



CLASTIX kMetal Technical Whitepaper

Three Stacks Into One: A Reference Architecture for Bare-Metal Kubernetes

Index

01. [Executive Summary](#)
02. [Market Context](#)
03. [The Problem](#)
04. [Architecture Overview](#)
05. [The Under Cluster](#)
06. [Hosted Control Planes](#)
07. [Compute](#)
08. [Networking](#)
09. [End to End](#)
10. [The Isolation Model](#)
11. [Failure Domains and High Availability](#)
12. [Operating the Fleet](#)
13. [Performance and Scale Reference](#)
14. [Open-Source Foundation and Commercial Platform](#)
15. [Conclusion and Next Steps](#)
16. [Glossary of Acronyms and Abbreviations](#)

CLASTIX kMetal

Technical Whitepaper

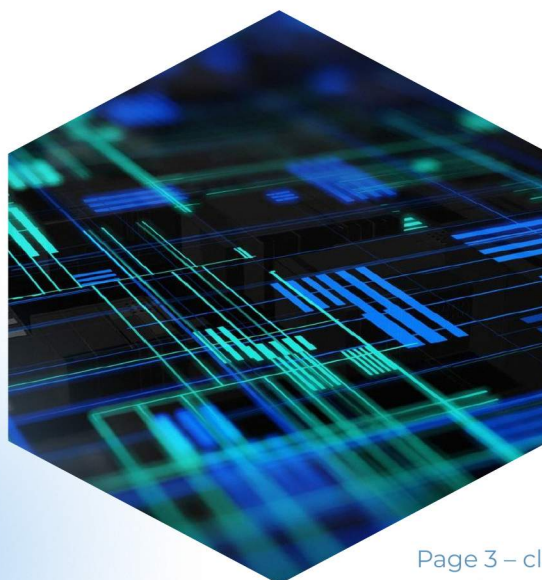
1 Executive Summary

Organizations running Kubernetes on bare metal at scale pay three compounding costs: dedicated control-plane machines for every tenant cluster, a separate hypervisor platform operated solely as an isolation mechanism, and operational sprawl as the fleet grows. At 50 tenant clusters, the conventional pattern means 150 machines running nothing but API servers and etcd, a licensed virtualization stack with its own consoles and skill set, and 50 independent lifecycles managed one at a time.

kMetal collapses these three stacks into a single Kubernetes-native platform. Tenant control planes run as pods on shared infrastructure (Kamaji). Tenant workers run as KVM virtual machines orchestrated by Kubernetes itself (KubeVirt, provisioned through Cluster API). Every tenant receives a private virtual network with its own routing domain (Kube-OVN). Worker operating systems are built as immutable, signed artifacts declared as Kubernetes objects (Kairos). The entire stack – from the OS image to the tenant cluster – is driven through the Kubernetes API and reconciled continuously via GitOps.

This paper describes the architecture in the depth a platform-engineering team needs to evaluate it: the layered component model, the anatomy of the under cluster and the tenant layer, the full provisioning flow of a tenant cluster, the three isolation boundaries (kernel, network, control plane), failure domains and high-availability behaviour, and the performance and scale characteristics of the reference deployment.

Who should read this. Platform architects, infrastructure and SRE leads, and security reviewers evaluating kMetal for enterprise private cloud, managed Kubernetes services, or GPU / AI-factory infrastructure. Familiarity with Kubernetes concepts (CRDs, controllers, CNI) is assumed.



Key architectural claims

Claim	Mechanism	Reference figure
Control-plane cost does not grow with tenant count	Tenant control planes run as pods on 3 shared nodes (Kamaji)	150 → 3 machines at 50 tenants; ~200 mCPU / ~256 MiB idle per tenant
Full tenant cluster in about two minutes	Single declarative workflow: CAPI + KubeVirt + Kube-OVN reconcile in parallel	~2 min to an isolated, production-ready cluster
Hard multi-tenancy without a hypervisor product	Kernel-native KVM per tenant worker + per-tenant OVN routing domain + dedicated etcd	No vCenter, no NSX, no parallel toolchain
A fleet governed like one cluster	Cluster definitions & continuous reconciliation (Kamaji with Cluster API) + Cluster profiles (Sveltos)	100 control planes reconciled in < 150 s
Zero-downtime lifecycle	Blue/green control-plane upgrades; rolling immutable-OS rollouts	Control-plane upgrade in ~16 s and 3 workers upgrade in ~90 s

2 Market Context: Why This Architecture, Why Now

The pressure that makes kMetal timely is not primarily technical – it is commercial and regulatory. IDC's 2026 market data shows European demand moving decisively toward infrastructure that organizations can run, control and account for themselves, under **cost and skills constraints** that the conventional multi-stack pattern does not respect.

Self-operated, private-cloud infrastructure is now mainstream, not a niche. By mid-to late 2026, 59% of organizations in Europe are expected to be active sovereign-cloud users, and committed near- and longer-term demand reaches around 86%. For more than a third of adopters, a sovereign or private cloud is the only cloud they use for the workloads in question. The centre of gravity is shifting from rented hyperscale capacity toward **infrastructure that runs on the organization's own terms** – exactly the deployment posture kMetal targets on bare metal.

Running where it matters. Government and telecommunications lead adoption because critical-infrastructure mandates make self-operated cloud a procurement prerequisite rather than a discretionary choice, and financial services shows the largest cohort planning to adopt, driven by DORA and concern over extraterritorial data requests. **Telcos, MSPs, financial services and public-sector operators** are precisely the segments that run Kubernetes on bare metal at the density where the control-plane tax bites hardest. CLASTIX has also found **AI Factories** to be leading adoption.

Cost is the defining constraint. The market will not absorb a large premium for control: most organizations expect to spend only 6–10% of total public-cloud budget on sovereign solutions, and providers considering ~20% premiums face resistance. A platform for which economics depend on **eliminating dedicated hardware, licensing and operational overhead** – rather than charging a premium for isolation – is aligned with how these budgets actually behave.



Portability and freedom from lock-in are explicit selection criteria. Buyers rank freedom from lock-in and the ability to move workloads back from cloud to on-premises among the attributes they look for in a partner. kMetal's open-source foundation (Kamaji, Apache 2.0) and its use of unmodified upstream components speak directly to that requirement: every tenant cluster is CNCF-conformant, with no proprietary fork to be trapped inside.

