Mastering Kubernetes Architecture

A Platform Engineer's Deep Dive

- 30 Slides of Advanced Insights
- Production-Ready Strategies
- Real-World Implementation Patterns
- Future-Proof Architecture Design

Swipe for comprehensive insights →

What We'll Cover

YOUR LEARNING JOURNEY

1 **Foundation** (Slides 3–8)

- Declarative philosophy
- Core architectural principles

2 **Control Plane** (Slides 9–16)

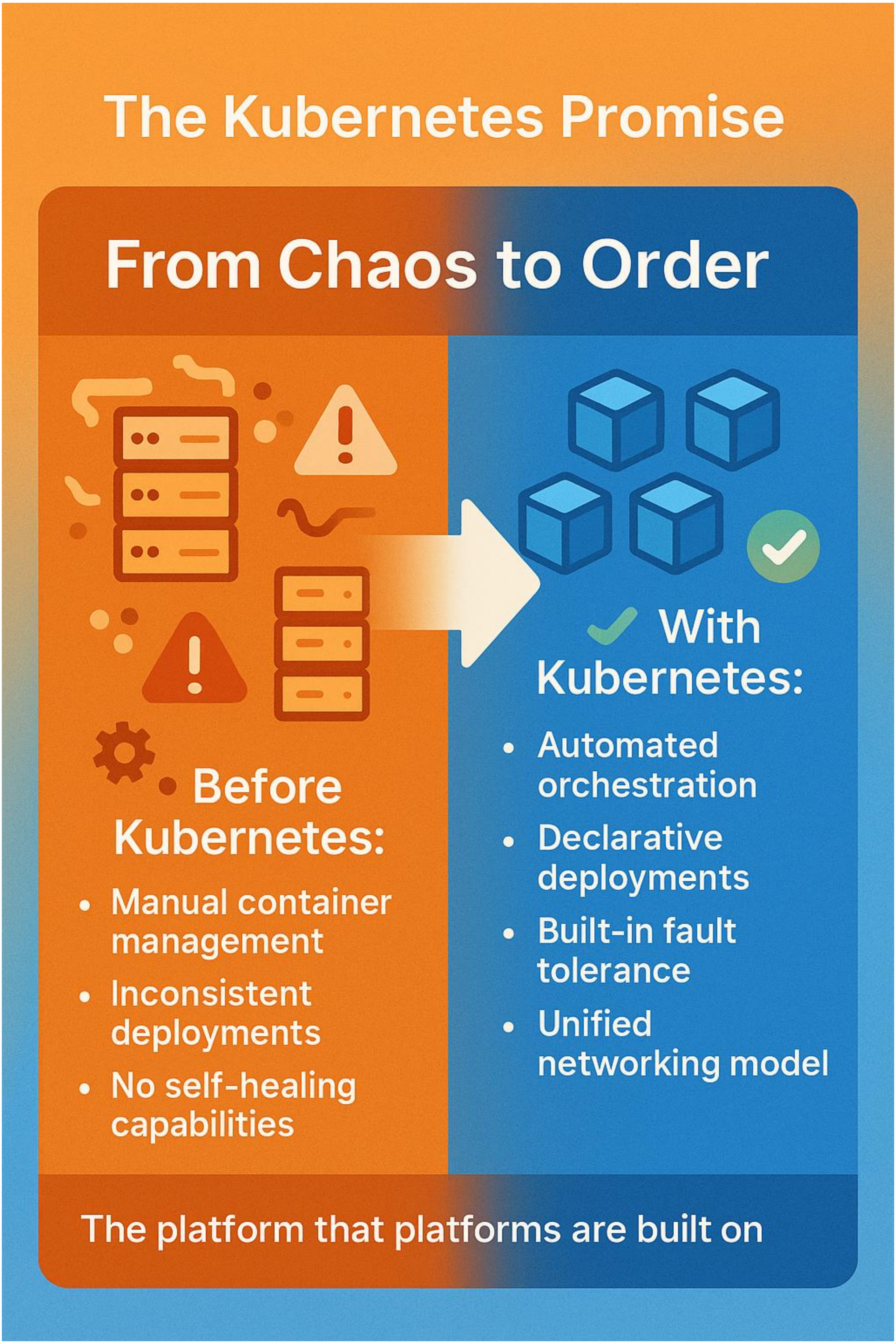
- API Server deep dive
- etcd & data persistence
- Scheduling intelligence

3 **Worker Nodes** (Slides 17–22)

- kubelet operations
- Container runtime evolution

4 **Advanced Topics** (Slides 23–30)

- Security, scaling, future trends


The Kubernetes Promise
From Chaos to Order
Before Kubernetes:
• Manual container management
• Inconsistent deployments
• No self-healing capabilities
With Kubernetes:
• Automated orchestration
• Declarative deployments
• Built-in fault tolerance
• Unified networking model
The platform that platforms are built on

Declarative vs Imperative

Two Approaches to Infrastructure

Imperative
(Traditional)

```
1. docker run
   nginx:latest
2. Configure load
   balancer
3. Set up health
   checks
4. Scale manually
   when needed
```

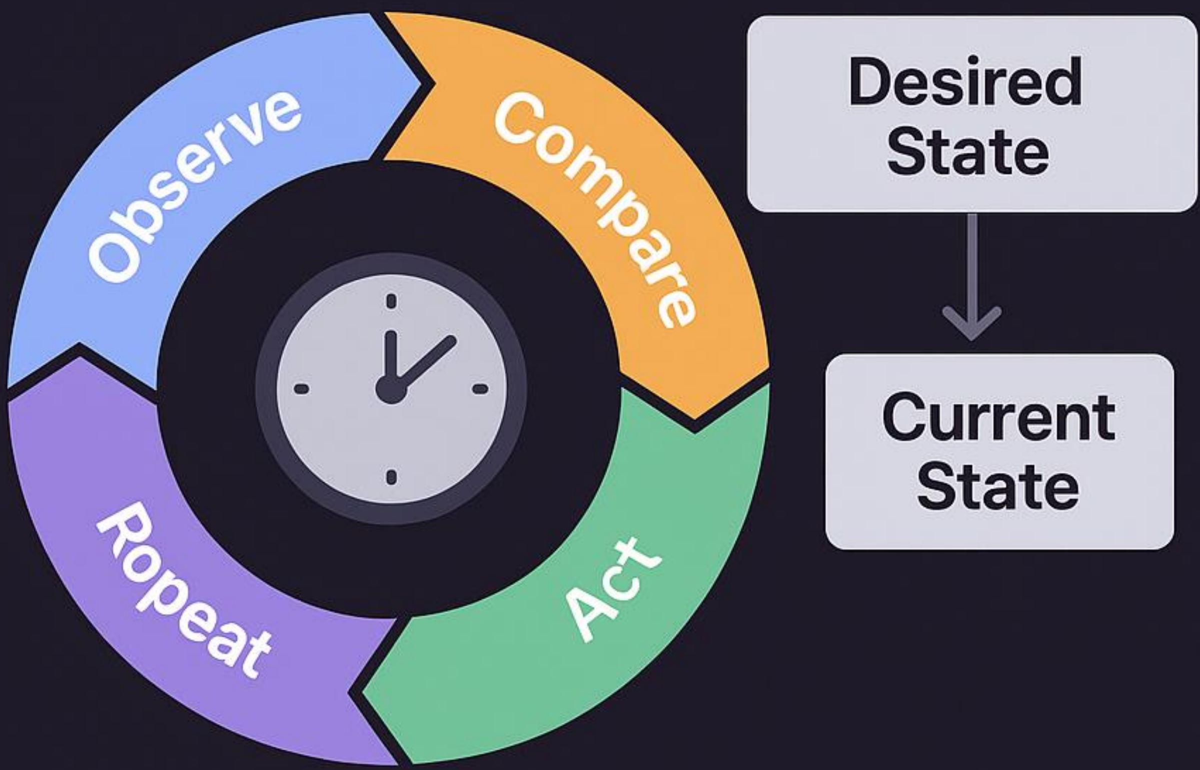
Declarative
(Kubernetes)

```
replicas: 3
image: nginx:latest
# K8s figures out
the "how"
```

