Kubernetes Platforms & Access

Local Runtimes • Managed Services • Install • Access • Isolation

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

Agenda

1. Local Kubernetes (Minikube, kind, k3s, Docker Desktop)
2. Managed Kubernetes (EKS, GKE, AKS, OpenShift)
3. Installing a Cluster (kubeadm, cloud provider blueprints)
4. Accessing Kubernetes (kubectl, API model)
5. kubeconfig & Context Switching
6. Namespaces & Resource Isolation

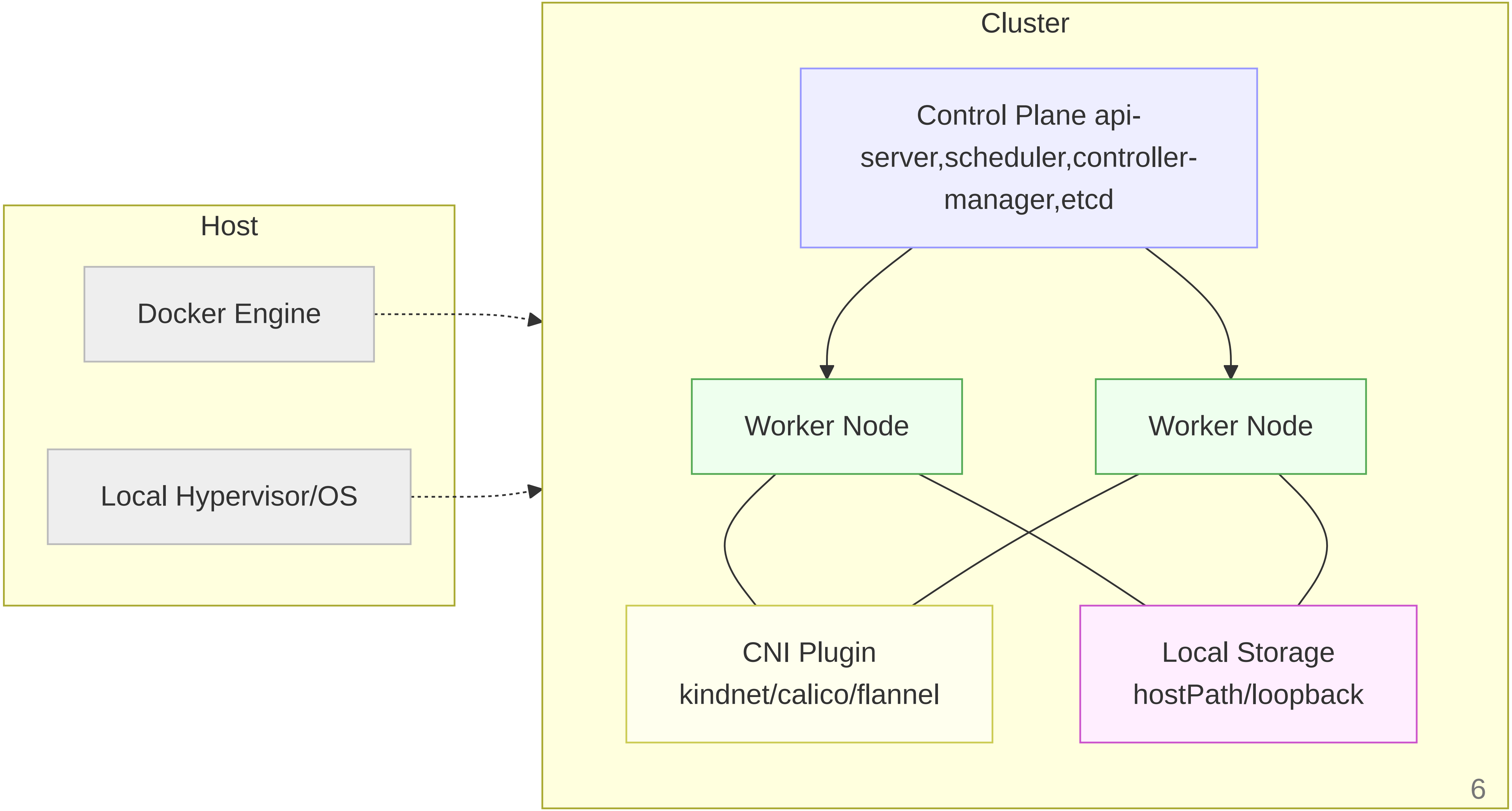
1 Local Kubernetes

Purpose & Design Goals

- Primary goals: Fast feedback, offline dev, reproducible test envs, PoCs
- Non-goals: Full parity with cloud networking, multi-AZ HA, managed control planes
- Common patterns: Single-node or few-node clusters, simplified CNI/CSI, host-embedded runtimes

Runtime	Architecture	Strengths	Caveats	Typical Use
Minikube	Single-node VM/driver; supports multiple drivers (Docker, HyperKit, KVM, etc.)	Feature-rich addons; near-vanilla K8s	VM/driver variability; slower start vs. container-only	Individual dev, demos
kind	K8s “in Docker” (nodes as containers)	Very fast, CI-friendly, multi-node topologies	Limited to Docker networking semantics	CI pipelines, integration tests

Runtime	Architecture	Strengths	Caveats	Typical Use
k3s	Lightweight distro (single binary), etcd-lite (embedded etcd/sqlite)	Low resource footprint, edge/IoT	Subset defaults; differences vs. upstream add-ons	Edge labs, local ops
Docker Desktop	GUI-managed single-node cluster using container runtime	Easiest onboarding on developer laptops	OS licensing/constraints; less configurable	Beginner dev envs


Host
Docker Engine
Local Hypervisor/OS
Cluster
Control Plane api-server,scheduler,controller-manager,etcd
Worker Node
Worker Node
CNI Plugin kindnet/calico/flannel
Local Storage hostPath/loopback