A skills shortage rewards operational consolidation. A lack of local knowledge and skills is among the top barriers to adoption, and 34% cite dependency on external suppliers of IT skills as a sovereignty concern in its own right, alongside 31% citing dependency on external suppliers of GPUs, CPUs and servers. Reducing three fragmented toolchains to one Kubernetes-native operating model is a direct response to a market that cannot easily hire its way out of complexity – and the GPU-supply pressure is the same pressure that makes kMetal's fast, repeatable GPU-node provisioning valuable.

kMetal is not positioned here as a compliance or certification product; see the language note in Section 15. The point of this section is narrower and stronger: **the architecture that kMetal already implements – self-operated, hardware-efficient, upstream-conformant, single-model – is the architecture the 2026 market is asking for.**

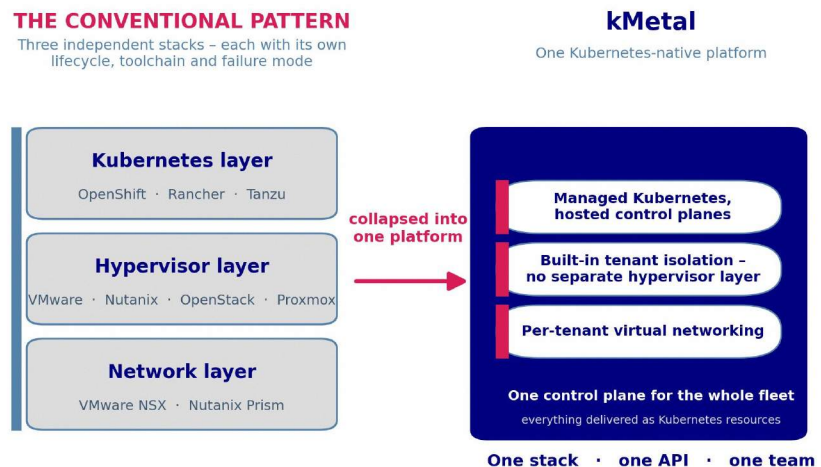
Sources

IDC, "Sovereign Cloud in Europe, 2026 – Market Update" (Rahiel Nasir), June 2026. Figures drawn from IDC's EMEA Multicloud Survey 2025 (September), Europe (N = 1,155) and IDC's Worldwide Digital Sovereignty Survey 2025 (July), Europe (n = 370). Percentages are reproduced as published; cite the underlying survey and sample size wherever these figures appear externally.

3 The Problem: Three Layers Where One Should Be

Multi-tenant Kubernetes on bare metal has traditionally been assembled from three independent layers, each a separate product with its own lifecycle, toolchain and failure mode (Figure 1): a Kubernetes distribution (e.g. OpenShift, Rancher, or Tanzu), a hypervisor platform beneath it (e.g. VMware, Nutanix, OpenStack, or Proxmox), and a network-virtualization layer alongside it (e.g. VMware NSX or Nutanix Prism). kMetal collapses all three into a single Kubernetes-native platform.

Figure 1. The conventional pattern maintains three separate stacks; kMetal collapses them into one Kubernetes-native platform driven by a single API.



3.1 The Kubernetes “control-plane tax”

Every tenant starts with a Kubernetes distribution – OpenShift, Rancher or Tanzu – and every production cluster needs its own highly available control plane: conventionally three dedicated machines for etcd quorum. This control-plane tax scales linearly – at 50 tenants, 150 machines serve zero workloads – and so does the operational load: 50 etcd clusters to back up, 50 PKI hierarchies to rotate, 50 upgrade windows, 50 failure domains.

3.2 The “hypervisor tax”

Containers sharing a kernel are not sufficient isolation for regulated workloads or paying tenants, so the conventional fix is a general-purpose virtualization platform beneath Kubernetes – VMware, Nutanix, OpenStack or Proxmox. That hypervisor tax is a second stack, licensed and operated solely for isolation, with its own compute model, consoles and on-call skill set.

3.3 The “network tax”

Isolating tenants on the wire is a third product – a network-virtualization layer such as VMware NSX or Nutanix Prism, supplying the per-tenant routing and policy a flat network cannot. The network tax is paid in a separate license and a separate operator — the network team — so every change becomes another ticket in another queue.

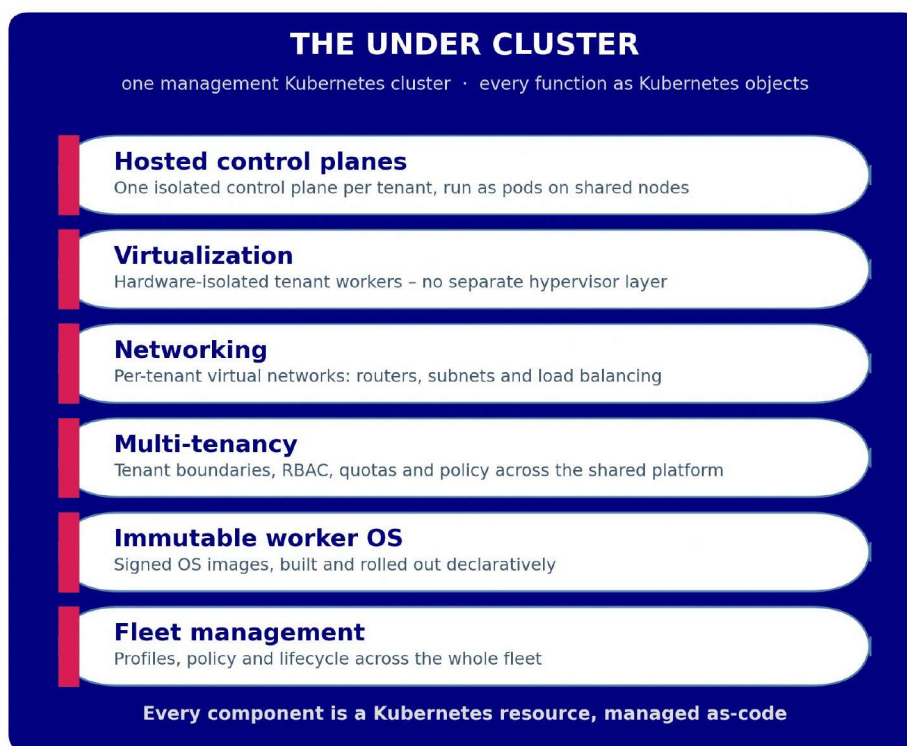
kMetal was designed from the position that none of this is the goal – running workloads with proper isolation is. All three layers, and the fleet sprawl they create as clusters multiply across teams, sites and customers, collapse when Kubernetes itself becomes the operational model for the entire stack.

4 Architecture Overview

kMetal is delivered as a curated, supported bundle of components installed on a management Kubernetes cluster – the under cluster – which hosts every platform function: hosted control planes, virtualization, software-defined networking, OS-image building and fleet reconciliation. Tenant clusters are full, upstream, CNCF-conformant Kubernetes clusters whose control planes live in the under cluster and whose workers run as KVM virtual machines on the same shared bare-metal fleet.

The platform is best understood as multiple components, each exposed as Kubernetes objects and each reconciled by controllers running in the under cluster (Figure 2).

Figure 2. Each component is expressed as Kubernetes custom resources and reconciled by controllers in the under cluster.



Three properties follow from this design and recur throughout the paper:

- **kubectl apply, all the way down.** There is no proprietary CLI and no out-of-band console. The same API that deploys a workload provisions a network, a virtual machine or an operating-system image.
- **GitOps-native by default.** Desired state lives in OCI repositories; Flux reconciles the under cluster and the fleet against it continuously. Drift is detected and remediated, and every change carries an audit trail.
- **No re-implementation.** kMetal is an opinionated, hardened integration of established open-source projects – Kubernetes, Kamaji, Cluster API, KubeVirt, KVM, OVN, Kairos – with no proprietary forks. Each component can be evaluated independently against its upstream.

