Modernizing Virtualization: Migrating from VMware to KubeVirt on Bare Metal

A technical implementation guide featuring Portworx Enterprise and Everpure FlashArray

Authors: Sagar Srinivasa, Jun Park, Charudath Gopal

Audience: VMware customers (infrastructure leaders, architects, operations teams) planning to migrate to an open-source, cloud-native alternative without sacrificing enterprise features.

Executive Summary

Organizations that have standardized on VMware face a new reality: rising licensing costs, increasing vendor lock-in, and a widening gap between traditional virtualization and modern, cloud-native operations. At the same time, many critical workloads still depend on VMs and cannot be rewritten overnight.

This technical guide details a strategy for transitioning from VMware to an open-source virtualization platform using **KubeVirt** on **bare-metal Kubernetes**. By integrating **Portworx Enterprise** and **Everpure FlashArray**, organizations can maintain enterprise-grade storage performance and data protection while avoiding rising licensing costs. The architecture promotes a unified operational model where virtual machines and containers coexist on a single control plane managed through GitOps and automated workflows. Key features like live migration, high availability, and high-speed networking via Cilium ensure that the new environment matches the reliability of traditional hypervisors. Ultimately, the source provides a comprehensive roadmap for modernizing legacy workloads without sacrificing the operational maturity expected by infrastructure teams.

Key Outcomes We Achieved

- **Matched VMware operational workflows** using KubeVirt and a purpose-built VM lifecycle UI, including live migration, snapshots, console access, and resource management.
- **Unified storage for VMs and containers** with Portworx Enterprise on Everpure FlashArray, providing enterprise-grade data protection and resilience.
- **Automated everything as code** with GitOps (ArgoCD) for platform components and Ansible/GitHub Actions for bare-metal lifecycle.
- **Migrated workloads seamlessly using Forklift** — an open-source migration toolkit that orchestrates vCenter connectivity, disk transfer via CDI/VDDK, automated guest conversion with virt-v2v, and DNS updates. For environments backed by Everpure FlashArray, XCOPY storage offload enabled fast, host-bypass disk transfers for large workloads. Standardized workloads benefited from near-seamless migration, while complex workloads required targeted remediation.

For VMware customers, the key takeaway is simple: you can reduce costs, avoid lock-in, and accelerate modernization without giving up the operational maturity and safety net you depend on today.

Why Move Off of VMware Now?

VMware has been the de facto standard for enterprise virtualization for over a decade. However, several structural pressures are driving organizations to reconsider:

- **Escalating TCO and licensing complexity** – Core-based licensing and bundled offerings create step-function cost increases as environments grow. Renewal cycles are increasingly seen as inflection points to evaluate alternatives.
- **Vendor lock-in and strategic risk** – Deep integration into a single proprietary ecosystem makes it harder to adopt cloud-native tooling, multi-cloud strategies, or open-source innovation at the pace the business demands.
- **Split-brain operations for VMs and containers** – Many enterprises now operate both VMware clusters and separate Kubernetes platforms. This leads to duplicated tooling, skill sets, and processes—and often fractured ownership between teams.

The approach we describe embraces KubeVirt on bare metal to consolidate these worlds:

- One platform for VMs and containers, using Kubernetes as the common control plane.
- Open source core (Kubernetes + KubeVirt) with enterprise-grade data services (Portworx Enterprise + Everpure FlashArray).
- GitOps and automation everywhere, dramatically simplifying change management, compliance, and repeatability.