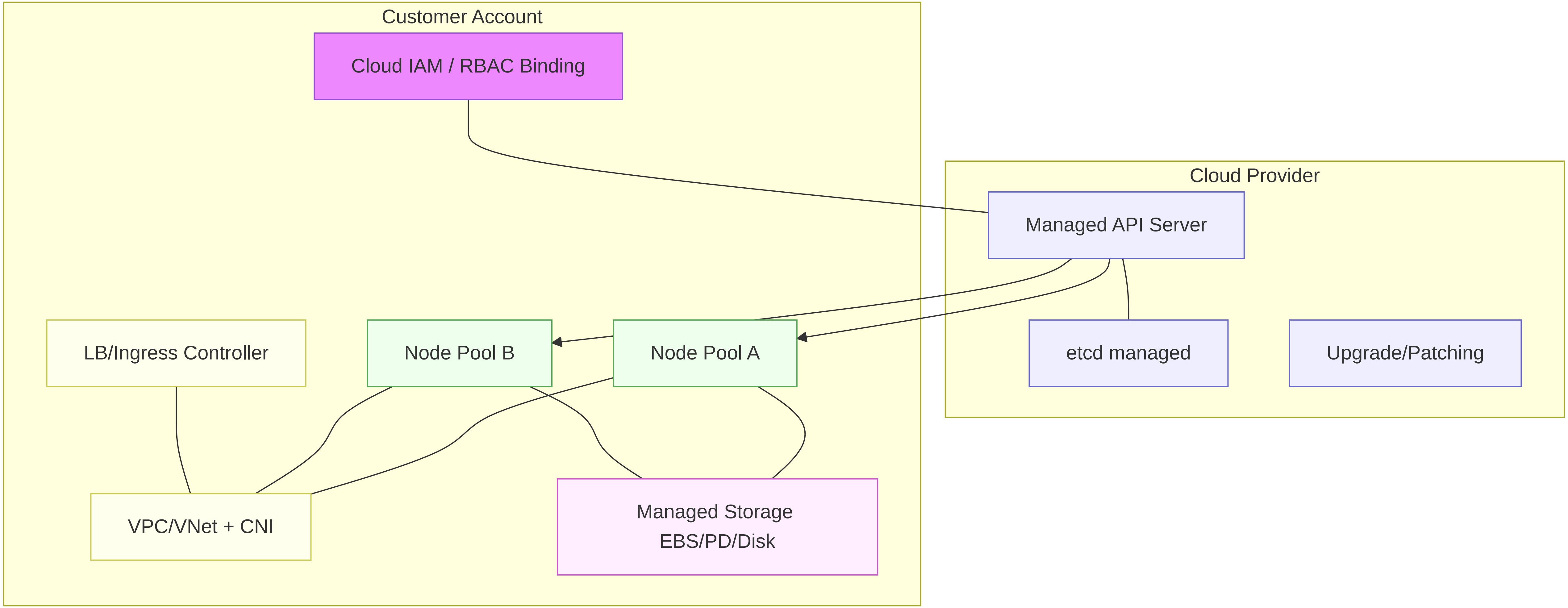
2 Managed Kubernetes Services

Why Managed?

- Managed control plane: SLA-backed API servers, automated patching, upgrades
- Integrated ecosystem: Cloud identity, storage, load balancers, observability
- Compliance & security: Baselines, policy hooks, regionality & IAM

Service	Control Plane	Worker Model	Networking	Value Adds
EKS	AWS-managed	Self-managed nodes or managed node groups / Fargate	VPC CNI (ENI); advanced via CNI addons	IAM integration, ALB/NLB, EKS add-ons
GKE	Google-managed	Standard/Autopilot modes	VPC-native; Dataplane v2 (eBPF)	Autopilot, fleet mgmt, Cloud Ops

Service	Control Plane	Worker Model	Networking	Value Adds
AKS	Azure-managed	VM scale sets, node pools	Azure CNI; kubenet	AAD integration, Gatekeeper add-on
OpenShift (ROSA/ARO/OSD)	Red Hat-managed options	IaaS worker pools	OVN-K/Kuryr	Built-in operators, OpenShift SDN, enterprise registry


Customer Account
Cloud IAM / RBAC Binding
LB/Ingress Controller