4.1 The two-plane deployment model

At deployment level, the four-layer stack maps onto two decoupled planes (Figure 3):

The under cluster (management plane). A Kubernetes cluster – typically three control-plane nodes plus the shared bare-metal worker fleet – that hosts all platform components and the source of truth of tenant clusters. Platform operators interact only with this cluster. Tenants never access this cluster.

The tenant plane. The set of tenant worker VMs, tenant VPCs and tenant Kubernetes API endpoints. Tenants interact only with their own cluster's API, which behaves identically to any managed Kubernetes service; they never see or touch the under cluster.

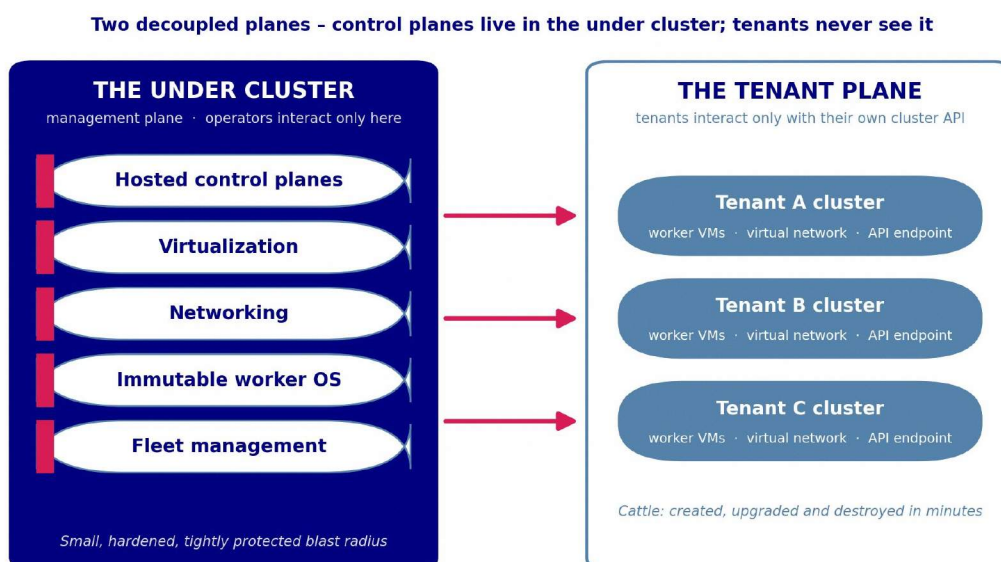


Figure 3. The under cluster is a small, hardened, tightly protected blast radius; tenant clusters are cattle, created and destroyed in minutes.

This decoupling is deliberate: the under cluster is a tightly scoped, well-understood, highly protected blast radius, while tenant clusters are cattle – created, upgraded and destroyed in minutes without coordination.

5 The Under Cluster: Platform Anatomy

The under cluster hosts six cooperating component groups. All of them are standard Kubernetes controllers; all of them are installed and upgraded as a single, versioned kMetal release.

Component	Role in the platform
GitOps & continuous reconciliation	Continuously converges the platform and every tenant cluster to a declared desired state held in a versioned source of truth.
Multi-cluster management	Manages the full lifecycle of tenant clusters and their hosted control planes, declared and reconciled as Kubernetes resources.
Networking	Per-tenant virtual networks – routers, subnets, load balancing and external connectivity – with automated TLS for reachable, certificate-secured endpoints, all programmed through Kubernetes resources.
Multi-tenancy	Enforces tenant isolation, policy and quotas across the shared platform, with self-service expressed as Kubernetes resources.
Lightweight virtualization	Runs tenant workers as hardware-isolated VMs scheduled alongside ordinary pods – no separate hypervisor layer.
Immutable OS	Signed, immutable worker OS images, built and rolled out declaratively, so nodes are replaced rather than patched in place.

5.1 Day 0: bootstrapping the under cluster

The under cluster is the one cluster kMetal does not create for itself. Installation proceeds in three stages: (1) stand up a conventional HA Kubernetes cluster on the management nodes; (2) install the kMetal platform bundle onto it via Helm / Flux; (3) run platform validation to verify networking, storage and virtualization capability before the first tenant is created. From that point on, everything else – OS images, VMs, networks, tenant clusters – is created through the platform itself.

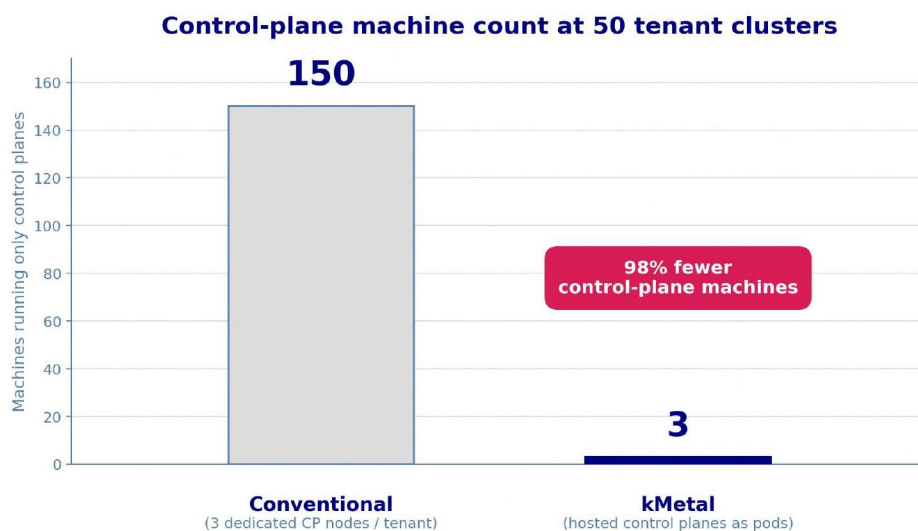
5.2 Resource profile

Because tenant control planes are pods, the marginal cost of a tenant cluster is measured in millicores, not machines: an idle hosted control plane consumes approximately 200 mCPU and 256 MiB of memory, far below the cost of a dedicated control-plane node. Sleep mode scales idle control planes to zero and restores them on demand, so development and staging clusters stop consuming resources entirely when not in use. The reference deployment runs 50 tenant clusters on 20 physical servers, with all 50 control planes hosted on 3 shared management nodes.

6 Hosted Control Planes (Kamaji)

kMetal implements hosted control planes with Kamaji, the Apache 2.0 open-source project created and maintained by the Clastix team. Each tenant cluster's control plane – kube-apiserver, controller-manager and scheduler – runs as pods in the under cluster. Tenant worker nodes connect to that control plane exactly as they would to any Kubernetes cluster; from the tenant's perspective, the model is identical to the managed Kubernetes offerings of the major clouds. The economic consequence is the elimination of the control-plane tax (Figure 4).

