Kubernetes Configuration & Storage

ConfigMaps • Secrets • Env & Volumes • PV/PVC • StorageClass • CSI

- Audience: Advanced / Intermediate (beginner-accessible)
- Focus: Theory, patterns, architecture & trade-offs

Agenda

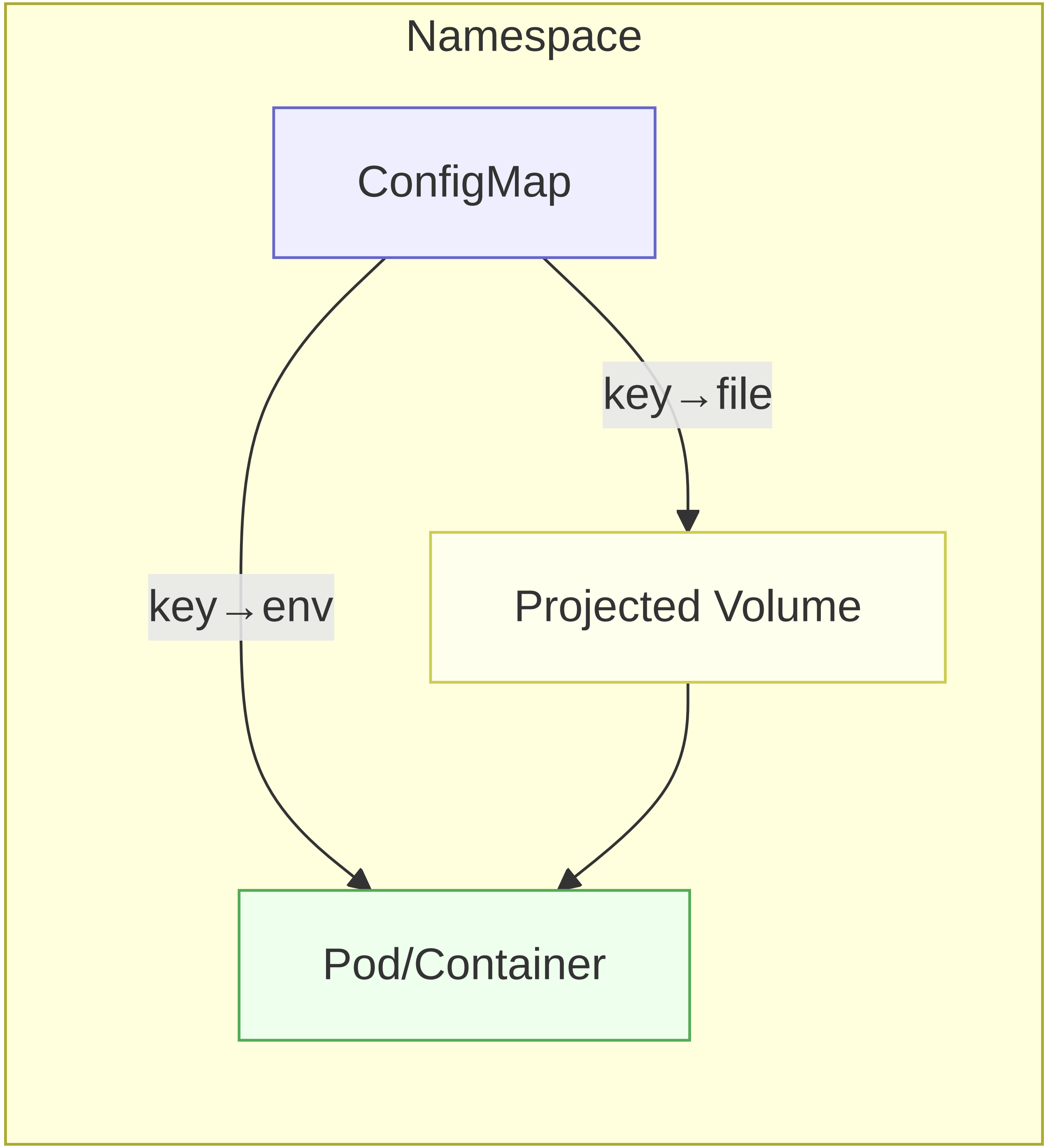
1. ConfigMaps (externalizing configuration)
2. Secrets (storing sensitive data)
3. Environment Variables & Volume Mounts
4. Persistent Volumes (PV) & Persistent Volume Claims (PVC)
5. StorageClasses & Dynamic Provisioning
6. CSI (Container Storage Interface)