Node Pool B
Node Pool A
VPC/VNet + CNI
Managed Storage
EBS/PD/Disk
Cloud Provider
Managed API Server
etcd managed
Upgrade/Patching

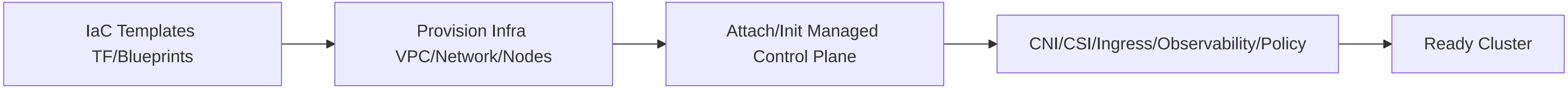
3 Installing a Cluster

kubeadm (Vanilla Upstream)

- Role: Reference installer for upstream Kubernetes on your own infra
- What it does: Bootstraps control plane & workers, certs, tokens, kubelet config
- What it doesn’t: No day-2 ops (upgrades, monitoring, CNI choice) beyond basics
- When to choose: On-prem labs, custom networking/storage, educational purposes

Cloud-Provider Blueprints

- IaaS-first tools: EKS Blueprints/Terraform, GKE Autopilot configs, AKS Landing Zones
- Opinionated stacks: CNI/CSI, Ingress, logging, metrics, policy hooks prewired
- Advantages: Faster time-to-value, supported patterns, easier compliance mapping


IaC Templates
TF/Blueprints
Provision Infra
VPC/Network/Nodes
Attach/Init Managed
Control Plane
CNI/CSI/Ingress/Observability/Policy
Ready Cluster