Figure 4. At 50 tenant clusters, dedicated control-plane machines fall from 150 to 3 – control planes become pods on shared management nodes.



6.1 The TenantControlPlane resource

The core abstraction is the TenantControlPlane custom resource: a single declarative object that defines a complete Kubernetes control plane, including replica count and resources, Kubernetes version and feature gates, API endpoint and certificate configuration, and the storage-backend binding. Creating a tenant control plane is one `kubectl apply`; Kamaji's controller reconciles the pods, services, secrets and PKI from there.

6.2 Datastore options

Cluster-state persistence is decoupled from the control plane via the Datastore resource, with three supported backends: etcd, MySQL, and PostgreSQL (CloudNativePG). The etcd backend runs in either of two modes – a dedicated instance per tenant cluster, or a shared etcd cluster serving multiple tenant clusters. In the kMetal hard-multi-tenancy posture, each tenant receives a dedicated etcd: no shared datastore fate, no cross-tenant key space, and per-tenant backup and restore.

Backend	Model	Best suited for
Dedicated etcd	One etcd instance per tenant; complete data isolation and per-tenant performance tuning	kMetal default for hard multi-tenancy – regulated tenants, paying customers, security-audited environments
Shared etcd	Multiple tenant control planes share one etcd cluster, isolated by key prefix	Very high-density fleets of internal / low-risk clusters where operational simplicity dominates
Kine (RDBMS)	SQL-backed persistence with streaming replication and point-in-time recovery	Organizations standardizing on existing RDBMS operations expertise

6.3 Per-tenant PKI

Every hosted control plane gets its own PKI hierarchy: a separate certificate authority per tenant, certificates issued at cluster creation, and automatic renewal before expiry. A compromise of one tenant's certificates does not affect any other tenant or the platform.

6.4 Connectivity: how the API server reaches the workers

Because the control plane lives in the under cluster and the workers live in an isolated tenant network, the api-server cannot open connections into the tenant VPC directly. kMetal solves this with Connectivity, the upstream Kubernetes network proxy, shipped as a sidecar in every TenantControlPlane pod (Figure 5).

- A connectivity-agent runs as a DaemonSet on each tenant worker. On startup it dials out to the tenant's connectivity-server (port 8132), which runs next to the api-server in the hosted control plane.
- All api-server → kubelet traffic (kubectl exec, kubectl logs, port-forward, metrics scraping, webhook callouts) rides back down that already-established reverse tunnel.
- The connection is initiated by the worker, so no inbound firewall rule, no per-worker NAT, and no route from the under cluster into the tenant network is required.

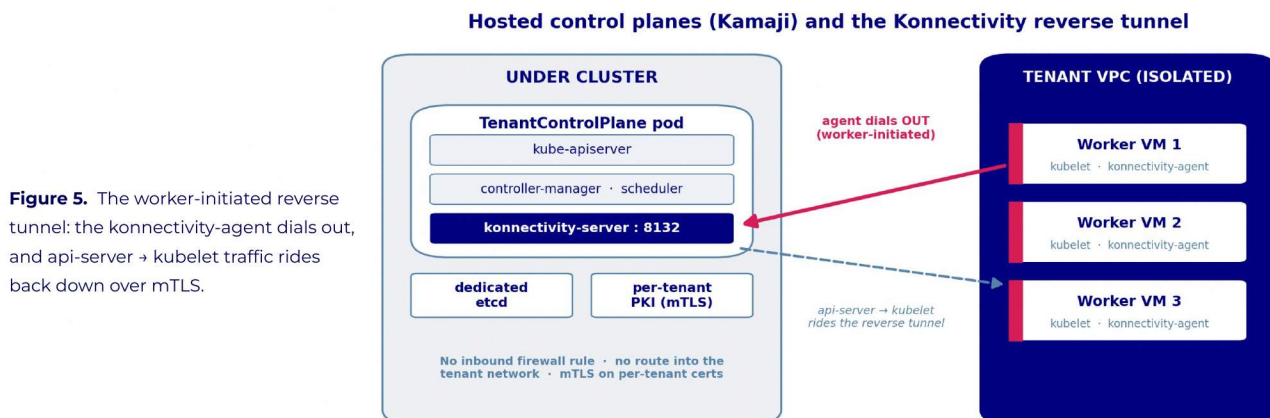


Figure 5. The worker-initiated reverse tunnel: the connectivity-agent dials out, and api-server → kubelet traffic rides back down over mTLS.

The tunnel uses mTLS with tenant-specific certificates issued by the same per-tenant PKI that signs the api-server certificates. The same outbound-only pattern is what allows workers at remote sites – co-locations, edge locations, cloud VPCs, even air-gapped networks with controlled egress – to join a hosted control plane with no infrastructure modifications at the remote site.

6.5 Control-plane high availability and upgrades

Production tenant control planes run with multiple replicas distributed across management nodes, fronted by a per-tenant LoadBalancer VIP (MetalLB) that provides automatic failover of unhealthy instances. Control-plane upgrades are blue/green: a new control-plane deployment at the target version is brought up alongside the old one and traffic is switched over, completing in approximately 16 seconds with zero API downtime.

7 Compute: KVM Workers as Kubernetes Objects

7.1 Why VMs, and why this is not “a hypervisor platform”

Modern servers are too large to serve as single worker nodes, and tying clusters to physical machine boundaries turns provisioning into a procurement-bound, weeks-long process. kMetal slices shared servers into right-sized, hardware-isolated workers using kernel-native KVM – the hypervisor already inside the Linux kernel – orchestrated by KubeVirt as first-class Kubernetes workloads.

The distinction from the legacy pattern is operational, not technological: there is no separate hypervisor layer to operate. The hypervisor is embedded in the platform and invisible to operators. There is no vCenter, no separate SDN console, no second toolchain and no virtualization licensing. Tenant worker VMs are Kubernetes objects scheduled by the under cluster's scheduler alongside ordinary pods, created and destroyed by the Cluster API KubeVirt provider (CAPK) as tenant clusters scale.

7.2 Immutable worker OS with Kairos

Worker images are not built with Packer pipelines or configured with Ansible after boot. Operators declare an immutable OS artifact as a Kubernetes object; kMetal builds it, signs it and rolls it out through the same declarative reconciliation loop used for workloads. A representative (illustrative, pre-GA) specification:

```
apiVersion: images.kmetal.io/v1alpha1
kind: ImmutableOS
metadata:
  name: tenant-gpu-node
spec:
  base: flatcar-stable
  kernel:
    modules: [nvidia, vfio_pci]
  packages:
    - name: cuda-toolkit
      version: "12.4"
  signing:
    secretRef: cosign-keys
  rollout:
    strategy: RollingDrain
    maxUnavailable: 10%
```

Immutability yields drift-free fleets: a node is never patched in place, it is replaced with a new signed artifact under a rolling-drain strategy. For GPU estates, driver- and CUDA-stamped images are versioned artifacts, which makes reclaiming, rescheduling and re-provisioning GPU nodes a repeatable, minutes-scale operation.