1 ConfigMaps

Purpose & Principles

- Externalize configuration: Decouple config from container images
- Scoping: Namespaced objects; consumed by pods in same namespace
- Data model: Key–value pairs (opaque strings), binary via ConfigMap data/base64 not needed
- Update model: Pod restart usually required to pick up changes (unless watched by app or projected with refresh semantics)

> Keep images immutable; vary behavior via ConfigMaps.

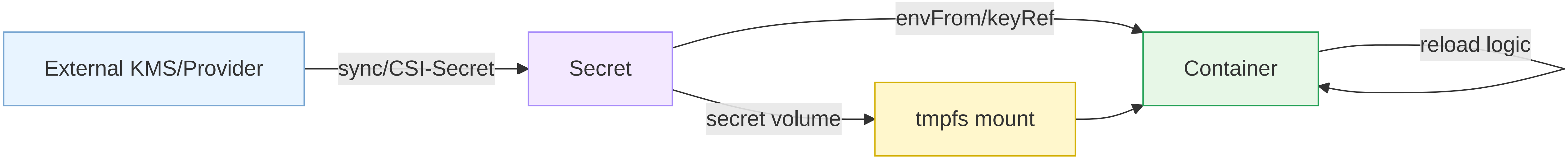

Namespace
ConfigMap
key→file
key→env
Projected Volume
Pod/Container

- Environment variables: Simple mapping for small configs
- Volume projection: Files under a mount path (e.g., /etc/config)
- Checksum strategy: Annotate Deployments with a hash of ConfigMap to trigger rollouts

2 Secrets

Purpose & Risk Model

- Sensitive data: Passwords, tokens, keys, certificates
- Storage: Base64-encoded in API; at-rest encryption requires EncryptionConfiguration on the API server; consider external KMS
- Access control: Namespaced; guarded by RBAC and admission policies
- Node security: Kubelet delivers secrets to nodes running the pods; protect node filesystem & memory
- Base64 ≠ encryption. Treat cluster storage and backups as sensitive.


External KMS/Provider
sync/CSI-Secret
Secret
envFrom/keyRef
secret volume
tmpfs mount
Container
reload logic

