# Kubernetes Workload Primitives

**Pods • Multi-Container Pods • ReplicaSets • Deployments • StatefulSets • DaemonSets • Jobs & CronJobs**

- Audience: Advanced / Intermediate (beginner-accessible)
- Focus: Theory, patterns, and controller semantics

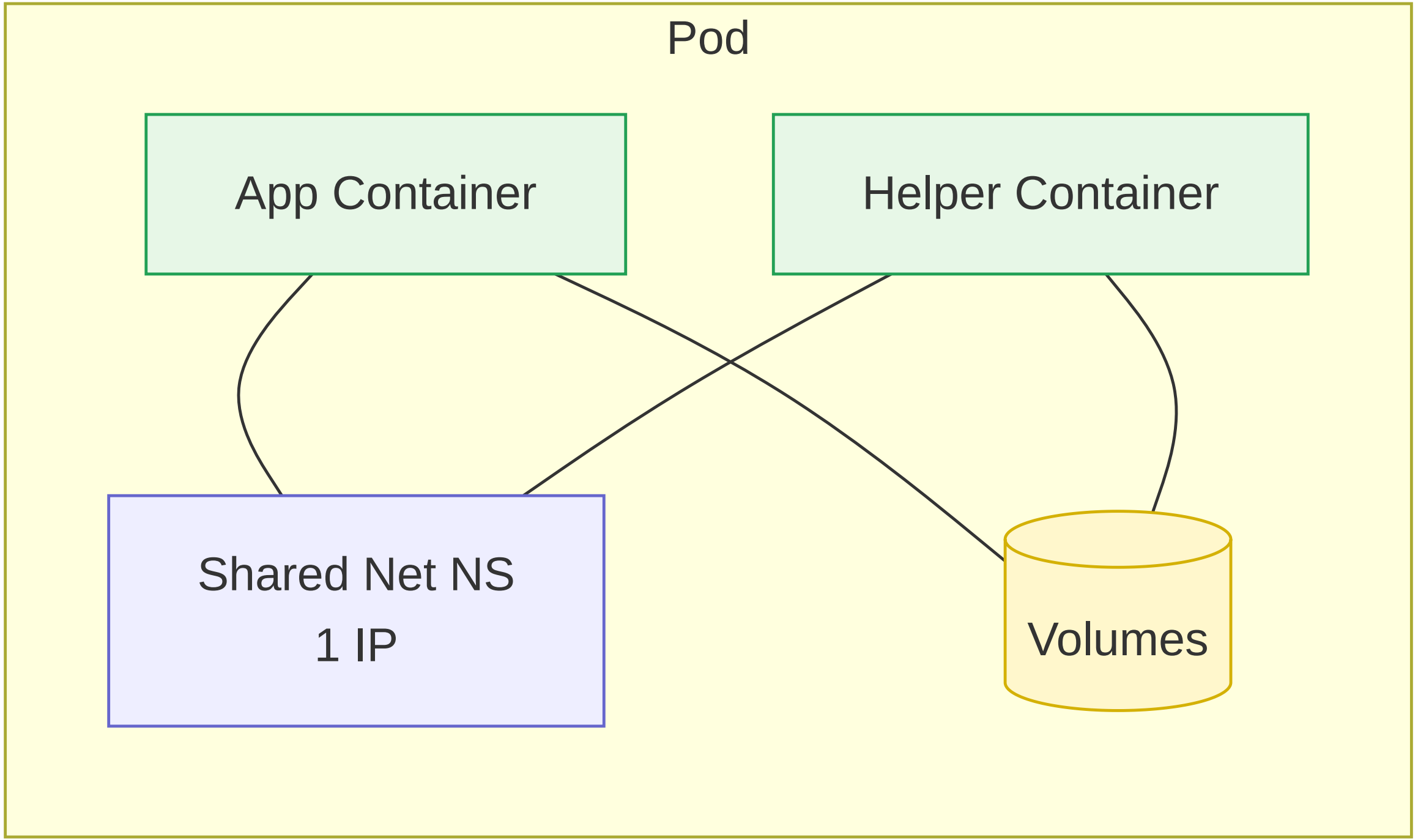
# Agenda

1. Pods (smallest deployable unit)
2. Multi-Container Pods (sidecar, ambassador, adapter)
3. ReplicaSets (ensuring pod availability)
4. Deployments (rolling updates, rollbacks)
5. StatefulSets (stateful applications)
6. DaemonSets (system-level workloads)
7. Jobs & CronJobs (batch & scheduled tasks)