7.3 kPool: pre-warmed node provisioning [BETA]

For expensive hardware – GPU nodes above all – the cost of slow provisioning is direct: idle warm-up time and revenue delay when quality checks fail. kPool maintains a pool of pre-warmed, validated nodes in a service cluster before they are needed. Each node is scrubbed and re-validated between tenant allocations, and hard multi-tenancy is maintained throughout the full lifecycle, so nodes move between tenants in seconds without compromising the security boundary.

8 Networking: Per-Tenant VPCs with Kube-OVN

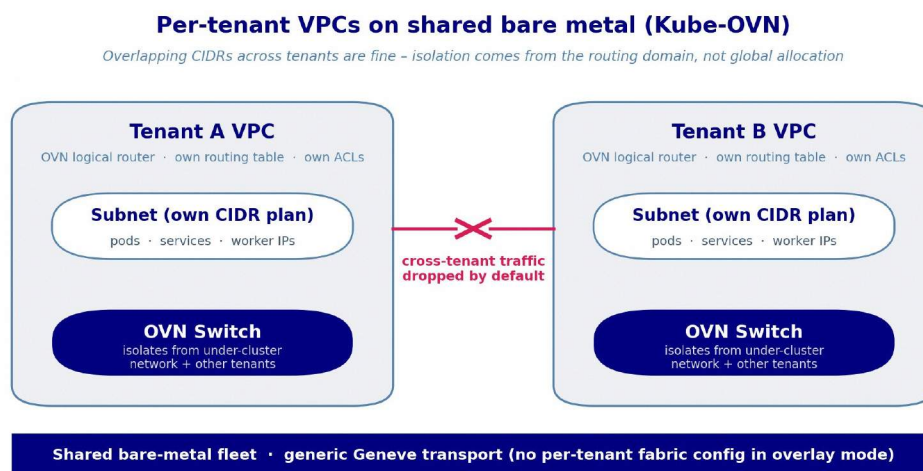
kMetal's network model gives every tenant a private virtual network on shared bare metal, built on Kube-OVN. The two top-level abstractions – VPC and Subnet – are first-class Kubernetes custom resources, and they are what a tenant claims when requesting a cluster (Figure 6).

8.1 VPC: the unit of tenant isolation

A VPC is one OVN logical router plus the logical switches attached to it: the routing domain that constitutes a tenant's private network. In kMetal, it is the unit of tenant network isolation, enforcing the following:

- **Cross-tenant traffic is dropped by default.** Two tenants on the same physical hosts cannot reach each other's pods, services or worker IPs – their routing tables simply do not see each other.
- **Hard multi-tenancy against the platform itself.** An OVN Logical Switch on every tenant VPC isolates tenant networks from the under-cluster network: the API server, platform services and other management components. A tenant cannot pivot from a compromised workload into the management plane.
- **Independent IP space.** Each tenant chooses its own pod and service CIDRs; overlapping CIDRs across tenants are fine, because isolation comes from the OVN switch layers, not from global allocation.

Figure 6. Each tenant is an isolated OVN routing domain; cross-tenant traffic and traffic toward the platform CIDR are dropped by default.



8.2 Subnets and multi-cluster tenants

A Subnet is bound to a VPC and defines the CIDR, gateway and transport parameters. One tenant cluster lives in one subnet; a tenant may hold multiple subnets in the same VPC. The kMetal ClusterClass exposes a “network.subnet” variable so a single tenant can run separate dev, staging and production clusters with separate IP planes inside one isolated network boundary.

8.3 Transport: overlay

Tenant networks run over a Geneve overlay: tunnels between under-cluster nodes carry generic Geneve traffic, so the switch fabric needs no per-tenant configuration. Adding a tenant is a pure software operation with no network-team involvement. The only fabric requirement is MTU 9100 (jumbo frames) on the overlay transport VLAN.

8.4 The provider network: egress and external IPs

Separate from tenant VPCs, the under cluster maintains a provider network – a dedicated VLAN that carries tenant SNAT / egress traffic and hosts the externally reachable public IP addresses used for ingress. A distributed gateway runs an external OVN Logical Router Port on every node, so any node can perform SNAT for any tenant using a dedicated IP address – there is no single egress chokepoint. For ingress from the internet to tenant services, a tenant LoadBalancer service is created for each service, receiving an external IP from the provider network. An external edge router with firewall functions can be placed in front of the provider network to protect incoming traffic; kMetal publishes the right IPs to the edge, which handles the public-facing side, and the two remain decoupled. Multiple provider networks are supported, including one per tenant.

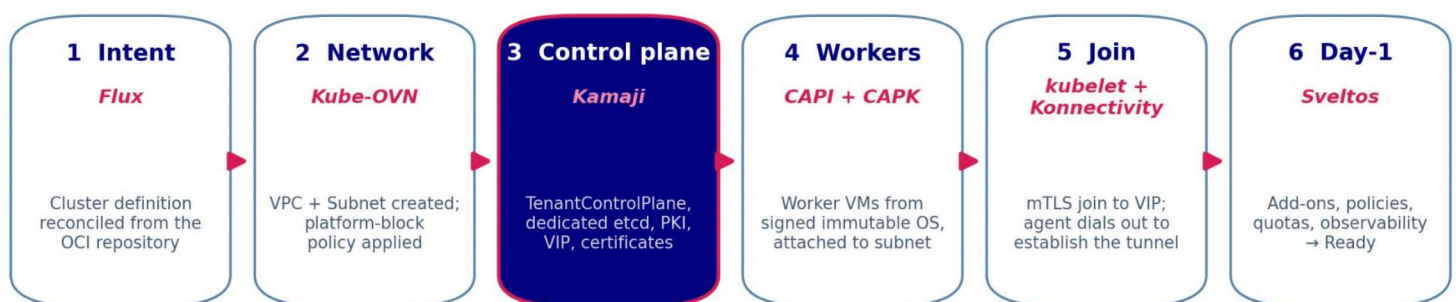
Self-service extends to networking: tenants claim their own virtual-network environments through the platform's claim mechanism, and VPC Peering [BETA] connects environments across tenants under policy control set by the platform team.

9 End-to-End: The Life of a Tenant Cluster

The sections above describe the layers in isolation; this section traces the wiring between them. When a tenant cluster is requested – via kubectl, a GitOps commit, or the self-service dashboard – a single declarative object set triggers the following reconciliation, with the independent controllers working largely in parallel (Figure 7). The full sequence completes in approximately two minutes.

The life of a tenant cluster - one declarative object set, ~2 minutes to Ready

Deletion is the same flow in reverse, equally declarative



independent controllers reconcile largely in parallel

Figure 7. One declarative object set fans out to independent controllers that reconcile in parallel, reaching a production-ready cluster in about two minutes.