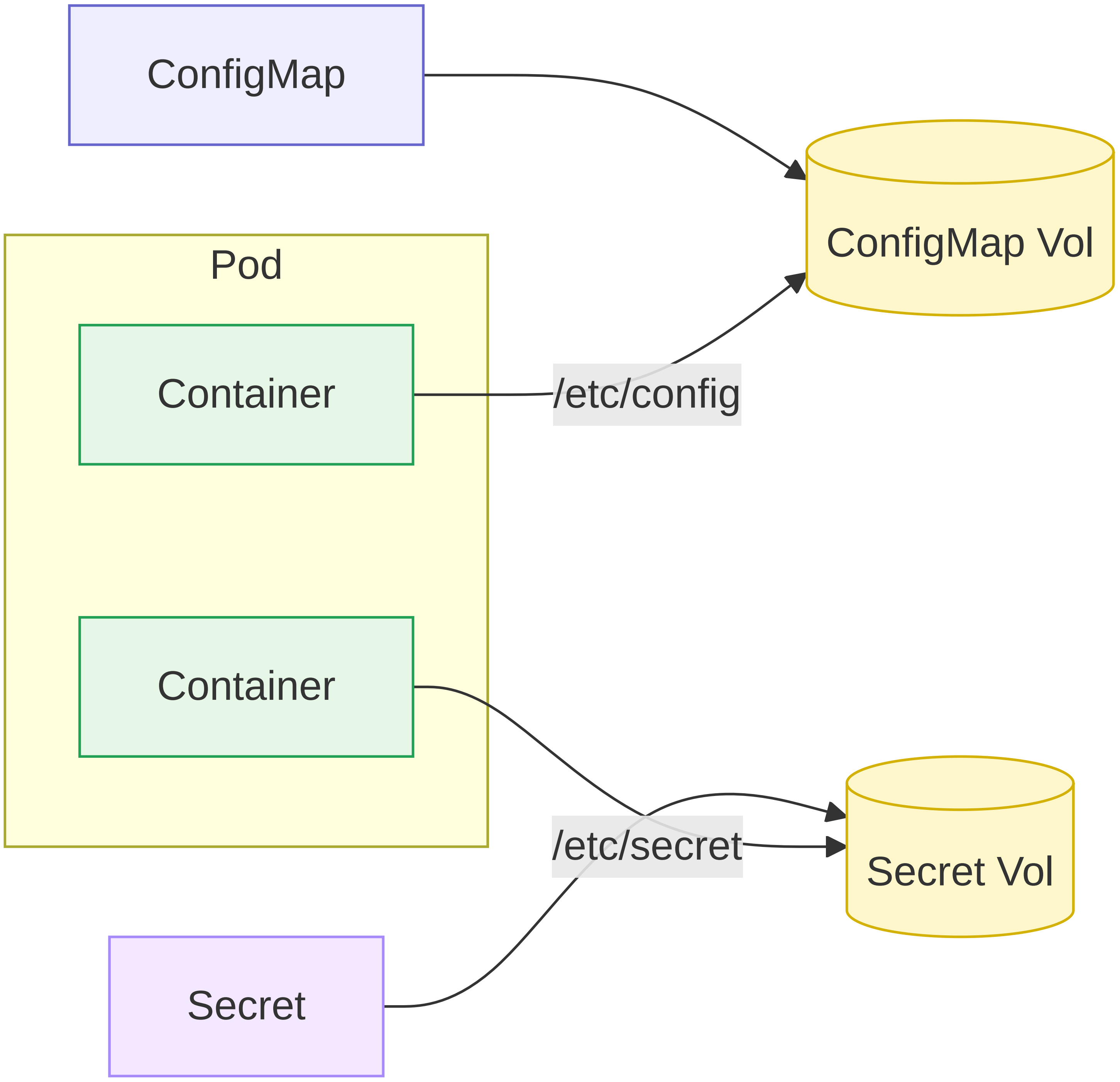
- Mount vs. env: Mount as files for larger material (certs/keys); env for small secrets
- Rotation: Prefer short-lived credentials (OIDC tokens, external secrets operators)
- Auditing: Track access via API audit logs and admission policies

3 Environment Variables & Volume Mounts

Configuration Injection Matrix

Method	Use Case	Pros	Cons
Env (ConfigMap/Secret refs)	Small scalar values	Simple, fast startup	Not dynamic; exposed in process env
Projected Volume (ConfigMap/Secret)	Structured config files, certs	File semantics, watchable by apps	Requires file IO, app logic
Downward API	Pod metadata (labels, resources)		

Method	Use Case	Pros	Cons
Self-awareness	Not for secrets		
InitContainers	Generate config at startup	Complex transforms	Startup latency
Sidecars/Agents	Live reload, templating	Dynamic updates	Operational overhead