Tell Kubernetes WHAT, not HOW

Continuous Reconciliation

The K8s Control Loop:

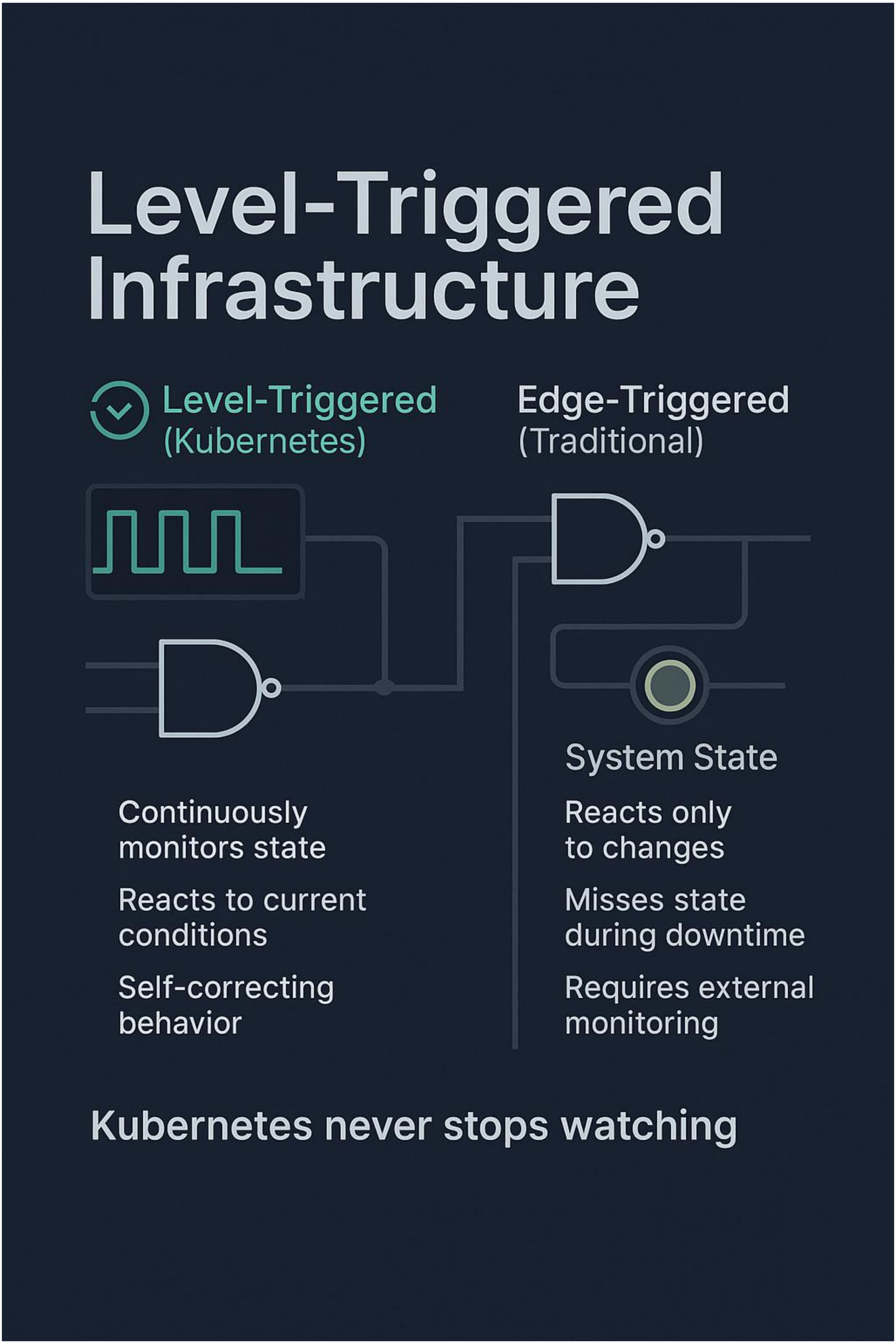


- Observe current state
- Compare with desired state
- Act to close the gap
- Repeat continuously

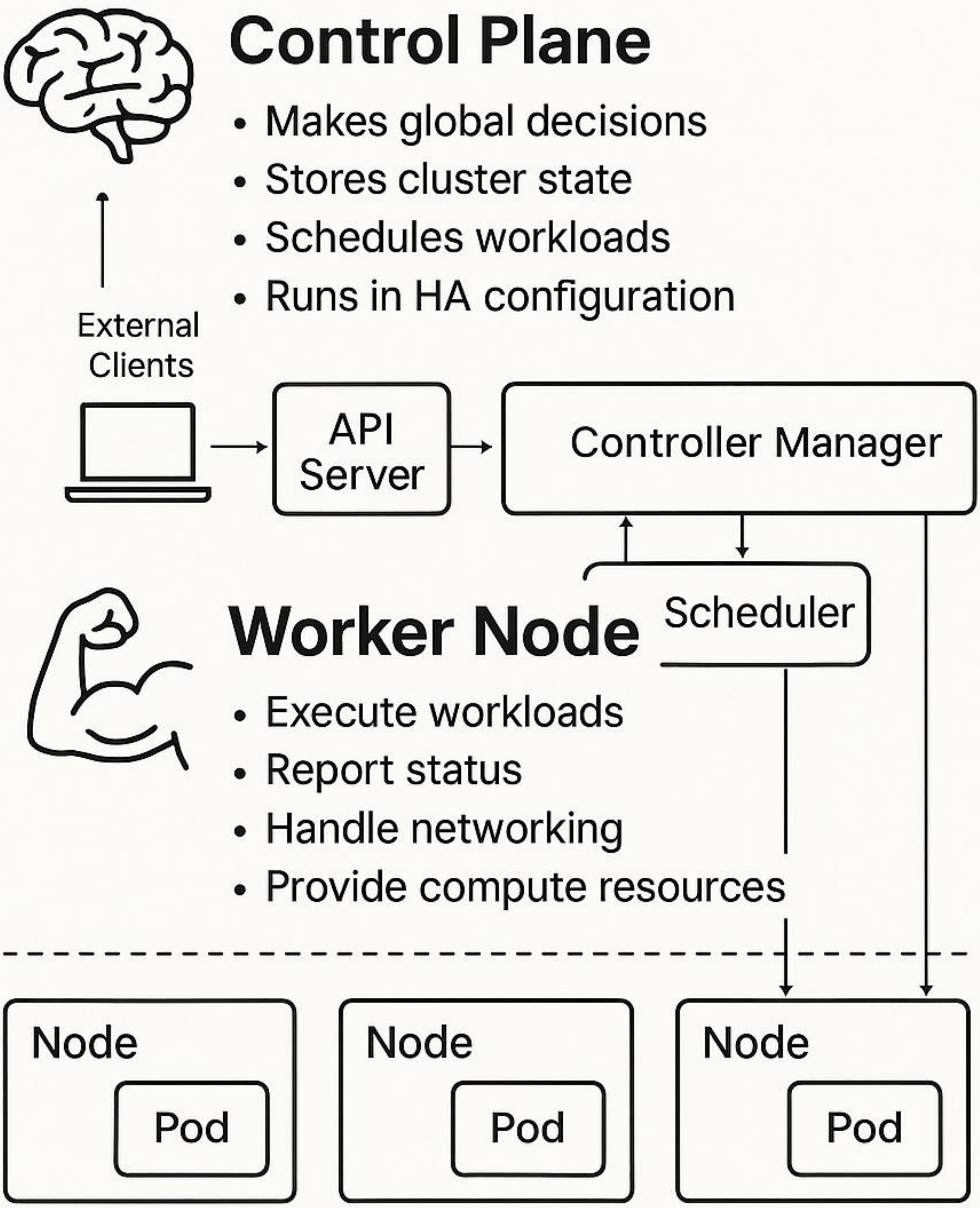
Example:

```
yamldesired: 5 replicas
current:     3 replicas
action: create 2 new pods
```