Step	Controller	What happens
1. Intent	Flux	The cluster definition (CAPI Cluster + ClusterClass variables: version, size, subnet, profile) is committed and reconciled from the OCI repository into the under cluster.
2. Network	Kube-OVN	The tenant's VPC (default switch + logical router + ACLs + default platform-block policy) and the target Subnet are created or matched. IP space is allocated inside the tenant's own CIDR plan.
3. Control plane	Kamaji	A TenantControlPlane is instantiated: api-server, controller-manager and scheduler pods start on the shared management nodes; a dedicated etcd is provisioned; the per-tenant PKI is generated; MetalLB assigns the control-plane VIP and cert-manager issues its serving certificate.
4. Workers	Cluster API + CAPK	Worker nodes are created as KubeVirt VMs from the tenant's signed immutable OS image, attached to the tenant subnet, and booted.
5. Join	kubelet + Konnectivity	Workers boot with join configuration, authenticate to the control-plane VIP over mTLS, and the konnectivity-agent dials out to establish the reverse tunnel for api-server → kubelet traffic.
6. Day-1 state	Sveltos	The cluster profile applies add-ons, admission policies, security baselines, quotas and observability wiring. The cluster is production-ready the moment it reports Ready – no post-provisioning checklist.

Deletion is the same flow in reverse and equally declarative: removing the cluster object tears down workers, control plane, PKI and network claims, with the dedicated etcd's data lifecycle governed by the backup policy in effect.

10 The Isolation Model: Three Independent Boundaries

kMetal's multi-tenancy claim rests on three boundaries that fail independently (Figure 8) – a design intended to satisfy security audits and regulated-customer requirements without a second infrastructure platform. “Multi-tenant” here means independent clusters, not shared namespaces with additional YAML.

Three independent isolation boundaries - a breach is contained to one cluster

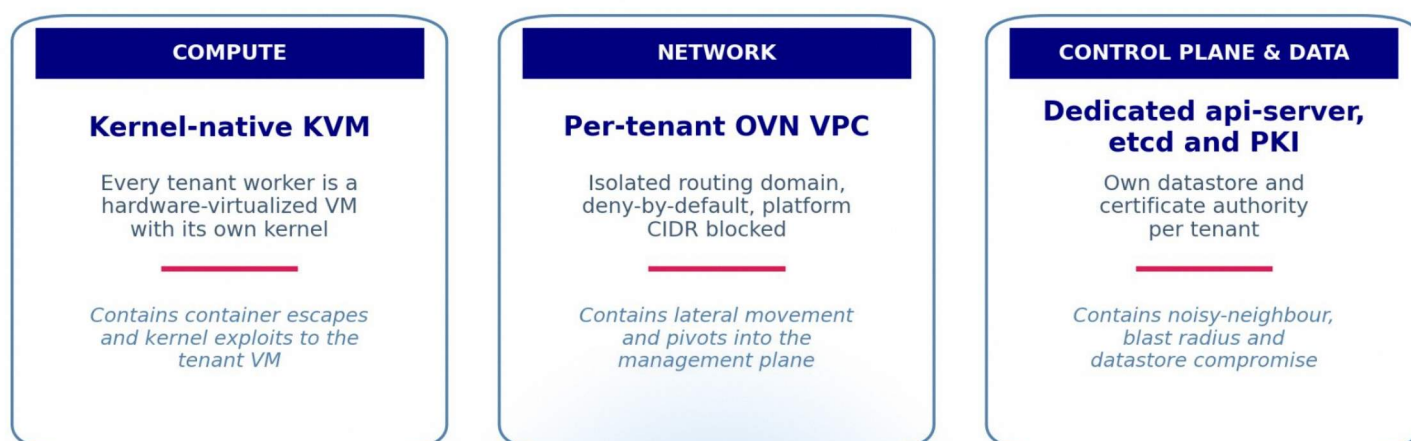


Figure 8. Compute, network, and control-plane / data boundaries fail independently, so the compromise scenario a reviewer must analyse is bounded at a single tenant.

Boundary	Mechanism	What it contains
Compute	Kernel-native KVM: every tenant worker is a hardware-virtualized VM with its own kernel	Container escapes and kernel exploits are contained to the tenant's own VMs; no shared-kernel exposure across tenants
Network	Per-tenant OVN VPC: isolated routing domain, deny-by-default cross-tenant, platform CIDR blocked by router policy	Lateral movement between tenants and pivoting from tenant workloads into the management plane
Control plane & data	Dedicated api-server, dedicated etcd, dedicated PKI per tenant	Noisy-neighbour and blast radius on the API path; datastore compromise or corruption confined to one tenant

On top of the boundaries, governance is applied at the tenant edge rather than inside it: resource quotas (CPU, memory, GPU, storage) and admission policies are enforced at the boundary and observable in Prometheus, while tenants retain full cluster-admin inside their own cluster – the complete upstream API, their own versions, add-ons and policies, on an independent life cycle.

A breach is contained to one cluster. The combination – separate kernel, separate network domain, separate datastore, separate PKI – means the compromise scenario a reviewer must analyse is bounded at the single-tenant level by architecture, not by configuration discipline.

11 Failure Domains and High Availability

11.1 What fails, what keeps running

The decoupling of hosted control planes from tenant workers produces a failure behaviour worth stating explicitly, because it is one of the architecture's strongest properties: tenant workloads do not depend on the under cluster's availability for steady-state operation.

- **Loss of a tenant control plane (or the under cluster).** Kubelets and workloads on tenant workers continue running and serving traffic; the kubelet is designed to operate through api-server unavailability. What is lost until recovery is orchestration: no new scheduling, scaling or configuration changes. Kamaji's controller restores control-plane pods automatically on healthy management nodes.
- **Loss of a management node.** Control-plane replicas are distributed across management nodes; the per-tenant VIP fails over to healthy replicas automatically.
- **Loss of a physical worker host.** Tenant VMs on that host are rescheduled by the platform to remaining capacity, subject to the tenant's quota; the tenant cluster sees it as ordinary node failure and re-schedules pods accordingly.
- **Loss of a full site.** See stretched clusters below.

11.2 Stretched clusters: surviving a site failure

For two-site deployments, kMetal supports stretched under clusters with synchronous storage replication between sites and a lightweight cloud quorum witness for split-brain protection. The platform survives a full site failure with zero RPO (no data loss) and automatic recovery of tenant control planes on the surviving site – without requiring a third physical site and without a manual failover procedure.

11.3 Backup and recovery

Per-tenant dedicated etcd makes backup and restore a tenant-scoped operation rather than a shared-fate one. Platform-level backup covers the under cluster's own state and the fleet configuration (which, being GitOps-managed, is inherently re-creatable from the repository); tenant-level backup covers each tenant's etcd and, where configured, workload data via the tenant's own tooling.

12 Operating the Fleet

12.1 Cluster profiles and continuous reconciliation

Fleet management in kMetal is the same reconciliation pattern applied one level up. A cluster definition defines desired state once – Kubernetes version policy, add-ons, security baselines, admission policies, observability wiring – and a continuous loop enforces it across every cluster in the fleet. Upgrades, patching, certificate rotation and add-on delivery run automatically; drift is detected and remediated; new clusters arrive production-ready the moment they exist. The reference figure: 100 control planes reconciled in under 150 seconds.