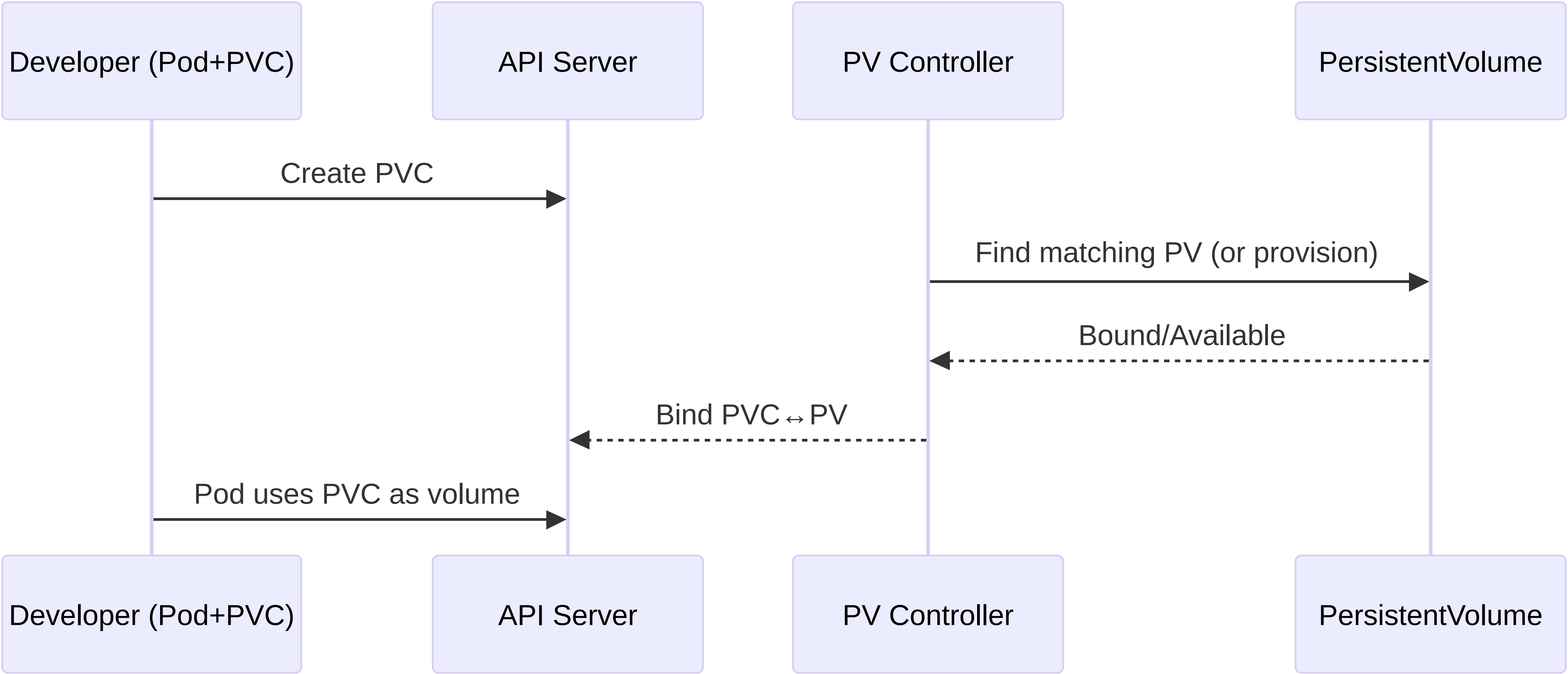

ConfigMap
Pod
Container
Container
Secret
ConfigMap Vol
Secret Vol
/etc/config
/etc/secret

- Default file permissions: Secrets often 0400; adjust via defaultMode cautiously
- Update behavior: Projected volumes update atomically (symlink switch); app must re-read

4 Persistent Volumes & Claims

PV/PVC Binding Model

- PersistentVolume (PV): Cluster-scoped storage resource (capacity, access modes, reclaim policy, storage class)
- PersistentVolumeClaim (PVC): Namespaced request for storage (size, access mode, class)
- Binding: Control loop matches PVC↔️PV based on class & constraints


