Kubernetes Foundations

Orchestrator • Features • Core Concepts • Architecture • Objects • YAML

- Audience: Advanced / Intermediate (beginner-accessible)
- Focus: Theory-first, vendor-neutral

Agenda

1. Kubernetes as a Container Orchestrator
2. Features Overview (scaling, healing, deployment)
3. Core Concepts (Cluster, Node, Pod)
4. Control Plane vs. Worker Nodes
5. High-Level Architecture Diagram
6. Objects & Management: Declarative vs. Imperative
7. YAML Basics for K8s Manifests

1 Kubernetes as a Container Orchestrator

Role & Problem Space

- What it is: A declarative control system that schedules containers, keeps desired state, and automates operations.
- Why it exists: To run distributed applications reliably across many machines with consistent policies.
- Key idea: You describe what you want (desired state); controllers reconcile the system to match it (actual state).
- Outcomes: Higher availability, elastic capacity, portability across infrastructure.

2 Kubernetes Features Overview

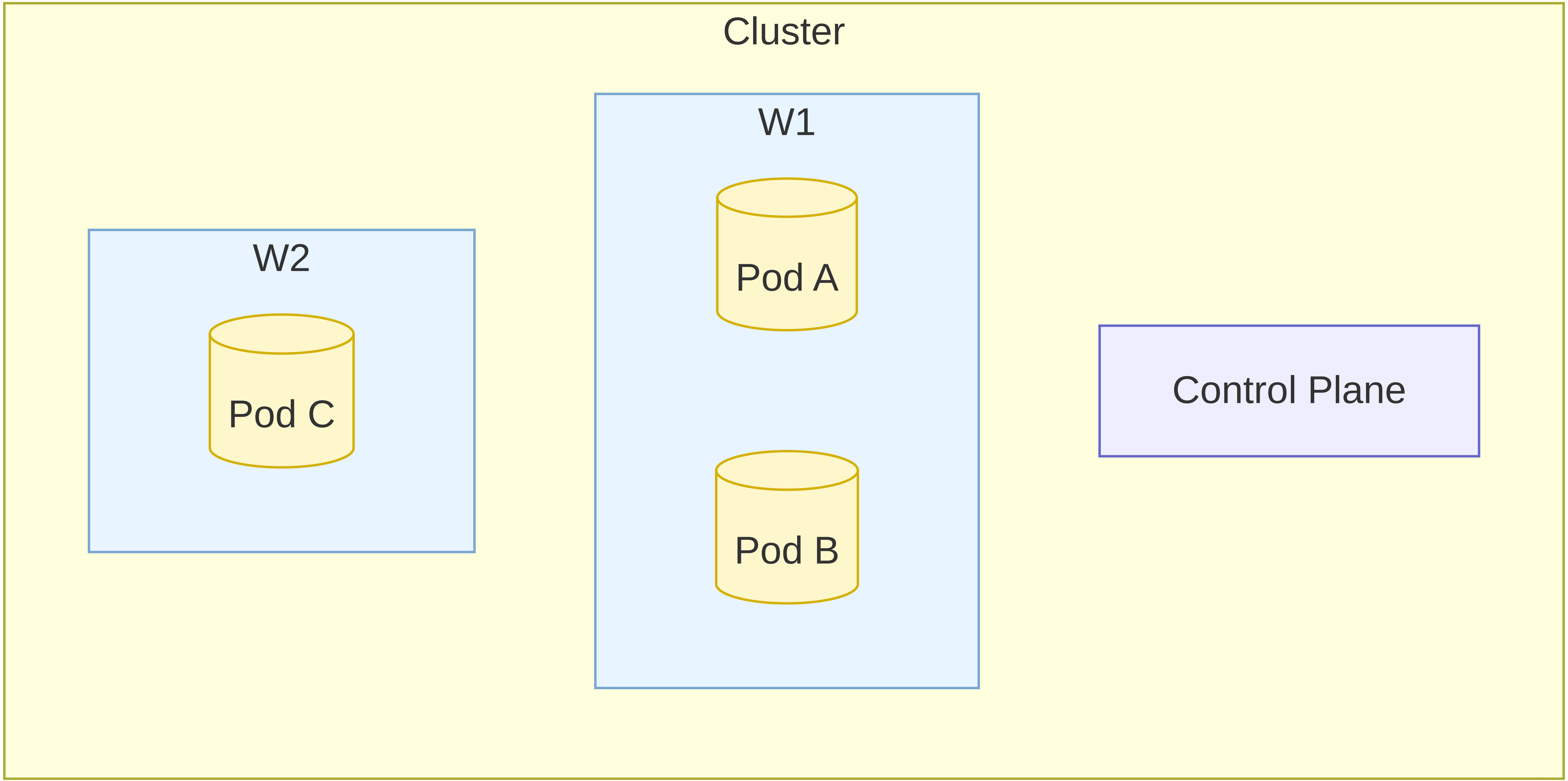