# 1 Pods

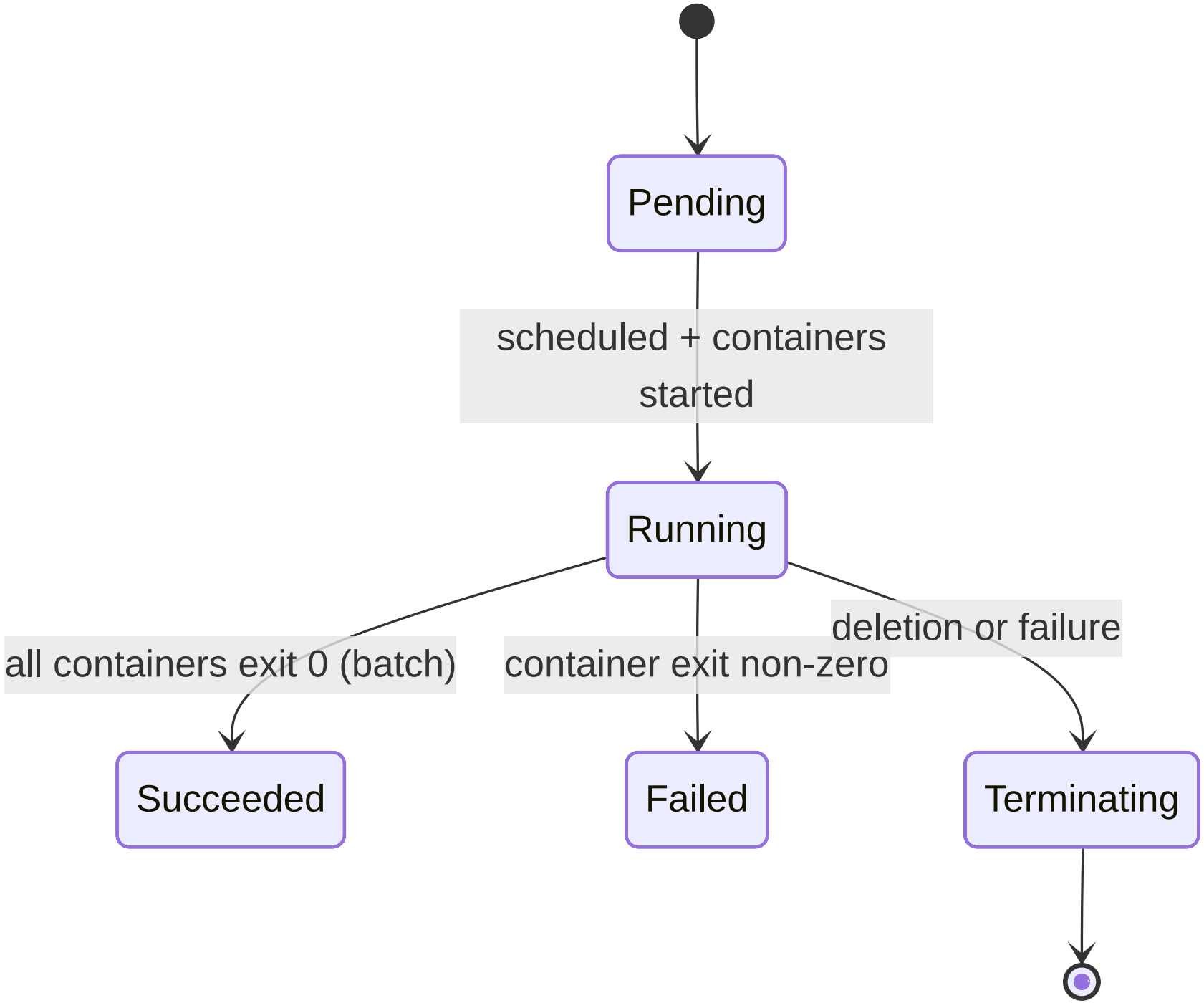
## Concept & Identity

- Atomic scheduling unit: One or more tightly coupled containers
- Shared context: Linux namespaces → same network namespace (localhost), shared IPC; per-container PID/UTS optional
- Pod identity: Single IP, shared volumes, shared service account
- Ephemeral by design: Pods are mortal; higher-level controllers manage lifecycle


