state vs actual state."**
- `kubelet` is responsible only for its node.
- `kube-proxy` handles **services and load-balancing** using iptables or IPVS.