Self-healing systems in action

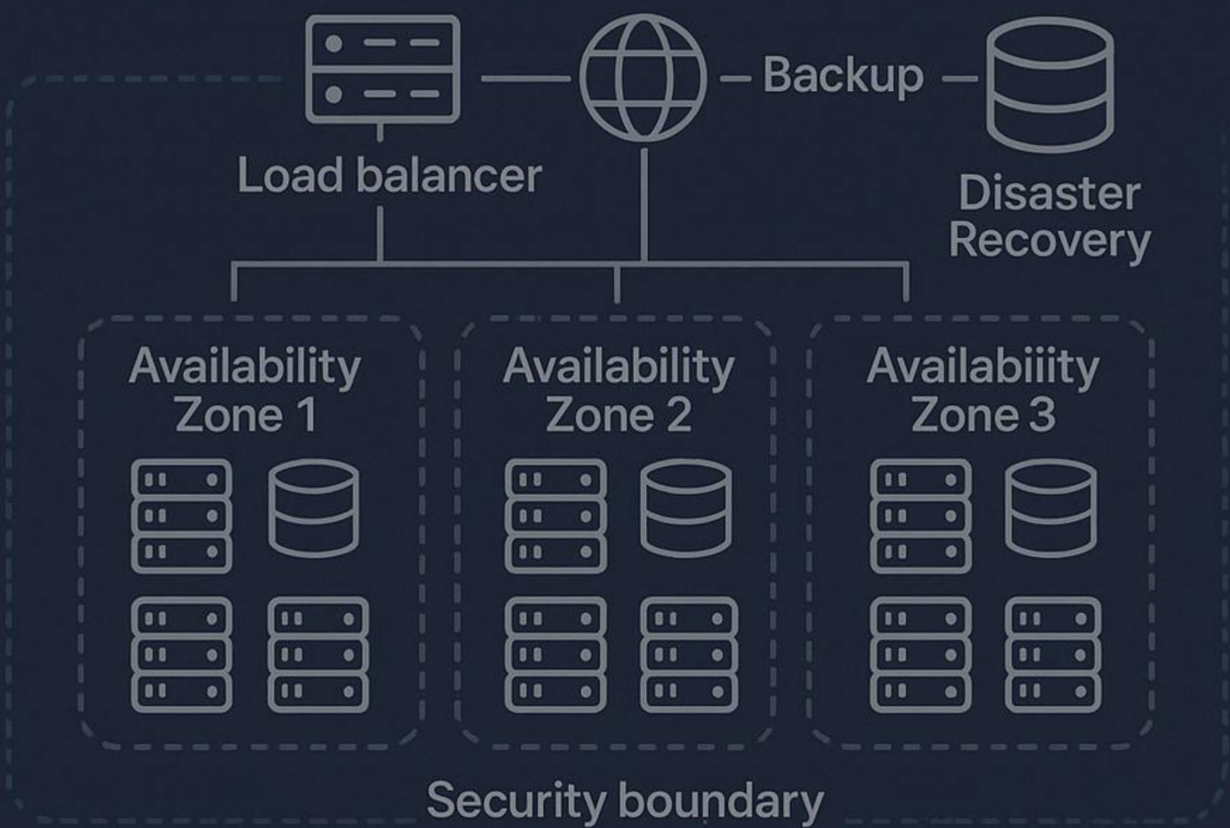

Level-Triggered Infrastructure
Level-Triggered (Kubernetes)
Edge-Triggered (Traditional)
System State
Continuously monitors state
Reacts to current conditions
Self-correcting behavior
Reacts only to changes
Misses state during downtime
Requires external monitoring
Kubernetes never stops watching

Kubernetes Cluster Anatomy



Separation of brain and brawn

Production-Ready Clusters



Enterprise Requirements

- Multi-zone deployment
- High availability control plane
- Automated backup strategies
- Security compliance (SOC2, PCI)

Scaling Targets

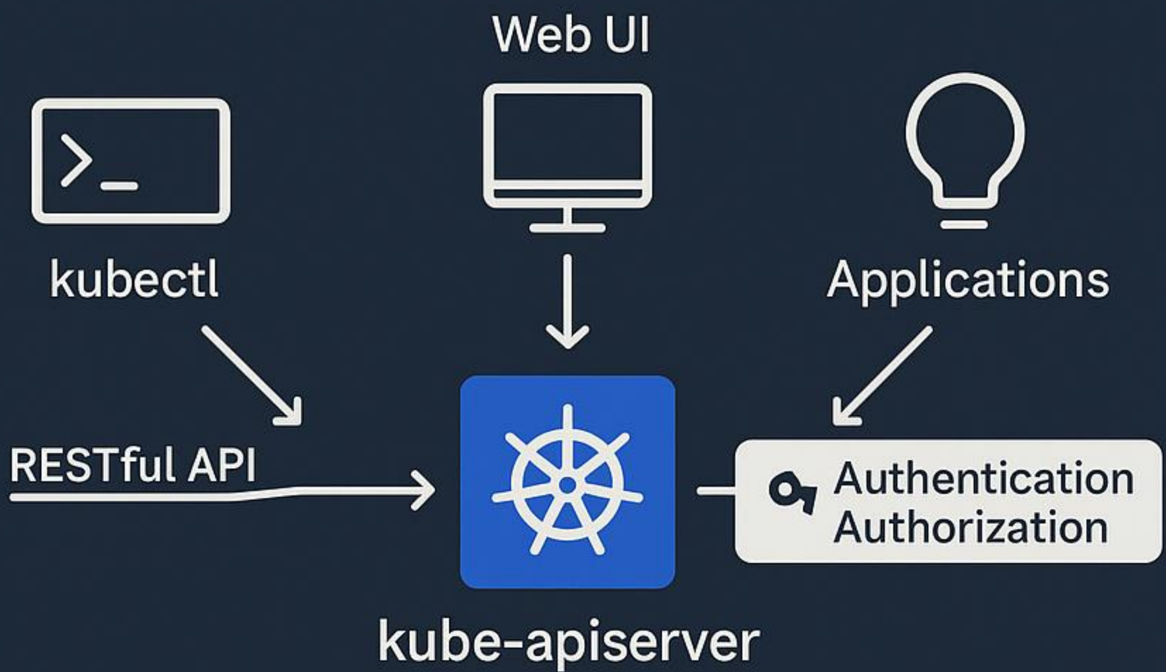
5,000 nodes maximum
150,000 pods per cluster
99.99% uptime SLA

Design for failure from day one

kube-apiserver: The Central Hub

Core Responsibilities:

- RESTful API exposure
- Authentication & authorization
- Request validation
- State persistence coordination