Pod
App Container
Helper Container
Shared Net NS
1 IP
Volumes

# Pod Lifecycle (High Level)

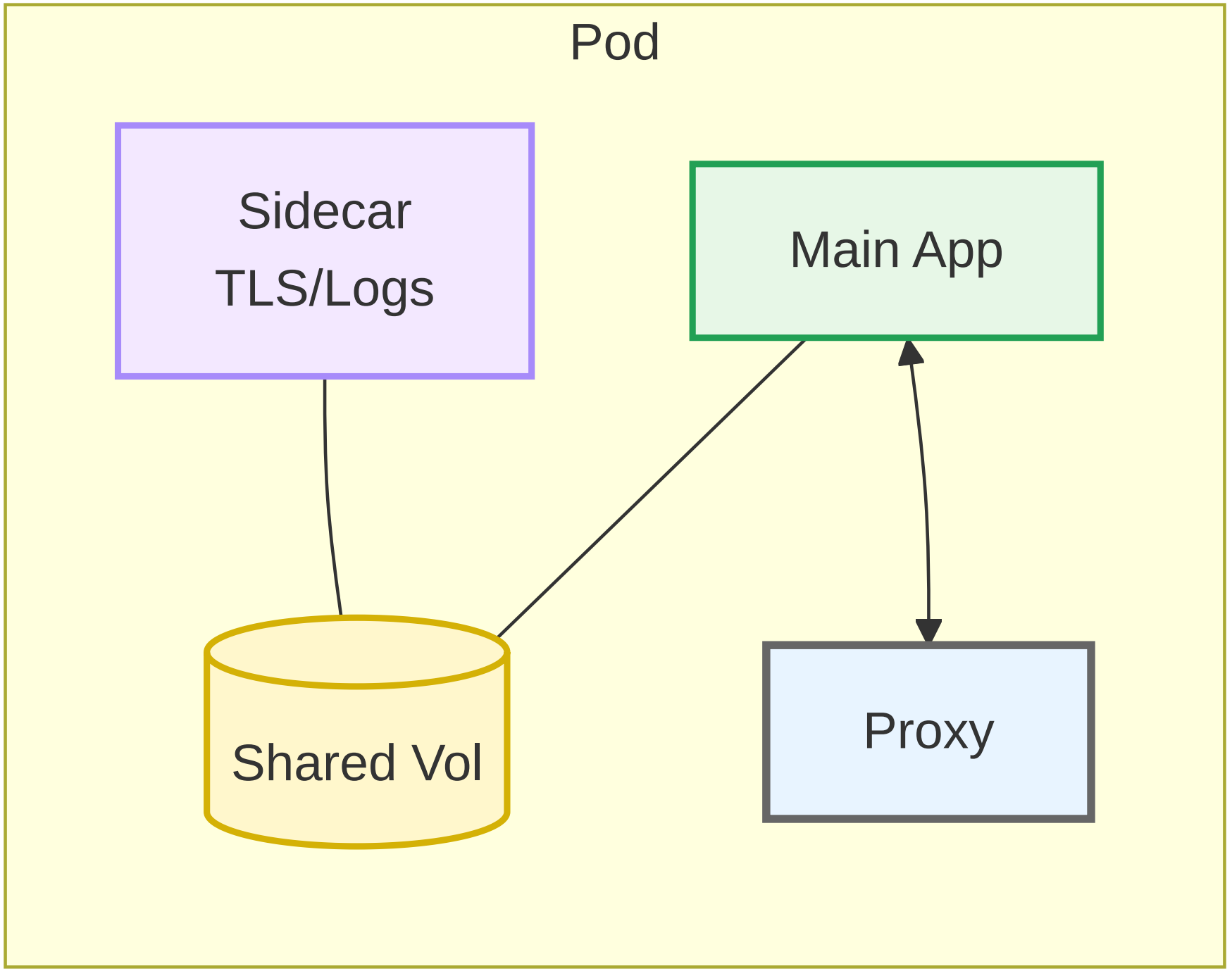
- Phases: Pending → Running → Succeeded/Failed → (Terminating/Gone)
- Restart policies: Always (for long-running), OnFailure, Never (batch)
- Probes: Liveness (restart), Readiness (traffic gating), Startup (init ordering)
- Init containers: Gate main containers via preconditions


Pending
scheduled + containers started
Running
all containers exit 0 (batch)
container exit non-zero
deletion or failure
Succeeded
Failed
Terminating

# 2 Multi-Container Pods

## When & Why

- Tight coupling: Share lifecycle & resources; communicate via localhost/volume
- Sidecar pattern: Cross-cutting concerns (logging agents, TLS proxies)
- Ambassador pattern: Local proxy to remote service/endpoints
- Adapter pattern: Transform metrics/log formats for external systems