Developer (Pod+PVC)
API Server
PV Controller
PersistentVolume
Create PVC
Find matching PV (or provision)
Bound/Available
Bind PVC↔PV
Pod uses PVC as volume
Developer (Pod+PVC)
API Server
PV Controller
PersistentVolume

- ReclaimPolicy: Retain, Delete, Recycle (deprecated)
- AccessModes: RWO, ROX, RWX (provider-dependent)

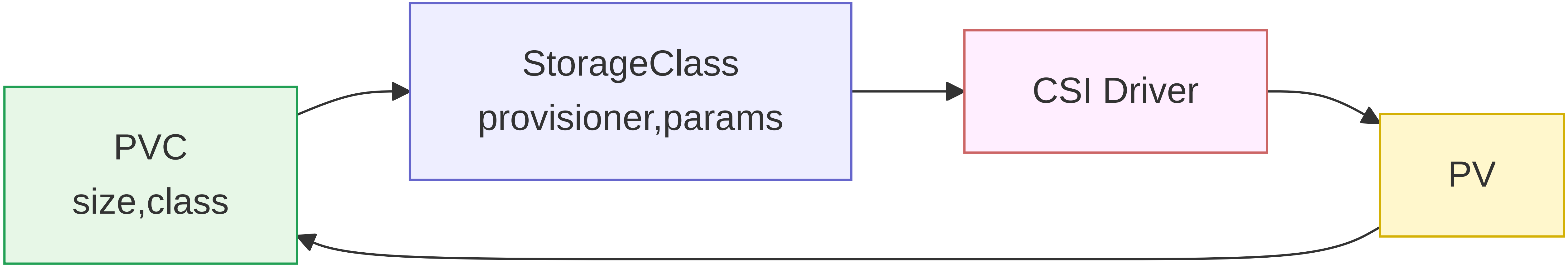
Volume Modes & Topology

- VolumeMode: Filesystem (default) vs. Block (raw device)
- Topology: Zonal/Regional constraints; use WaitForFirstConsumer for correct placement with dynamic provisioning
- Snapshots & Clone: CRDs allow point-in-time copies and PVC-to-PVC clones (same class constraints)

5 StorageClasses & Dynamic Provisioning

Why StorageClasses?

- Abstraction: Describe classes of storage (performance, redundancy, encryption)
- Dynamic provisioning: Automatically create PVs when PVCs request them
- Parameters: Driver-specific (type, IOPS, filesystem, encryption keys)


PVC
size,class
StorageClass
provisioner,params
CSI Driver
PV

- Binding modes: Immediate vs. WaitForFirstConsumer
- AllowVolumeExpansion: Enable online/offline resize (driver-dependent)
- Default class: One StorageClass can be annotated as default