12.2 Self-service within guardrails

Tenants self-serve – clusters, scaling, upgrades within allowed versions, virtual networks – through the API or the tenant dashboard, inside boundaries the platform team defines: quotas, allowed profiles, policy baselines. The platform team operates the platform; it does not operate every cluster.

12.3 Observability

Monitoring, logging and alerting ship with the platform: control-plane resource usage, request latency and throughput per tenant, and fleet-wide state in a single pane. Quota consumption and tenant metrics are exposed in Prometheus format, integrating with existing Grafana / alerting stacks rather than replacing them.

12.4 Ecosystem integration

kMetal connects to existing infrastructure rather than replacing it. Storage is consumed through the standard CSI interface (validated with NetApp, Pure, Dell, HPE and Ceph). Automation surfaces include Cluster API, Terraform, GitOps (Flux) and Helm – the toolchain most platform teams already run.

13 Performance and Scale Reference

The figures below reflect the reference deployment (50 tenants on 20 physical servers) and internal analysis of on-premises deployments. They are the numbers behind the claims used across kMetal materials, consolidated with their context.

Metric	Figure	Context
Idle control-plane footprint	~200 mCPU / ~256 MiB per tenant	Hosted control-plane pods at idle; sleep mode reduces idle environments to zero
Tenant cluster provisioning	~2 minutes	Full isolated cluster: control plane, network, workers, day-1 profile – single workflow
Control-plane upgrade	~16 seconds, zero downtime	Blue/green replacement of the hosted control plane
Fleet reconciliation	100 control planes in < 150 s	Profile-driven convergence across the fleet
Control-plane machines	150 → 3 at 50 tenants	Dedicated machines eliminated; control planes as pods on 3 shared nodes
TCO: depreciation	> 90% reduction	Internal analysis, on-premises deployments
TCO: SRE operations	> 65% reduction	One operational model, no fragmented toolchains
TCO: productivity	> 60% reduction	Provisioning in seconds instead of multi-stage workflows
TCO: energy & facility	> 50% reduction	Fewer servers to power and cool

Note: Figures above are illustrative, drawn from representative deployments; actual results vary by environment, workload and configuration.

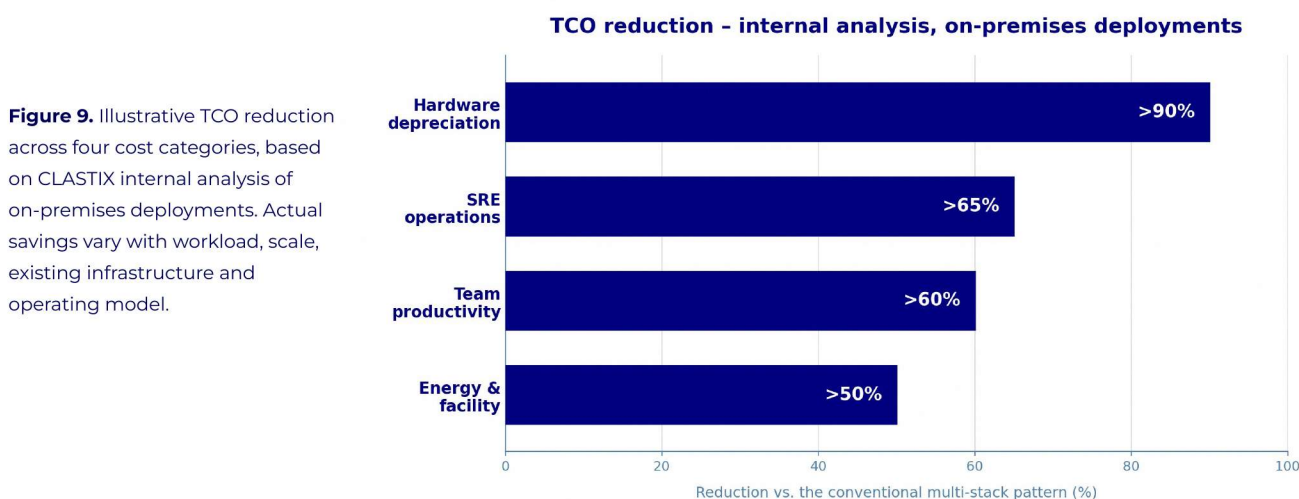
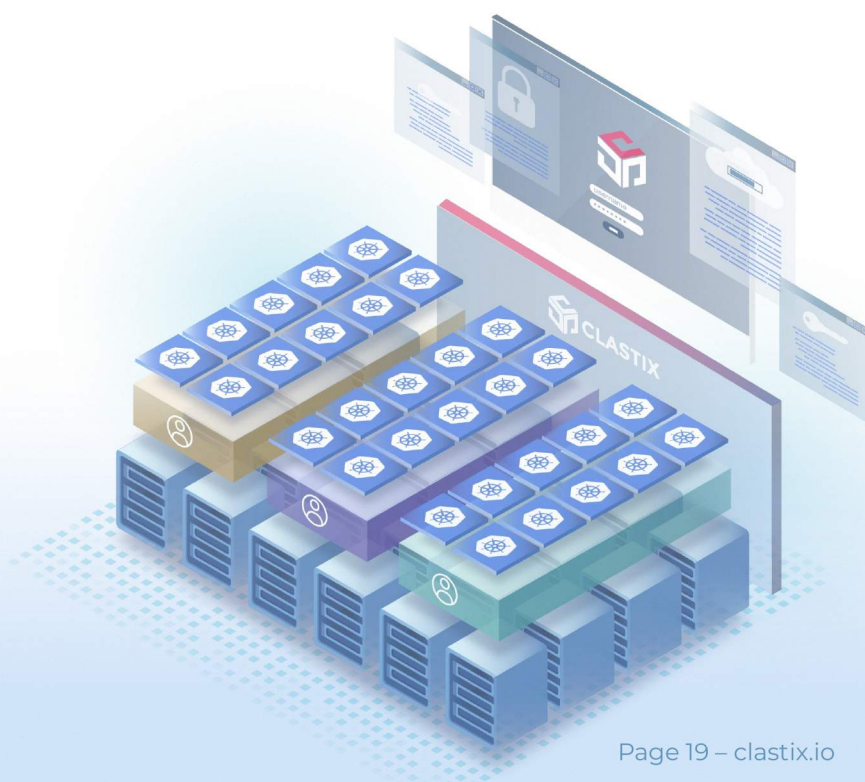


Figure 9. Illustrative TCO reduction across four cost categories, based on CLASTIX internal analysis of on-premises deployments. Actual savings vary with workload, scale, existing infrastructure and operating model.

14 Open-Source Foundation and Commercial Platform

kMetal is built on Kamaji, the Apache 2.0 hosted-control-plane project created and maintained by the CLASTIX team. Teams can validate the core architecture with the open-source project; kMetal shares that proven core and adds what a production bare-metal fleet requires, delivered as one platform rather than a stack of separately assembled products.