Pod
Sidecar
TLS/Logs
Main App
Shared Vol
Proxy

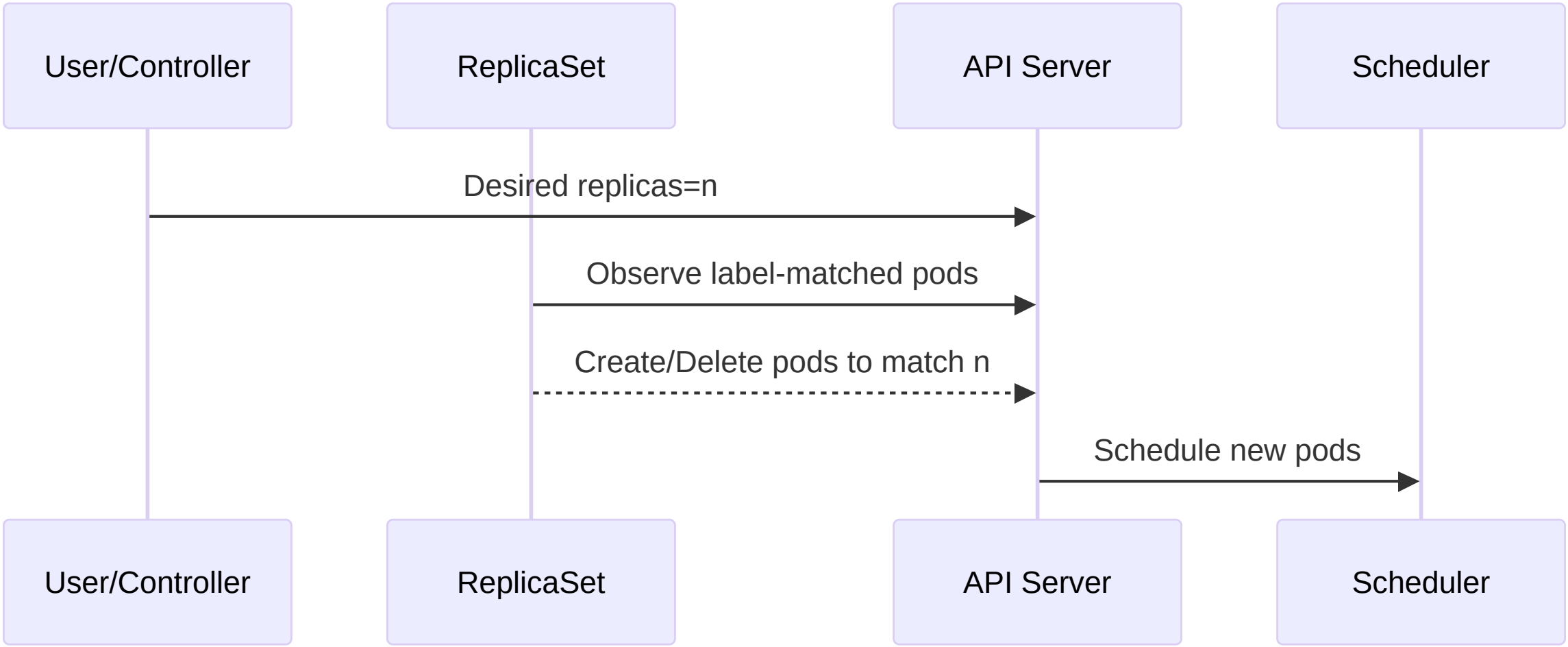
# Init, Sidecar, and Ephemeral Containers

- Init containers: Sequential setup (migrations, bootstrap files)
- Sidecars: Long-lived helpers; consider sidecar lifecycle & termination
- Ephemeral containers: Debug-only; no ports/volumes by default; injected at runtime

# 3 ReplicaSets

## Purpose & Semantics

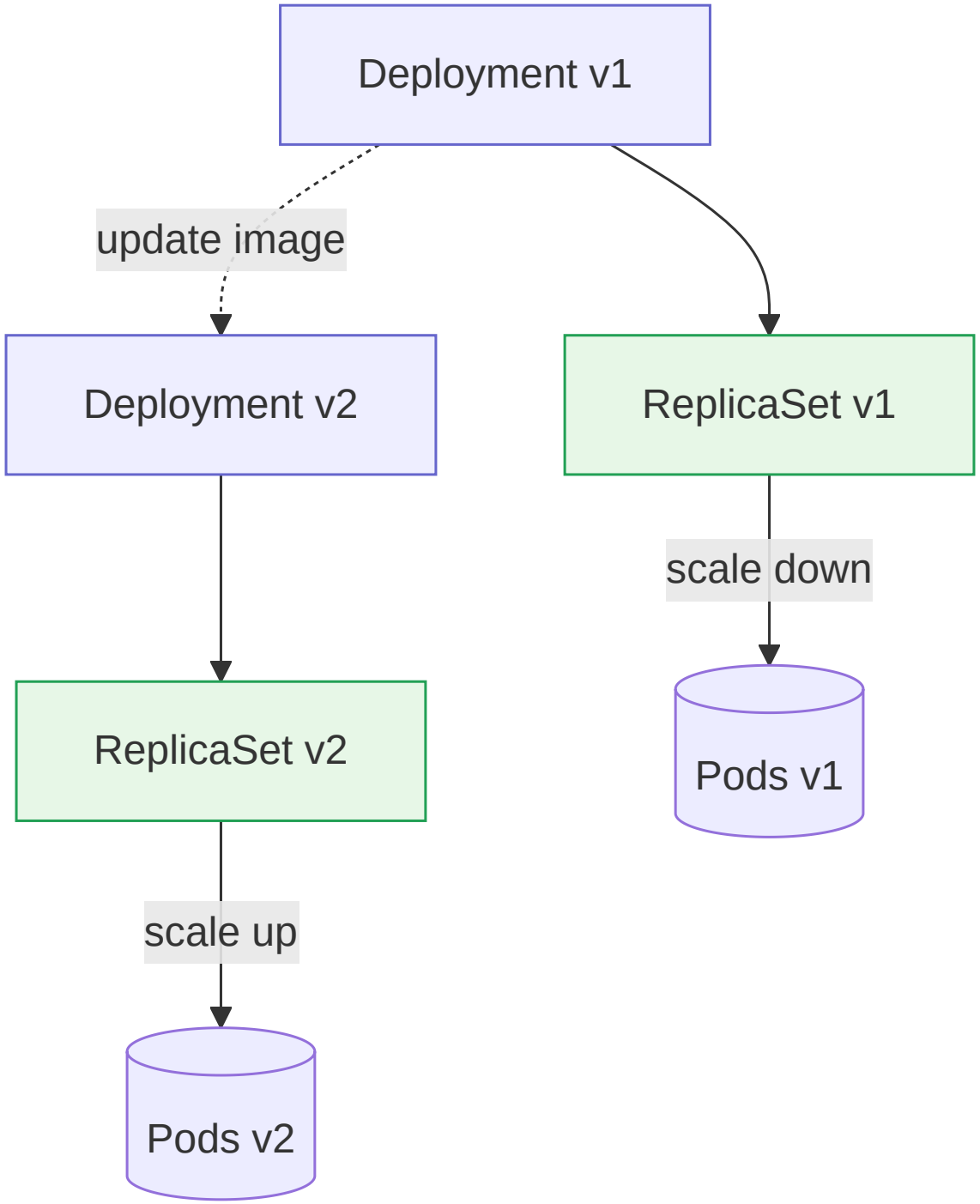
- Goal: Maintain a stable set of identical pod replicas
- Selector: Label selector determines pod membership
- Reconciliation loop: Creates/Deletes pods to match desired replicas
- Not used directly for app rollout; owned by Deployments