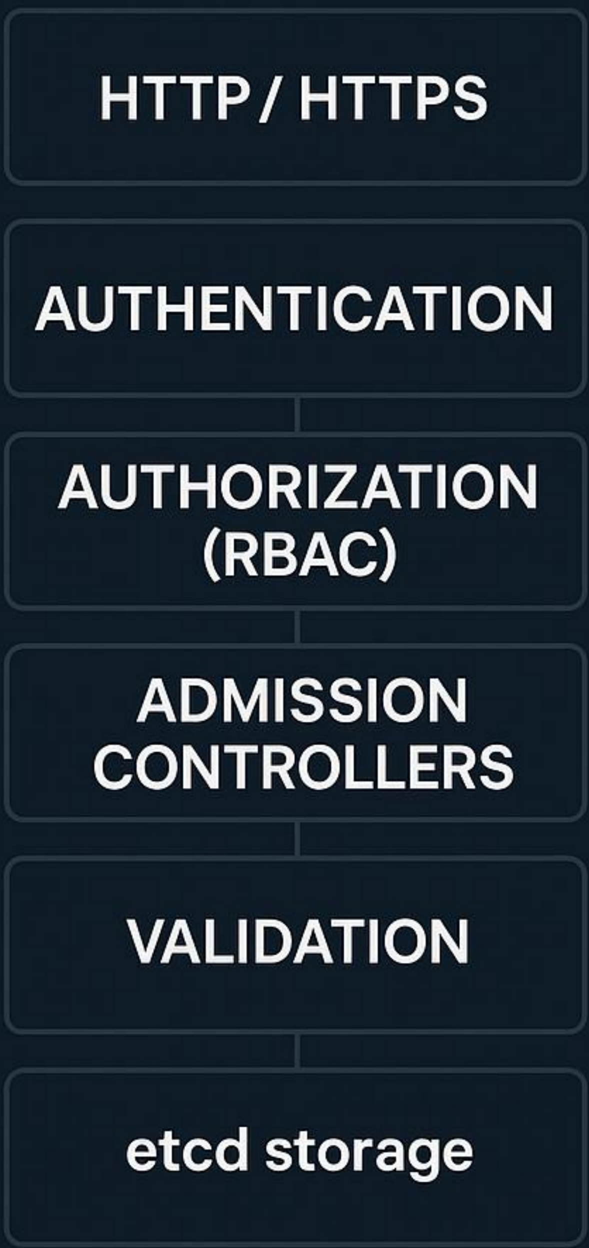


Performance Characteristics:

```
bash --max-requests-inflight=400
```

```
--max-mutating-requests-inflight=200
```

Every Kubernetes operation flows through here



API Server Request Pipeline

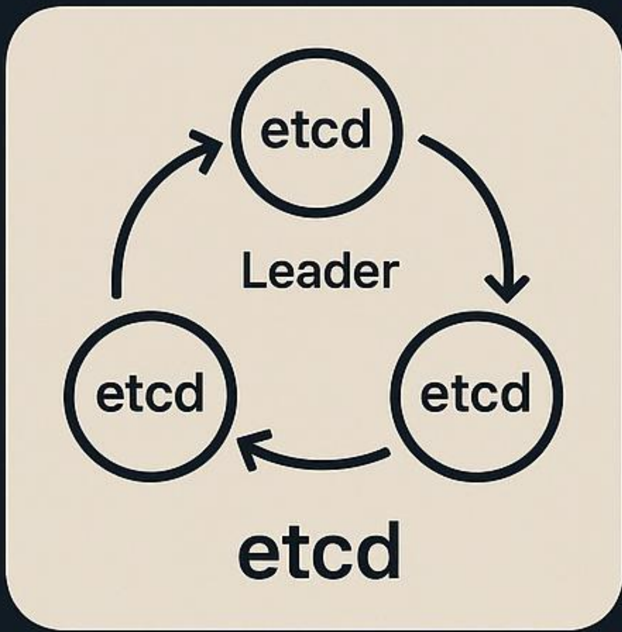
🔒 **Security Layers:**

- **Authentication**
  Who are you?
- **Authorization**
  What can you do?
- **Admission Control**
  Should we allow this?

⚡ **Horizontal Scaling:**

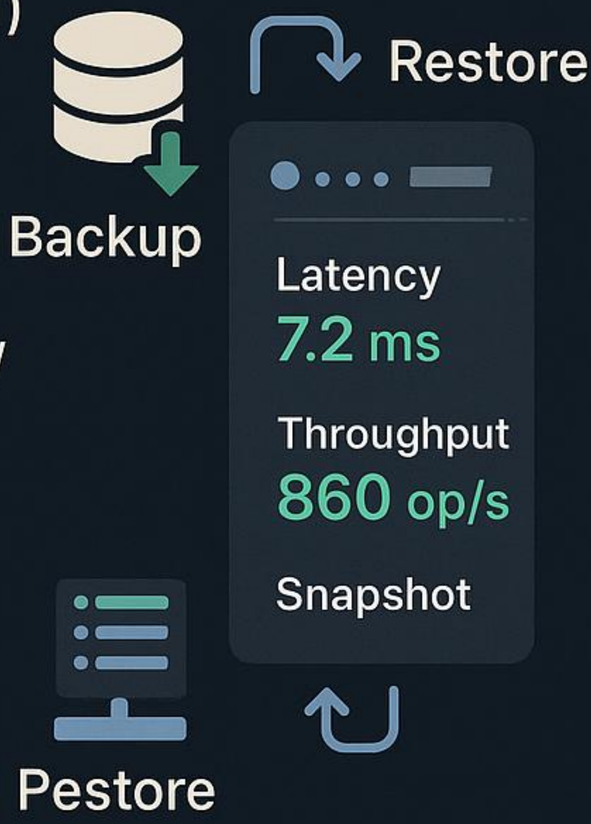
- Stateless design
- Load balancer ready
- Session affinity not requi

etcd:
Kubernetes Database



Raft Consensus:

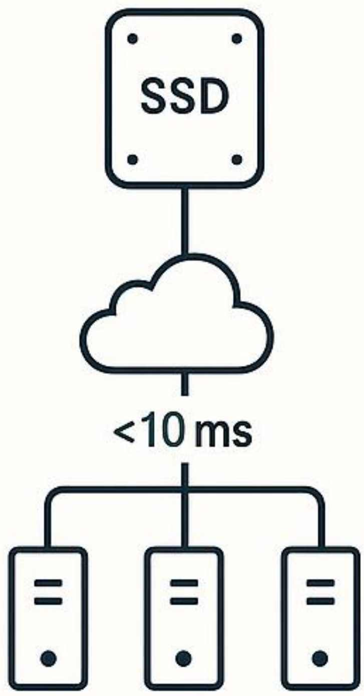
- Leader election process
- Log replication
- Strong consistency guarantee
- Fault tolerance (N/2+1)



Storage Structure:

- /registry/pods/default/web-app-xyz
- /registry/services kube-system/dns
- /registry/secrets production/db-creds
- All cluster state lives here

etcd Production Guidelines



Storage Requirements:

- Dedicated SSD storage
- Low-latency disk I/O
- 8GB+ backend storage

Network Requirements:

- <10ms latency between nodes
- Dedicated network interfaces
- Stable connectivity

Monitoring Metrics:

```
bastletcd_disk_wal_fsync_duration_seconds
etcd_network_peer_round_trip_time
```

kube-scheduler: Smart Placement

Scheduling Factors:

- Resource requests/limits
- Node capacity and availability
- Hardware constraints
- Affinity/anti-affinity rules
- Priority and preemption