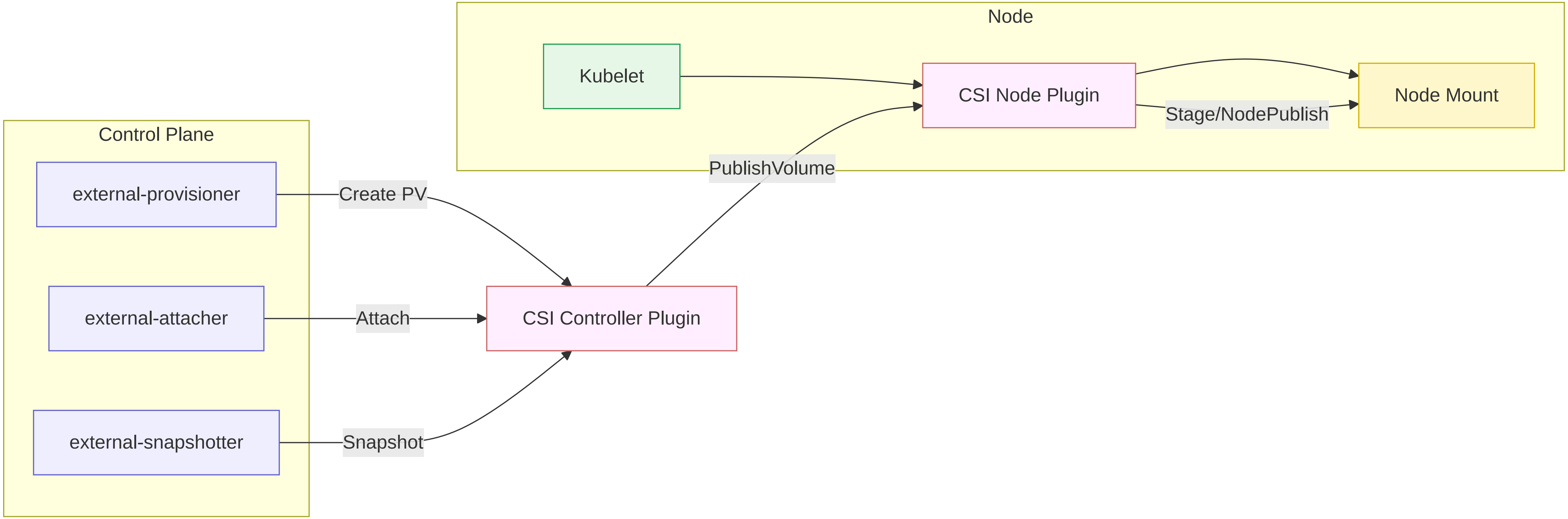
Policy & Governance

- Quotas: Limit number/size of PVCs per namespace
- Class exposure: Offer curated classes (gold/silver/bronze) via RBAC/admission
- Encryption-at-rest: Prefer provider-managed keys; define rotation processes

6 CSI (Container Storage Interface)

Architecture & Components

- Goal: Standardize how K8s talks to storage systems (block & file)
- Drivers: Out-of-tree components implementing CSI spec (controller & node plug-ins)
- Objects: CSIDriver, CSINode, external provisioner/attacher/snapshotter sidecars


Node
Kubelet
CSI Node Plugin
Node Mount
Stage/NodePublish
Control Plane
external-provisioner
external-attacher
external-snapshotter
PublishVolume
Create PV
Attach
Snapshot
CSI Controller Plugin

- Lifecycle verbs: CreateVolume, ControllerPublish/Unpublish, NodeStage/Publish/Unpublish, DeleteVolume
- Snapshots: VolumeSnapshotClass, VolumeSnapshot, VolumeSnapshotContent

CSI Capabilities & Considerations

- Access modes & multi-attach: Dependent on driver/backend (e.g., RWX via NFS/SMB, block devices typically RWO)
- Topology & zoning: Drivers advertise topology keys; scheduler aligns pod placement
- Performance: IOPS/throughput/latency vary by class; monitor with storage metrics
- Security: Node publish points, mount options, fsPermissions; secret references for backend auth

Patterns & Anti-Patterns (Summary)

Do:

- Keep application images stateless; move state to PVCs via CSI
- Use ConfigMaps for non-sensitive config; Secrets for sensitive data with at-rest encryption
- Prefer dynamic provisioning via curated StorageClasses; enforce quotas and policies
- Adopt hash-annotated rollouts for config changes; plan secret rotation

Avoid:

- Embedding config/secrets in images or manifests
- Using env vars for large or long-lived secrets
- Relying on default StorageClass without understanding parameters/topology
- Treating namespaces as storage isolation guarantees without quotas/policies

References & Further Reading

- Kubernetes Docs: ConfigMaps, Secrets, Volumes, Persistent Volumes/Claims, StorageClasses
- CSI Spec & Kubernetes CSI Sidecars
- Security: EncryptionConfiguration, Secret management best practices
- Patterns: Projected volumes, Downward API, VolumeSnapshots