User/Controller
ReplicaSet
API Server
Scheduler
Desired replicas=n
Observe label-matched pods
Create/Delete pods to match n
Schedule new pods
User/Controller
ReplicaSet
API Server
Scheduler

# 4 Deployments

## Rolling Updates, Strategy, Rollbacks

- Abstraction over ReplicaSets: Manages versions via new RS per revision
- Strategies:
  - RollingUpdate: Gradual replace; maxUnavailable / maxSurge
  - Recreate: Full stop, then start (for non-coexistable workloads)
- Rollbacks: Keep RS history; revisionHistoryLimit controls depth
- Progress: progressDeadlineSeconds to detect stalled rollouts


Deployment v1
update image
Deployment v2
ReplicaSet v1
scale down
ReplicaSet v2
Pods v1
scale up
Pods v2

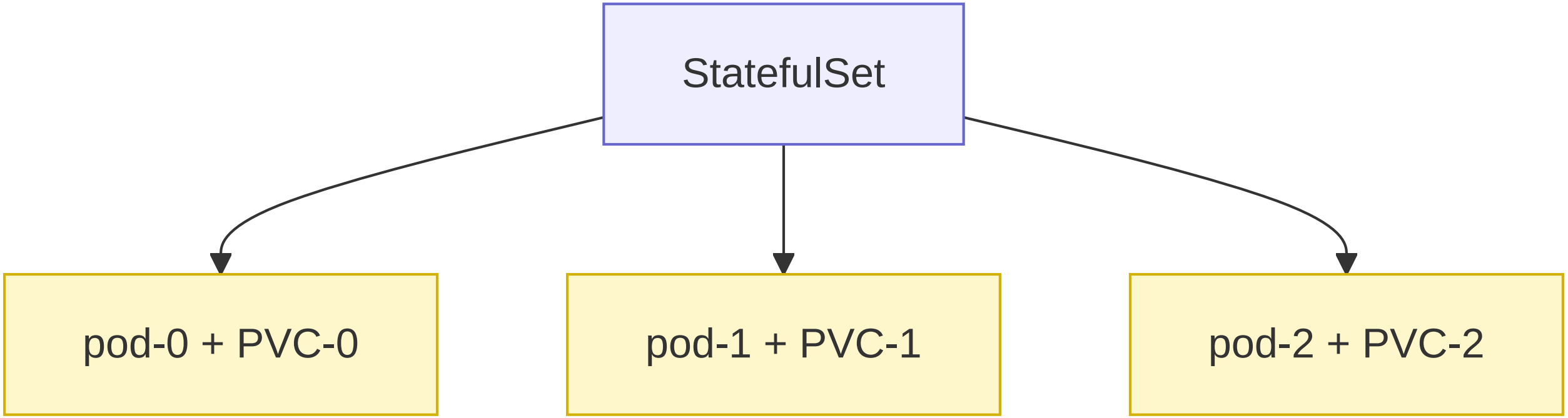
# Readiness-Driven Traffic Shifts

- Readiness gates service endpoints: Only Ready pods receive traffic
- PodDisruptionBudget (PDB): Protect minimum availability during voluntary disruptions
- Surge vs. Unavailable: Balance capacity vs. speed

# 5 StatefulSets

## Stateful Workloads & Identity

- Stable network identity: Ordered pod names (-0, -1…), stable DNS
- Stable storage: One PVC per pod via volumeClaimTemplates
- Ordered semantics: OnDelete or RollingUpdate with ordinal guarantees
- Use cases: Databases, quorum systems, brokers needing sticky IDs