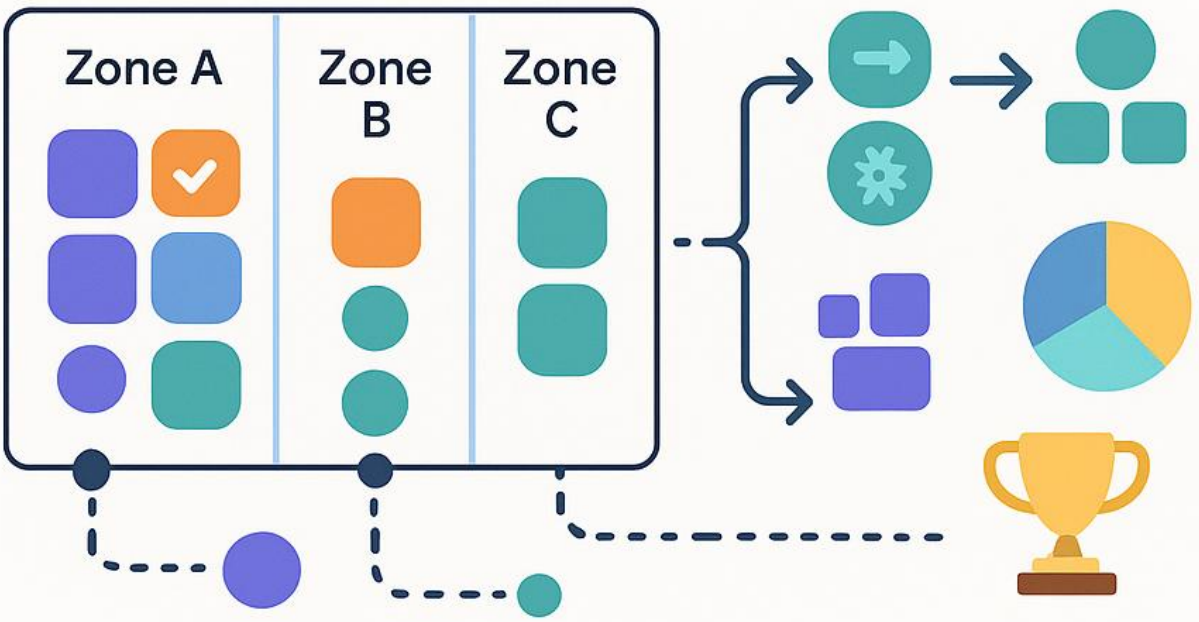
⚡ Two-Phase Process

Filtering - Find feasible nodes

Scoring - Rank and select best

10.000+ scheduling decisions per minute

Advanced Scheduling Strategies



Affinity Examples:

```
prefer differenı zones
podAntiAffinity:
  preferredDuringSchedulling
    topologyKey: failure-
  domain.beta.kubernetes.io/
nodeAffinity:
  requiredDuringScheduling
  kubernetes.io/arch: amd64
```

Priority Classes:

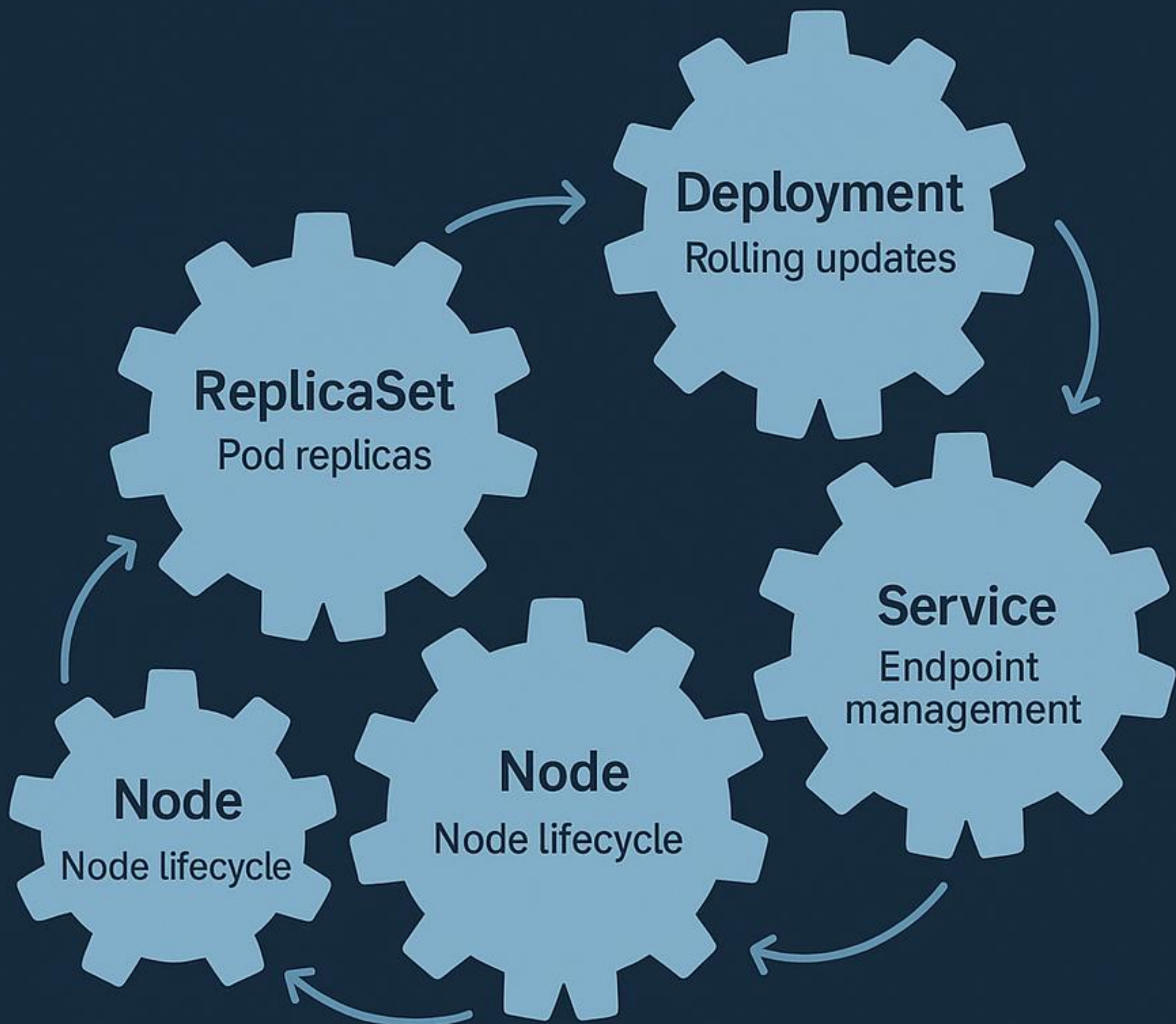
System-critical
1000+

Production
100-999

Development
0-99

CONTROLLER MANAGER: RECONCILIATION ENGINE

Built-in Controllers:



Control Loop Frequency:

```
-node-monitor-period=5s
-node-monitor-grace-period
```

Extending Kubernetes Logic

🔧 Operator Pattern:

```
apiVersion:
apiextensions.kBs.io/v1
kind:
CustomResourceDefinition
metadata:
  name databases.example.com
spec:
  group: example.com
  scope: Namespaced
```