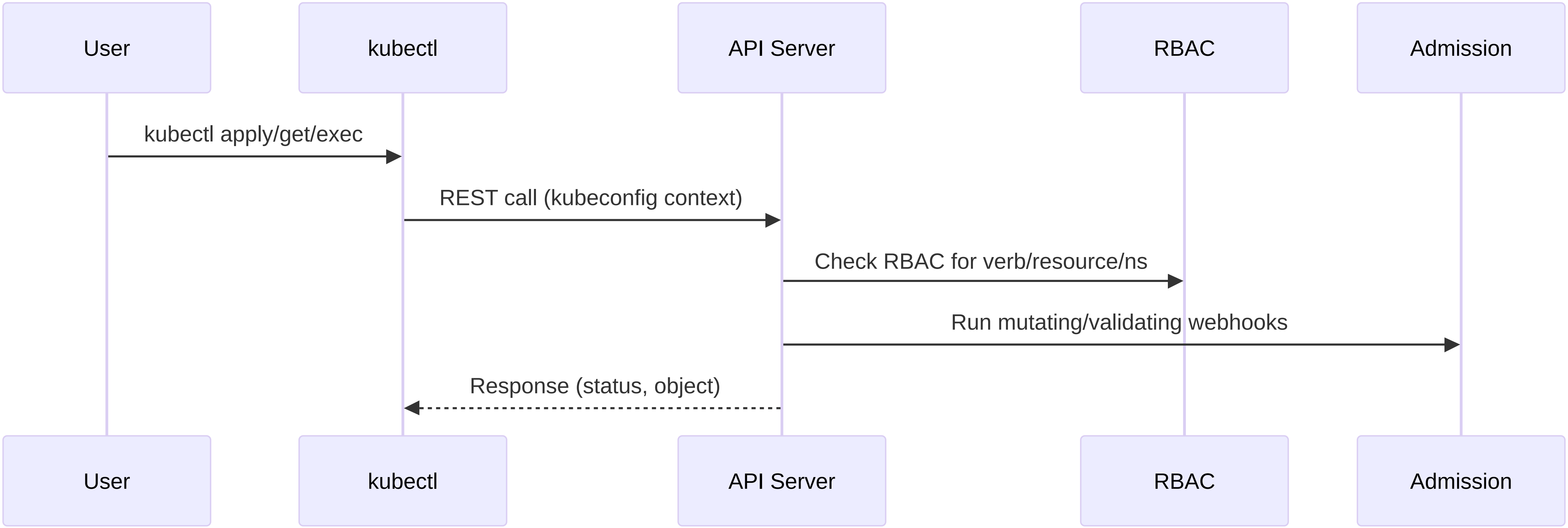
HA Considerations (Theory)

- Control plane: Multi-master, etcd quorum (odd number), separate failure domains
- Workers: Multiple node pools, PodDisruptionBudgets, topology spread constraints
- Data plane: CNI/CSI failure domains, zonal replication, backup/restore strategy

4 Accessing Kubernetes

API-First Model

- Everything via API server: kubectl, controllers/operators, CI/CD
- AuthN: Client certs, bearer tokens, OIDC, cloud-issued short-lived creds
- AuthZ: RBAC (verbs on resources), ABAC rarely, admission webhooks for policy
- Audit: Request/response audited per policy



User

kubectl

API Server

RBAC

Admission

kubectl apply/get/exec

REST call (kubeconfig context)

Check RBAC for verb/resource/ns

Run mutating/validating webhooks

Response (status, object)