Capability	Kamaji (open source, Apache 2.0)	kMetal (commercial platform)
Workload isolation	Select and integrate a hypervisor yourself	Kernel-native KVM – separate kernel per tenant, embedded
Tenant networking	None	OVN software-defined networking – isolated domain per tenant
Self-service networking	None	Tenant environments via claim mechanism; VPC Peering [BETA]
Fleet management	Assemble with FluxCD / ArgoCD	Profiles, bulk operations, hundreds to thousands from one plane
Governance & policy	None	Policy enforcement, drift remediation, guardrails, self-service
Cluster lifecycle	Create, upgrade, delete (basic)	Full automation; blue/green upgrades ~16 s, zero downtime
Immutable OS	None	Signed, immutable worker OS images, built and rolled out declaratively
GPU / accelerators	None	kPool pre-warmed pool; GPU nodes in seconds [BETA]
Availability & DR	None – delegated to the infrastructure	Multi-site stretched clusters, automated failover, zero RPO
Support	Community (GitHub, Slack)	Enterprise SLA – dedicated engineers, defined response times
Licensing	Apache 2.0	Commercial subscription

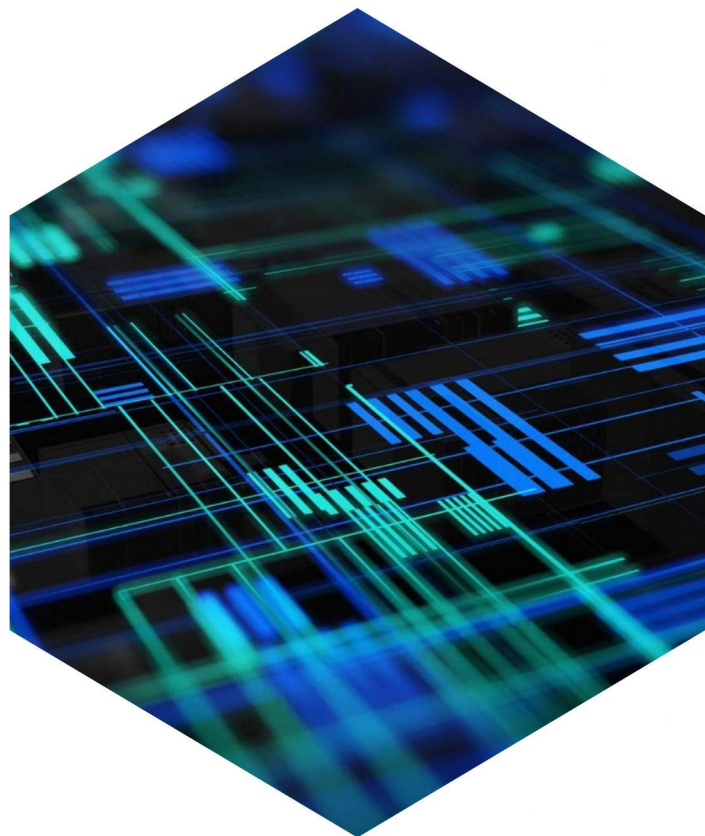


15 Conclusion and Next Steps

kMetal's architectural bet is simple to state: if Kubernetes is the operational model your team already trusts, then the control planes, the virtual machines, the networks and the operating system underneath them should all be Kubernetes objects too. The result is one stack, one API, one team – with hard multi-tenancy enforced by three independent boundaries (kernel, network, datastore) rather than by a second infrastructure platform, and with per-tenant cost measured in millicores rather than machines.

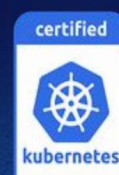
Clastix works hands-on with each customer or design partner to validate the architecture within the specific environment.

- Evaluate kMetal's core architecture today with Kamaji (open source): kamaji.clastix.io
- Learn more about kMetal: kmetal.clastix.io
- Read the kMetal platform documentation: kmetal.clastix.io/docs
- Request a demo or design-partner evaluation: clastix.io/contact – evaluation access includes full platform capability, technical support and architecture consultation.



Talk to us

Please contact us to request a quote or learn more about our support offerings.



CLASTIX SRL
V. Francesco Caracciolo, 17
80122 Naples, Italy

VAT No. IT11183320966
REA: NA-1152378
© 2026 Clastix | All Rights Reserved

hello@clastix.io
www.clastix.io
<https://github.com/clastix>

16 Glossary of Acronyms and Abbreviations

ACL — Access Control List
AI — Artificial Intelligence
API — Application Programming Interface
BGP — Border Gateway Protocol
CAPI — Cluster API (declarative Kubernetes cluster-lifecycle project)
CAPK — Cluster API Provider for KubeVirt
CIDR — Classless Inter-Domain Routing
CLI — Command-Line Interface
CNCF — Cloud Native Computing Foundation
CNI — Container Network Interface
CPU — Central Processing Unit
CRD — Custom Resource Definition
CSI — Container Storage Interface
CUDA — Compute Unified Device Architecture (NVIDIA GPU platform)
DORA — Digital Operational Resilience Act (EU financial-sector regulation)
etcd — distributed key-value store backing a Kubernetes control plane
Geneve — Generic Network Virtualization Encapsulation
GPU — Graphics Processing Unit
HA — High Availability
ICP — Ideal Customer Profile
IDC — International Data Corporation (analyst firm cited in Section 2)
IP — Internet Protocol
KVM — Kernel-based Virtual Machine (Linux kernel hypervisor)
Kine — Kine Is Not Etcd; a shim that lets Kubernetes use a relational database (RDBMS) instead of etcd
mCPU — millicpu (Kubernetes CPU unit; 1,000 mCPU = 1 core)
MiB — mebibyte (2^{20} bytes)
MIG — Multi-Instance GPU (NVIDIA GPU partitioning)
MSP — Managed Service Provider
mTLS — mutual Transport Layer Security
MTU — Maximum Transmission Unit
NAT — Network Address Translation
NSX — VMware NSX (network-virtualization / SDN product)
OCI — Open Container Initiative (image / artifact format)
OS — Operating System
OVN — Open Virtual Network
OVS — Open vSwitch
PKI — Public Key Infrastructure
RBAC — Role-Based Access Control
RDBMS — Relational Database Management System
RPO — Recovery Point Objective
RTO — Recovery Time Objective
SDN — Software-Defined Networking
SLA — Service-Level Agreement
SNAT — Source Network Address Translation
SQL — Structured Query Language
SR-IOV — Single-Root I/O Virtualization
SRE — Site Reliability Engineering
TCO — Total Cost of Ownership
TLS — Transport Layer Security
VFIO — Virtual Function I/O (Linux device-passthrough framework)
VIP — Virtual IP address
VLAN — Virtual Local Area Network
VM — Virtual Machine
VPC — Virtual Private Cloud
YAML — YAML Ain't Markup Language