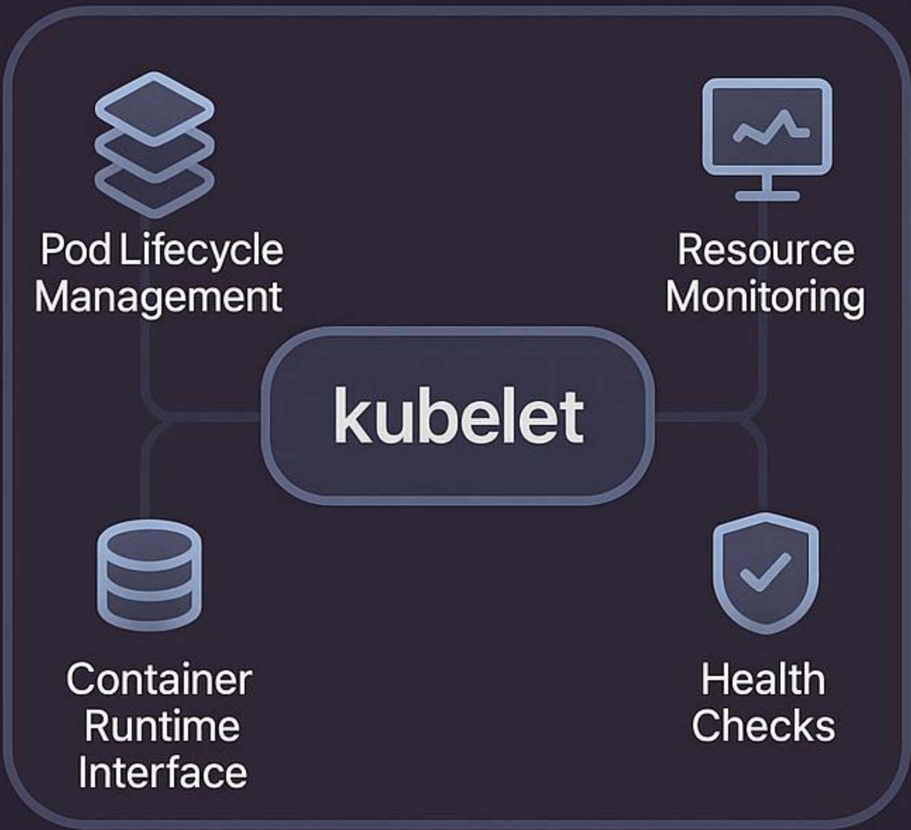
Popular Operators:

- Prometheus Operator
- Istio Operator
- PostgreSQL Operator
- cert-manager

🏭 Popular Operators:

Domain-specific automation

kubelet – The Node Agent



Core Functions:

- Pod lifecycle management
- Container runtime communication
- Resource usage reporting
- Health probe execution

Pod States:

Pending → Running → Succeeded/Failed

Health Checks:

- Liveness probes
- Readiness probes
- Startup probes

Container Runtime Landscape

Evolution Timeline:

- 2014–2020: Docker dominance
- 2017 CRI specification
- 2020 containerd default
- 2022: dockershim removal
- 2024+: WASM integration

Current Options:

- containerd Lightweight, efficient
- CRI-O OCI-compliant, minimal
- Kata Containers VM isolation

kubelet

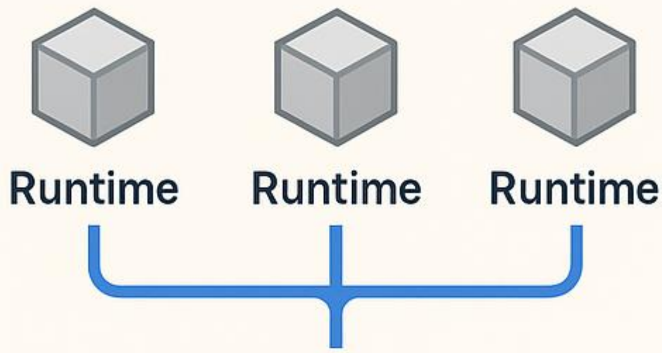
CRI: Runtime Abstraction

Interface Benefits:

- Runtime pluggability
- Standardized operations
- Implementation flexibility
- Easier testing and development

gRPC Services:

```
protobuf service
RuntimeService {
  rpc RunPodSandbox(...)
  rpc CreateContainer(...
  rpc StartContainer(...)
}
```



Kubernetes ❤ Runtime Diversity

kube-proxy: Traffic Director

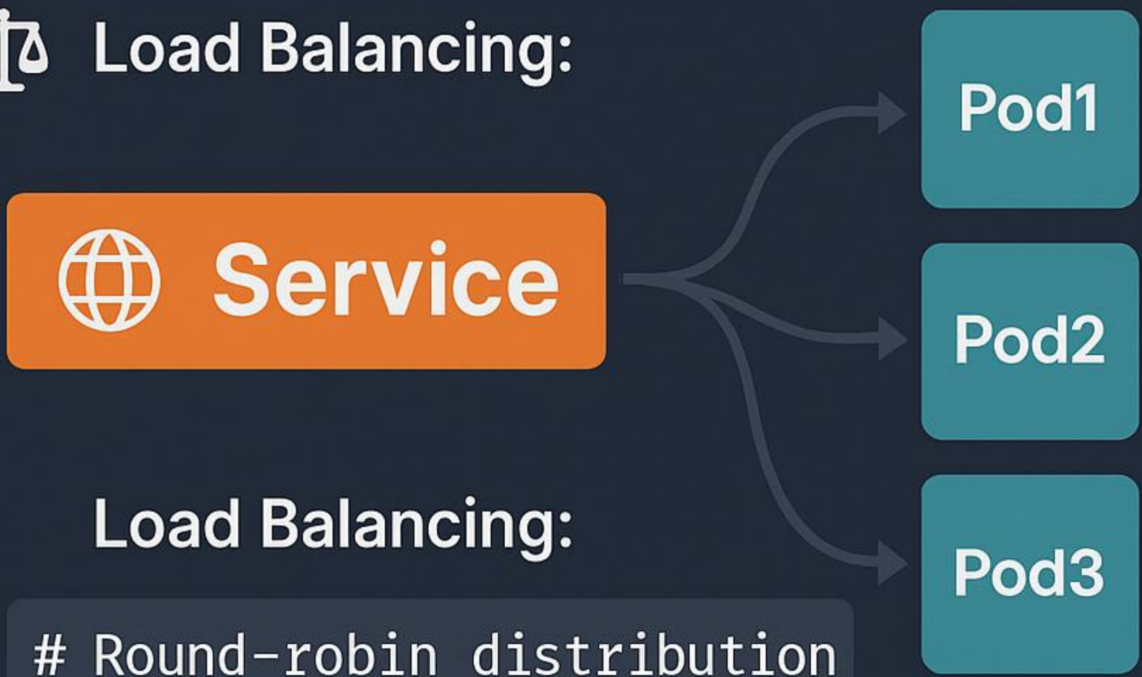
Implementation Modes:

iptables – Default, 1K services

IPVS – High performance, 10K+ services

userspace – Legacy, debugging only

Load Balancing:



Load Balancing:

```
# Round-robin distribution
Service → Pod1 (33%)
Service → Pod2 (33%)
Service → Pod3 (34%)
```

* Every node is a load balancer

Pod Networking Fundamentals

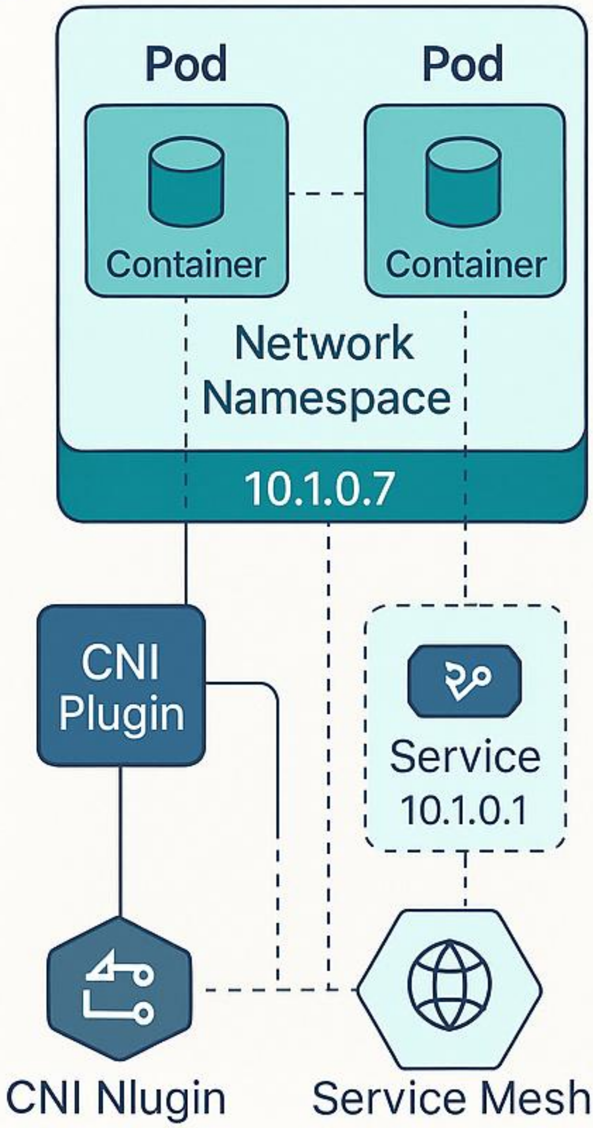
🌐 Networking Requirements:

- Every pod gets unique IP
- Pods can communicate without NAT
- Containers share network namespace
- Services provide stable endpoints

CNI Responsibilities:

 # Pod creation flow

1. kubelet calls CNi plugin
2. Plugin allocates IP address
3. Sets up network interface
4. Configures routing rules



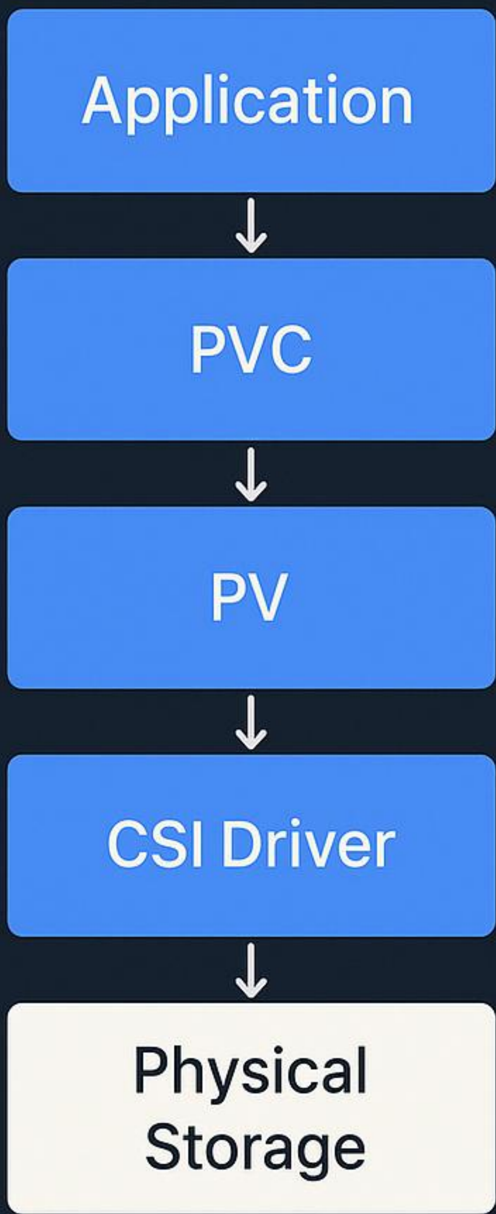
Kubernetes Storage Stack

📖 Storage Abstraction:

```
yamlPersistentVol-
umeClaim (PVC)
```

🔄 Dynamic Provisioning:

- StorageClass defines policy
- PVC triggers provisioning
- CSI driver creates volume
- Automatic binding



Kubernetes API Excellence

🏗 RESTful Design:

```
GET  /api/v1/pods    # List
POST /api/v1/pods    # Create
GET  /api/v1/pods/my-pod  Read
PUT  /api/v1/pods/my-pod Update
DELETE /api/μods/my-pod Delete
```

📈 Resource Categories:

- **Workload:** Pods, Deployments
- **Service:** Services, Ingress
- **Config:** ConfigMaps, Secrets
- **Metadata:** Labels, Annotations

API EVOLUTION MANAGEMENT

Version Lifecycle:

(v1alpha1) – Experimental

(v1beta1) – Pre-release

(v1) Production ready

Deprecation Policy:

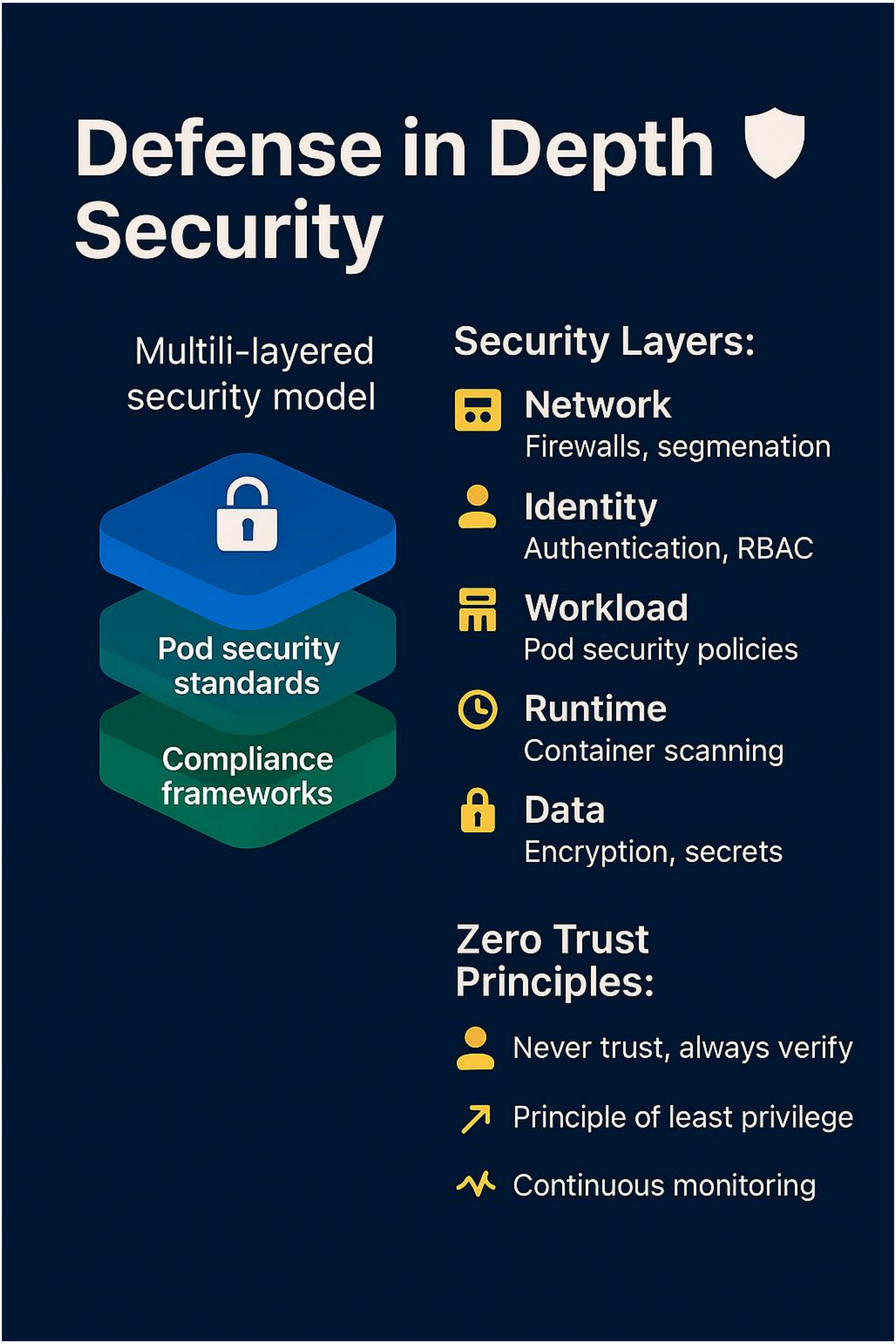
Alpha: No guarantee

Beta: 9 months minimum

Stable: 12 months minimum

Feature Gates:

```
--feature-gates=SomeNewFeature=true
```


Defense in Depth Security
Multili-layered security model
Pod security standards
Compliance frameworks
Security Layers:
Network
Firewalls, segmenation
Identity
Authentication, RBAC
Workload
Pod security policies
Runtime
Container scanning
Data
Encryption, secrets
Zero Trust Principles:
Never trust, always verify
Principle of least privilege
Continuous monitoring