StatefulSet
pod-0 + PVC-0
pod-1 + PVC-1
pod-2 + PVC-2

- Headless Service: Governs DNS (pod-x.service.namespace.svc)
- Caveats: Scale-down does not delete PVCs by default (data safety)

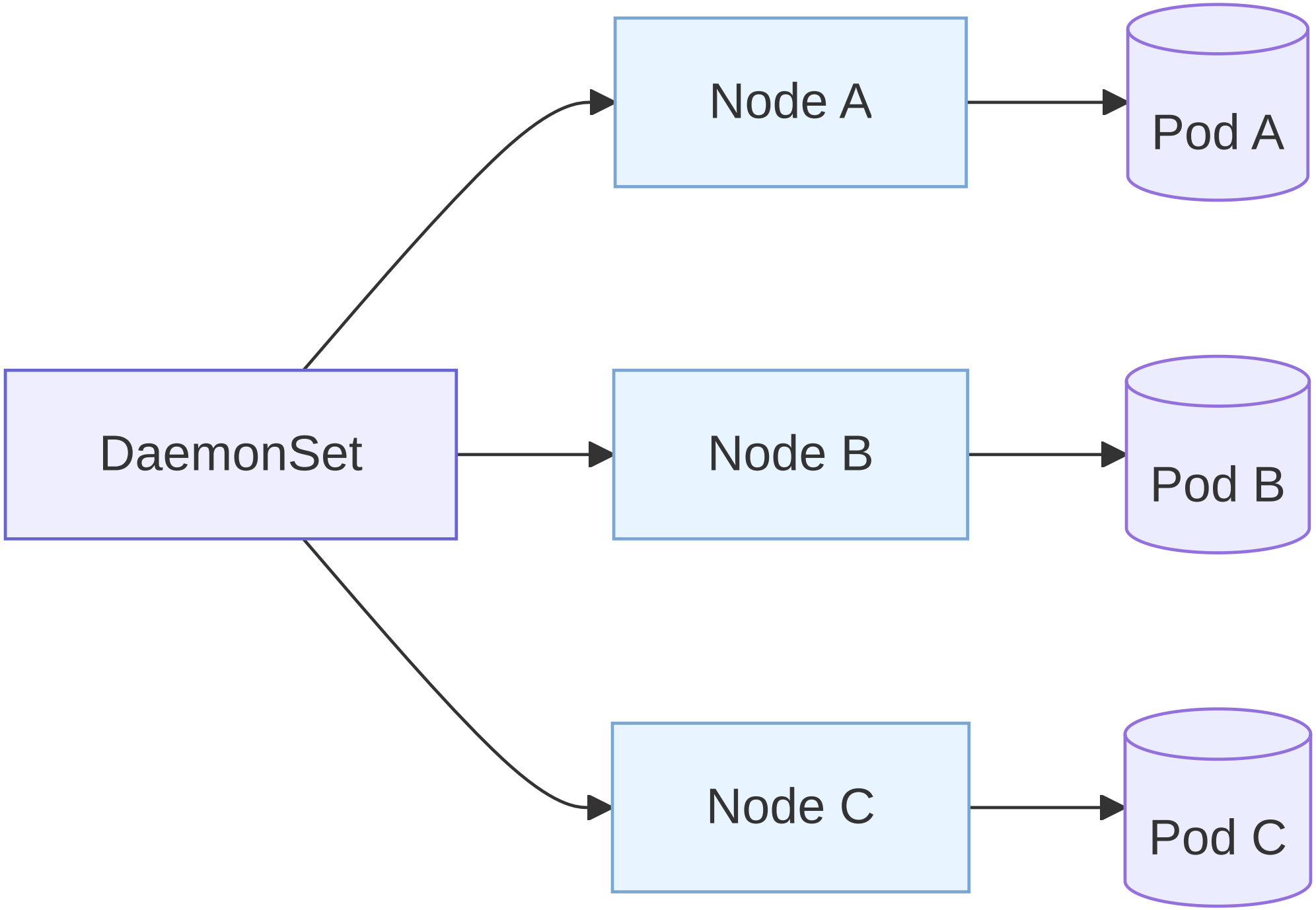
# Update Strategies & Partitioning

- RollingUpdate with partition: Gate rollouts by ordinal (partial updates)
- PodManagementPolicy: OrderedReady (default) vs. Parallel
- Quorum safety: Coordinate updates with app-level readiness & budgets