User

kubectl

API Server

RBAC

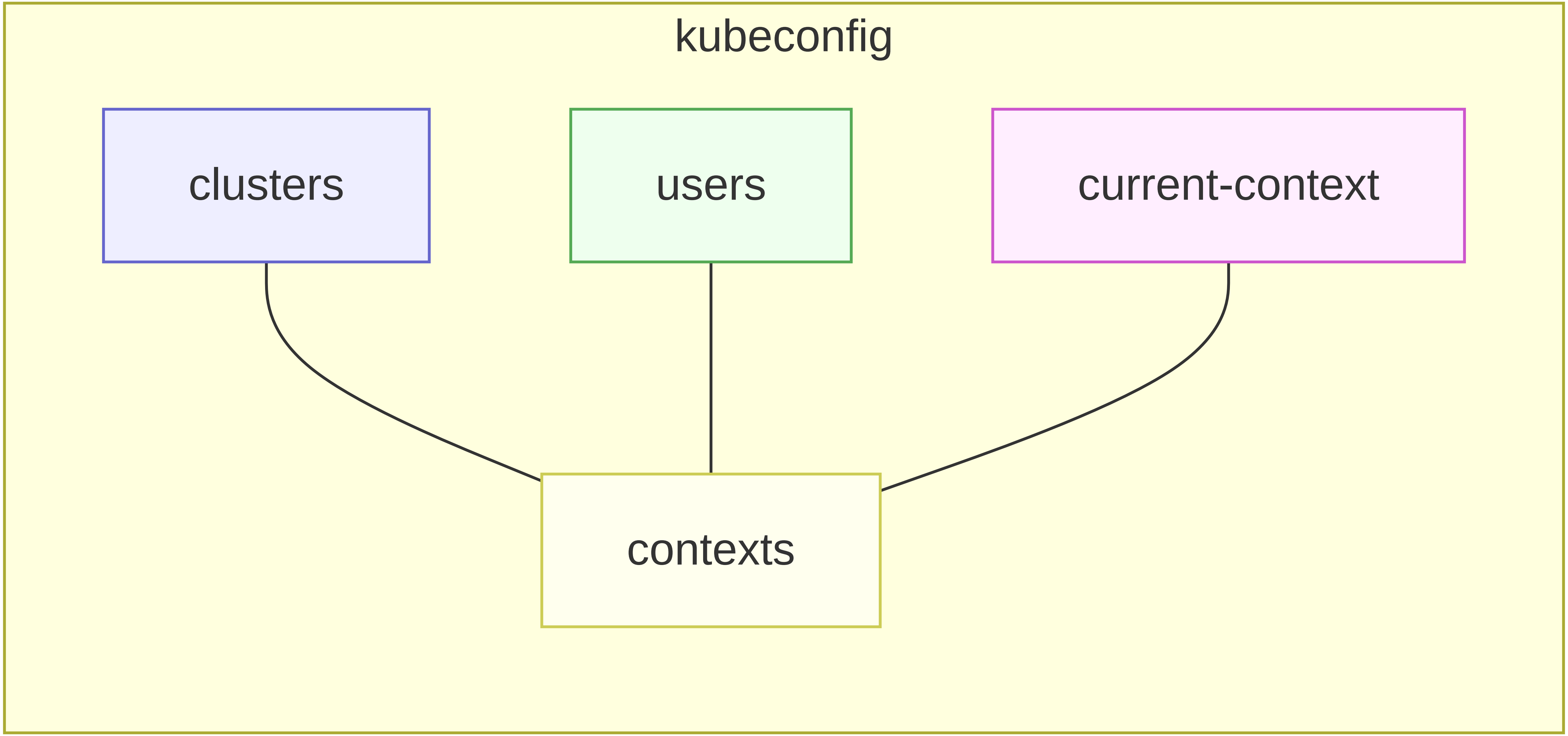
Admission

- Declarative workflows: kubectl apply with server-side apply & field managers
- Diff/preview: kubectl diff, --dry-run=server
- Extensibility: kubectl krew plugins

5 kubeconfig & Context Switching

kubeconfig Structure

- Clusters: API endpoints + CA data
- Users: Credentials (certs, tokens, exec-plugins like cloud auth)
- Contexts: Triples (cluster, user, namespace) + current-context pointer


kubeconfig
clusters
users
current-context
contexts

- Merging: KUBECONFIG can be a list; files merge by key names
- Exec plugins: Obtain short-lived tokens (OIDC/cloud), improve security posture