Scaling • Healing • Deployment

- Autoscaling: Horizontal Pod Autoscaling (HPA) on metrics; cluster autoscaling (node count) via cloud integrations.
- Self-healing: Controllers recreate failed pods; readiness gates shield traffic until ready.
- Service discovery & networking: Stable virtual IPs (Services), DNS, policies; CNI pluggability.
- Declarative rollout: Rolling updates with surge/unavailable controls; rollbacks via revision history.
- Config & secrets: Externalize configuration (ConfigMaps) and sensitive data (Secrets).

3 Core Concepts

Cluster • Node • Pod

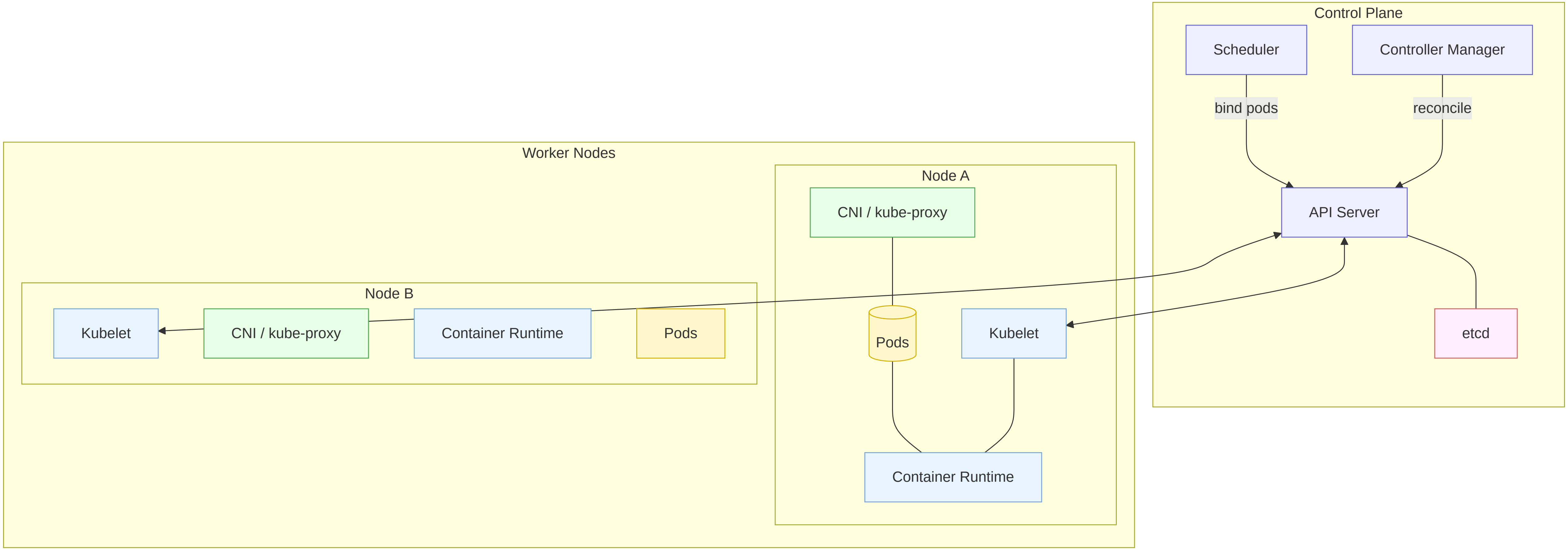
- Cluster: A set of machines (physical/virtual) managed as one logical system.
- Node: Worker machine that runs pods; includes kubelet, kube-proxy/CNI, container runtime.
- Pod: Smallest deployable unit—one or more tightly coupled containers sharing network/IP and volumes.
- Labels & selectors: Key/value tags used for grouping and targeting (services, deployments).
- Controllers: Higher-level objects (Deployment, StatefulSet, Job) that manage pods to a policy.