# 6 DaemonSets

## Node-Scoped Workloads

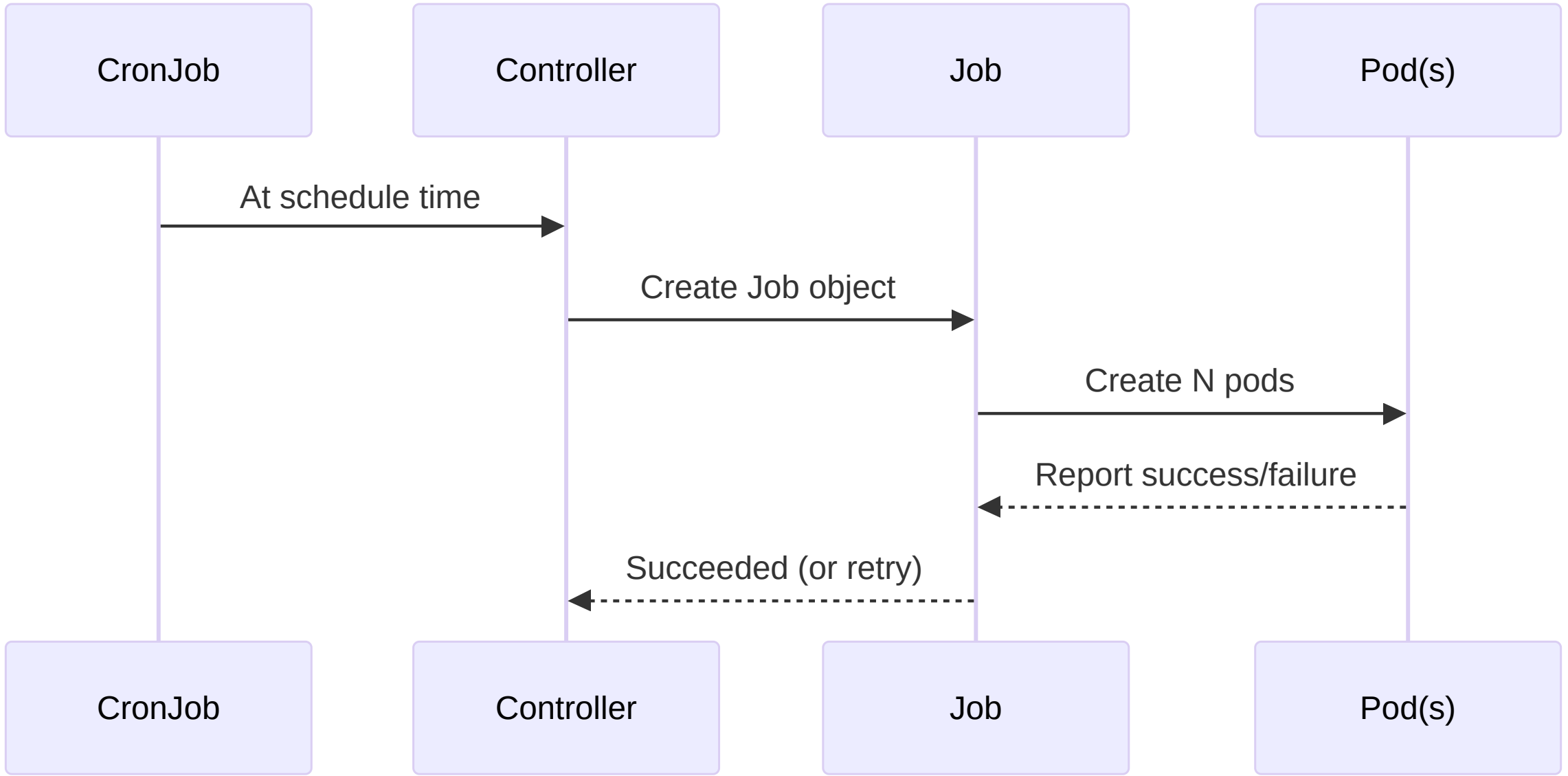
- Guarantee: Run one (or N) pod per eligible node
- Use cases: Node agents (logging, metrics), CNIs, storage daemons, security agents
- Node selection: Labels, taints/tolerations, nodeSelectors/affinity
- Updates: Rolling updates; surge via maxUnavailable for DaemonSets


DaemonSet
Node A
Node B
Node C
Pod A
Pod B
Pod C

# 7 Jobs & CronJobs

## Batch & Scheduling Semantics

- Jobs: Ensure a task completes successfully (pod retries, backoffLimit)
- Completions/Parallelism: Configure map/reduce-style parallel work
- CronJobs: Time-based creation of Jobs via cron format; concurrencyPolicy controls overlap
- Durability: Missed runs handling via startingDeadlineSeconds


CronJob
Controller
Job
Pod(s)
At schedule time
Create Job object
Create N pods
Report success/failure
Succeeded (or retry)
CronJob
Controller
Job
Pod(s)