Role-Based Access Control

RBAC Components:

Users / ServiceAccounts → Roles / ClusterRoles → Permissions (verbs on resources)

Example Implementation:

```
apiVersion: rbac.authorization.kbs.io/v1
kind: Role
rules:
  - apiGroups: ["]
    resources: ["pods"]
    verbs: ["get","list,"watch"]
```

Least privilege by default

PRODUCTION HA ARCHITECTURE

HA Components:

- API Servers: 3+ replicas
- etcd: 3/5/7 nede elustor
- Controllers: Leader election
- Workers: Multi-zone spread

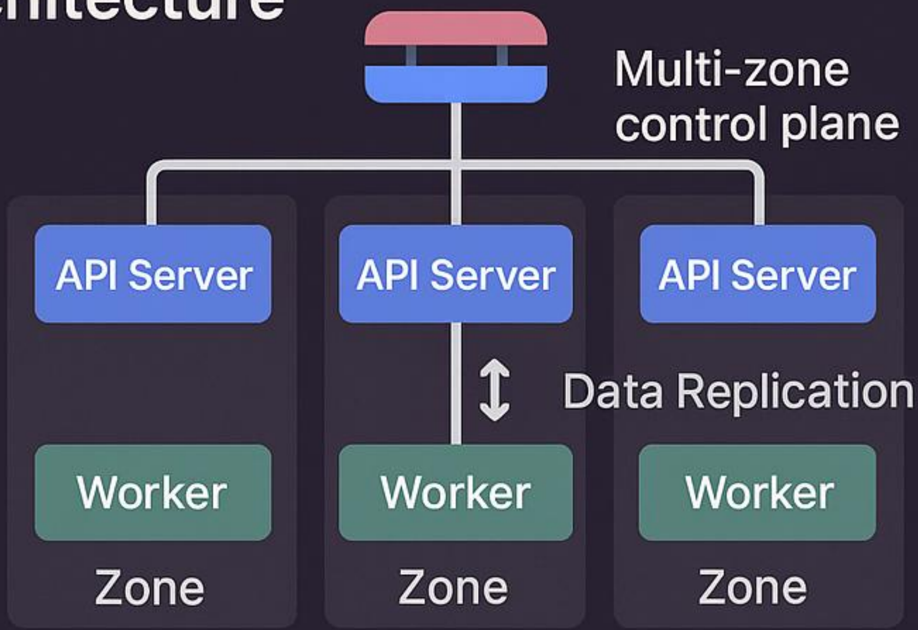
SLA Targets:

99.9% – 8.76 hours downtime/year

99.99% – 52.56 minutes/year

99.9999 – 5.26 minutes/year

Production HA Architecture



Design for inevitable failures

Intelligent Scaling Patterns

Autoscaling Types:

- HPA – Scale pods based on metrics
- VPA – Adjust resource requests
- Cluster Autoscaler – Add/remove nodes
- Custom Metrics – Business KPIs

Scaling Example:

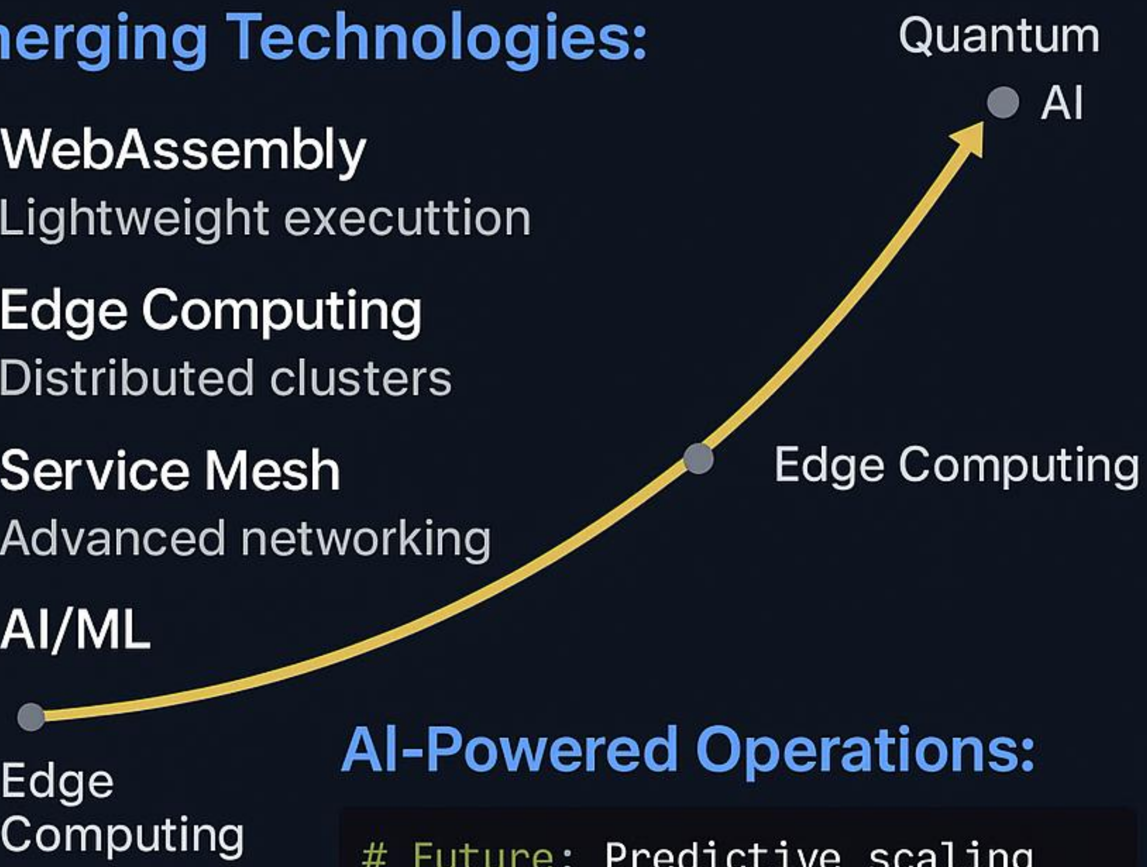
```
minReplicas: 2
maxReplicas: 100
targetCPUUtilizationPercentage
```

Scale with demand, optimize costs

The Future of Kubernetes 🚀

Emerging Technologies:

- **WebAssembly**
  Lightweight executtion
- **Edge Computing**
  Distributed clusters
- **Service Mesh**
  Advanced networking
- **AI/ML**



AI-Powered Operations:

```yaml
# Future: Predictive scaling
autoscaling:
  mode: predidive
  algorithm: machine-learning
  forecastHorizoh: 24h
```

Evolution never stops

Platform Engineer's Next Steps

🎯 Action Items:

✅ Master the fundamentals
✅ Practice with real workloads
✅ Contribute to open source
✅ Build operator expertise
✅ Stay current with CNCF landscape

⭐ Keep Learning:

- ✦ Follow CNCF project updates
- ✦ Join Kubernetes community
- ✦ Attend KubeCon conferences
- ✦ Build and share knowledge

📖 Read the full article:

https://medium.com/@salwan.mohameed/mastering-kubernetes-architecture-a-deep-dive-for-devops-and-platform-engineers-f0376c319d45

Let's connect and learn together!