Cluster
W2
Pod C
W1
Pod A
Pod B
Control Plane

4 Control Plane vs. Worker Nodes

Responsibilities & Boundaries

- Control Plane (brains):
    - API Server: Front door for all requests (authn/z, admission).
    - Scheduler: Chooses nodes for unscheduled pods.
    - Controller Manager: Runs controllers (e.g., Deployment, Node, Job).
    - etcd: Consistent key-value store for cluster state.
- Workers (muscle):
    - Kubelet: Ensures containers for a pod are running as specified.
- CNI / kube-proxy: Networking & service routing.
- Container runtime: Executes containers (e.g., containerd, CRI-O).


Control Plane
Scheduler
Controller Manager
bind pods
reconcile
API Server
etcd
Worker Nodes
Node A
CNI / kube-proxy
Pods
Kubelet
Container Runtime
Node B
Kubelet
CNI / kube-proxy
Container Runtime
Pods

6 Kubernetes Objects & Management

Declarative vs. Imperative

- Imperative (do this now): Direct commands that change state immediately.
  - Examples: kubectl run, kubectl create deployment ..., kubectl scale --replicas=3.
  - Pros: Quick, ad-hoc actions; good for experiments.
  - Cons: Drift risk; history outside version control; harder to reproduce.
- Declarative (ensure this state): Store manifests; cluster reconciles to match.
  - Examples: kubectl apply -f *.yaml (server-side apply, field ownership).
  - Pros: Versioned, reviewable, repeatable; supports GitOps; safer rollouts.
  - Cons: Requires manifest discipline; change review process.

> Principle: Treat manifests as code; use Git as source of truth; prefer declarative for

Reconciliation & Change Safety

- Reconciliation: Controllers compare desired vs. actual state continuously.
- Diff & dry-run: kubectl diff and --dry-run=server preview changes.
- Ownership: Field managers prevent clobbering; labels/annotations carry intent.
- Rollback: Use controller history (Deployments) or revert manifest commits.

7 YAML Basics for K8s Manifests

Anatomy & Conventions

- Four pillars:
  - apiVersion: Which API group/version the object uses
  - kind: The object type (Deployment, Service, etc.)
  - metadata: Name, namespace, labels/annotations
  - spec: Desired state for that object
- Common patterns: Labels for selection, selectors for targeting, resource requests/limits, probes, env/config refs.

```yaml
# Minimal illustrative Deployment (theory)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
  namespace: app
  labels:
    app: web
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: app
          image: ghcr.io/org/web:1.2.3
          ports:
            - containerPort: 8080
          resources:
            requests: { cpu: "100m", memory: "128Mi" }
            limits:   { cpu: "500m", memory: "256Mi" }
          livenessProbe:
            httpGet: { path: /healthz, port: 8080 }
            initialDelaySeconds: 10
          readinessProbe:
            httpGet: { path: /ready, port: 8080 }
```

Multi-Document & Composition

- Multi-doc files: Separate objects with --- in one file (e.g., Deployment + Service).
- Kustomize/Helm: Template/overlay systems for environment-specific composition.
- Best practice: Keep base manifests minimal; add overlays/values for env differences.

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: web
  namespace: app
spec:
  selector: { app: web }
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: ClusterIP
```

Key Takeaways

- Kubernetes is a declarative orchestrator: you set desired state; controllers reconcile.
- Core units: Pods run containers; nodes host pods; the control plane drives intent.
- Features: Scaling, healing, service discovery, policy, and extensibility.
- Prefer declarative management with manifests and GitOps; keep imperative for ad-hoc.
- YAML is the interface to the API: consistent labels/selector design is crucial.

References & Further Reading

- Kubernetes Docs: Concepts, Workloads, Services/Networking, API Machinery
- Server-Side Apply & Field Management
- Labels/Selectors, Probes, Resource Management
- GitOps patterns (Argo CD, Flux)