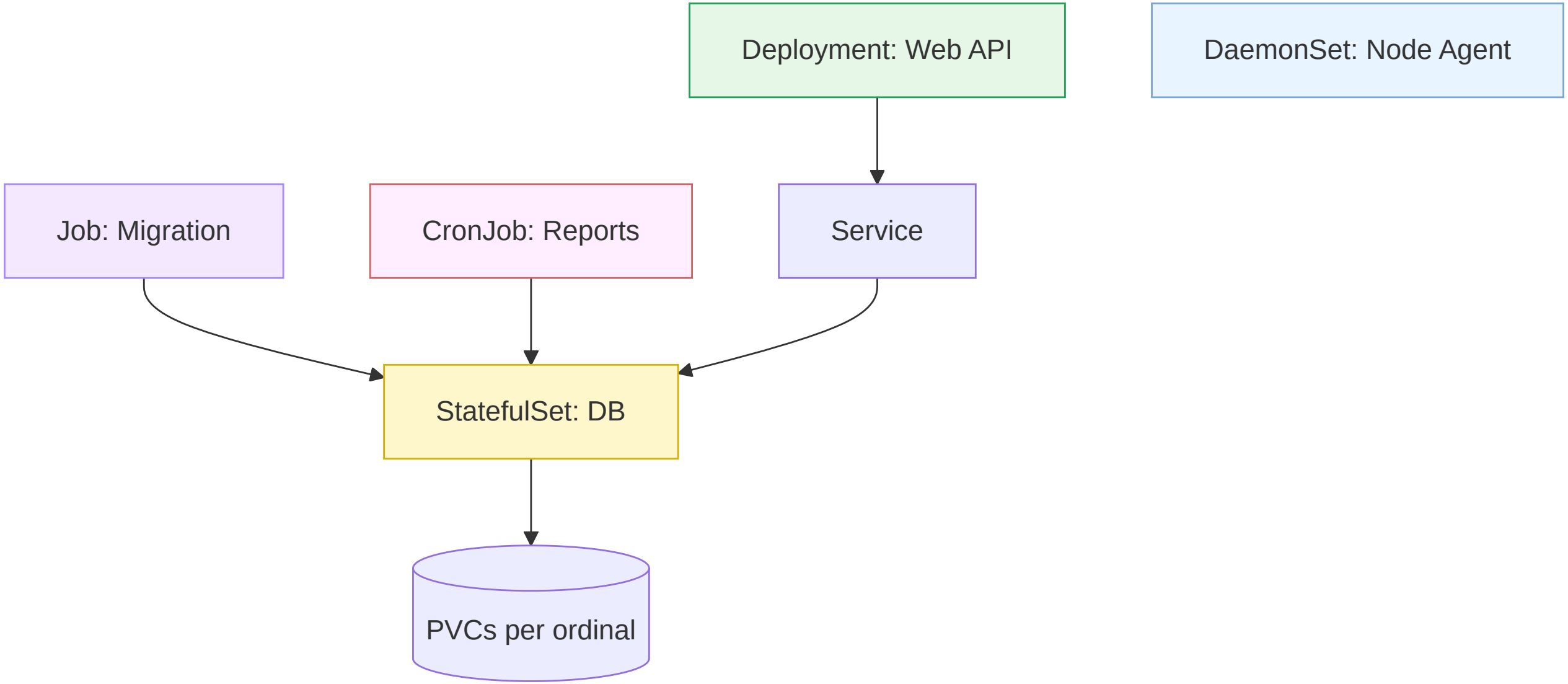
| Concern | Pod | ReplicaSet | Deployment | StatefulSet | DaemonS |
|---|---|---|---|---|---|
| Identity & IP | Single IP, ephemeral | n/a | n/a | **Stable ordinal** | Per node |
| Availability | None | Keep N pods | Versioned rollout | Ordered availability | Per node presence |
| Storage | Volumes in spec | n/a | n/a | **PVC per ordinal** | Node-local allowed |
| Update | Replace pod | Scale only | **Strategy & rollback** | Ordered/partitioned | Rolling by node |

# Scheduling & Disruption Fundamentals

- Affinity/Anti-Affinity & Topology Spread: Control placement and distribution
- PDBs (Deployments/StatefulSets): Protect minimum replicas during drain/upgrade
- PriorityClasses: Influence preemption under pressure
- TTL Controllers: Clean up finished Jobs (TTLAfterFinished)

# Design Patterns & Anti-Patterns

- Use Deployments for stateless services; StatefulSets for sticky identity/storage
- Use DaemonSets for node agents; avoid coupling agents as sidecars to every app
- Use Jobs/CronJobs for finite tasks; avoid long-running jobs for services
- Multi-container pods only for tight coupling; otherwise separate Deployments
- Avoid manual pod creation for apps; avoid rewriting ReplicaSets directly


Deployment: Web API
DaemonSet: Node Agent
Job: Migration
CronJob: Reports
Service
StatefulSet: DB
PVCs per ordinal

# Key Takeaways

- Pod = unit of execution; controllers = lifecycle & semantics
- Choose controllers by identity needs (stateless vs. stateful), availability, and execution model (long-running vs. batch)
- Rolling strategies, readiness gating, and PDBs define safe upgrades
- StatefulSets provide stable identity + storage; PVCs persist beyond pods
- DaemonSets guarantee per-node presence; Jobs/CronJobs guarantee completion/scheduling

# References & Further Reading

- Kubernetes Docs: Pods, ReplicaSets, Deployments, StatefulSets, DaemonSets, Jobs, CronJobs
- Probes & Pod Lifecycle, PDBs & Topology Spread Constraints
- Workload Patterns: Sidecar/Ambassador/Adapter