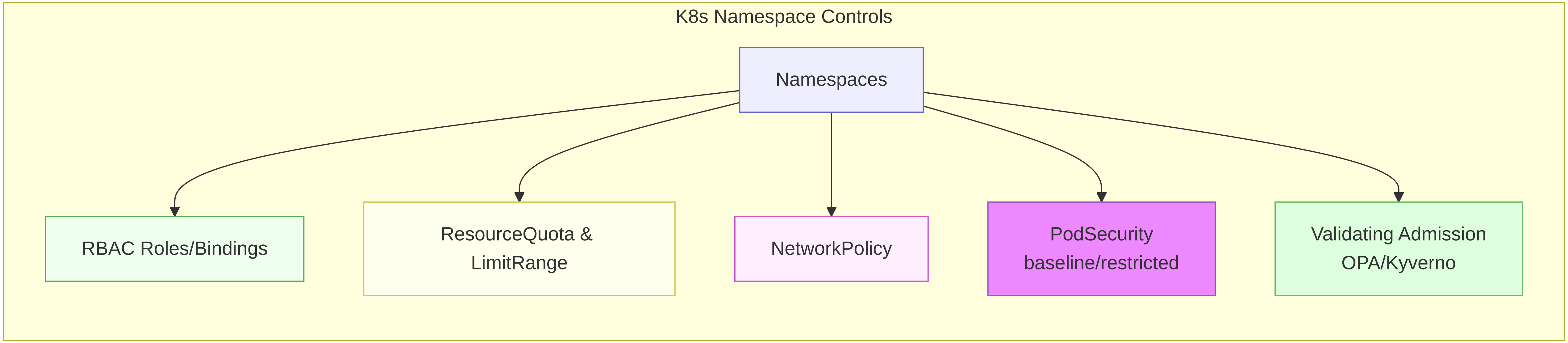
Context Management (Theory & Conventions)

- Naming: <org>-<env>-<region>-<cluster> for cluster; <role>-<idp> for users
- Default namespace: Avoid default; use per-team/project namespaces in context
- Security posture: Store kubeconfig securely; prefer short-lived creds; rotate periodically

6 Namespaces & Resource Isolation

What Namespaces Do / Don’t

- Do: Provide resource scoping, name scoping, RBAC boundaries, NetworkPolicy scopes, quota domains
- Don’t: Hard multi-tenancy by themselves; not a security barrier against kernel/container escapes


K8s Namespace Controls
Namespaces
RBAC Roles/Bindings
ResourceQuota & LimitRange
NetworkPolicy
PodSecurity baseline/restricted
Validating Admission OPA/Kyverno

- Combine: Namespaces + RBAC + Quotas/Limits + NetworkPolicy + PodSecurity + Admission policies
- Goal: Limit blast radius, fair resource sharing, least-privilege execution

Resource Management & Fairness

- LimitRange: Default CPU/memory requests/limits per namespace
- ResourceQuota: Aggregate caps on CPU/memory, object counts (pods, PVCs, LoadBalancers)
- PriorityClasses & PDBs: Control scheduling priority, graceful disruptions

Putting It Together

Reference Environments (Theory)

- Local dev: kind/Minikube, simple CNI, hostPath volumes, fast cycles
- Pre-prod: Managed K8s with IaC blueprints, gated ingress, CSI snapshots, policy hooks
- Prod: Managed control plane, multi-pool nodes, strict NetworkPolicy, centralized IAM/OIDC, admission policies, quotas, audit

Key Design Takeaways

- Choose local runtime by speed vs. fidelity needs
- Prefer managed control planes for SLA, upgrades, integrations
- For installs: kubeadm for learning/custom; cloud blueprints for speed/ops
- Access is API-centric: secure RBAC, short-lived creds, audited
- Treat namespaces as logical scopes, not hard security walls; layer controls

References & Further Reading

- Kubernetes Docs: kubeadm, kubectl, kubeconfig, Namespaces, RBAC
- kind / Minikube / k3s documentation
- EKS / GKE / AKS official guides & blueprints
- OpenShift docs (Operators, SDN)
- CNI/CSI concepts; NetworkPolicy best practices