Reference Architecture Overview

At a high level, the modernized platform comprises of:

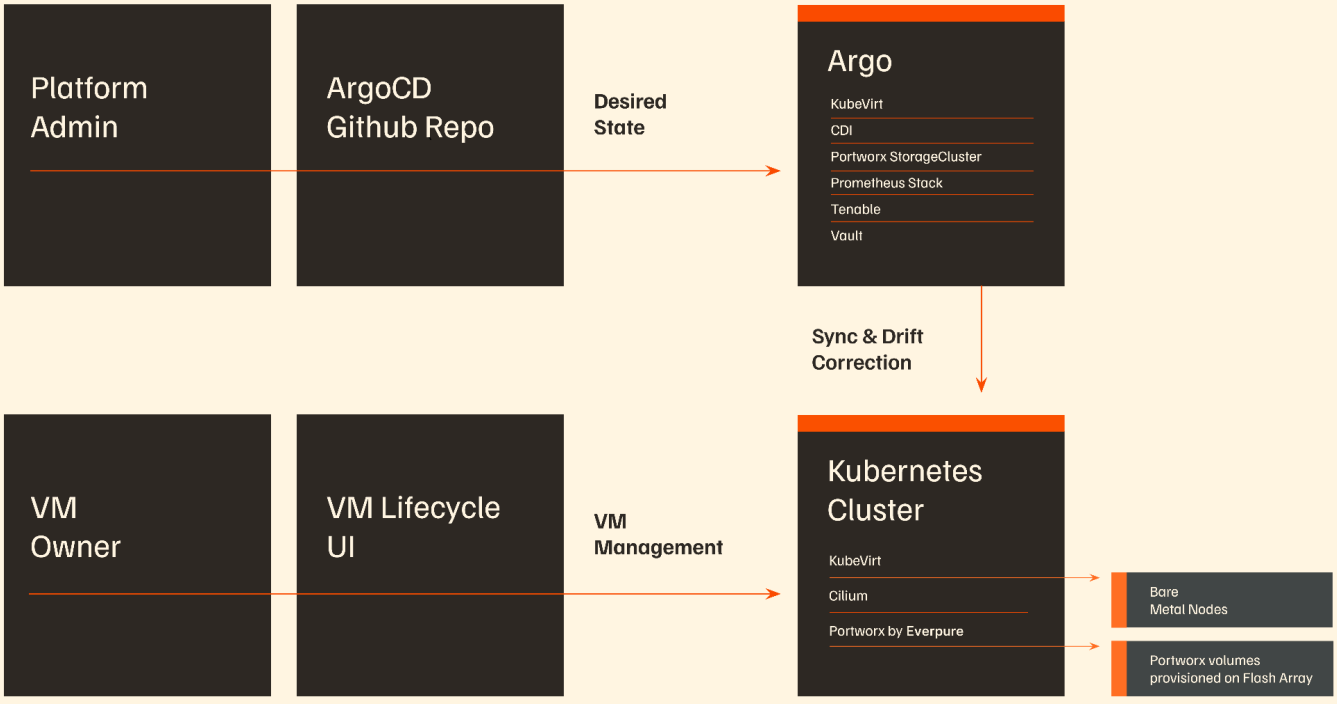

Figure 1.

A fleet of physical servers with access to Everpure FlashArray, providing high-performance, resilient storage. Nodes are provisioned and re-provisioned via Ansible playbooks triggered from GitHub Actions.

The reference implementation described in this paper runs on:

- **Vanilla Kubernetes** 1.31.4 on bare metal, using **Cilium** as the CNI.
- **KubeVirt** 1.5.1 for virtualization, paired with **Containerized Data Importer** (CDI) 1.62.0 for VM image import and cloning.
- **Portworx Enterprise** 3.4.x as the Kubernetes data platform, backed by **Everpure FlashArray**. Portworx exposes RWX volumes using FlashArray Direct Access (FADA) for VM disks and container workloads.
- A set of GitOps-managed platform add-ons including:
    - Certificates and ingress for TLS termination and HTTP/S routing,
    - Secrets and key management integrated with an enterprise secrets platform,
    - Monitoring and logging stacks for metrics, events, and log collection,
    - Security and governance components for centralized RBAC and vulnerability scanning,
    - KubeVirt management and tooling (dashboards, console proxy, supporting controllers).

All of these components are managed declaratively via Git and continuously reconciled by a GitOps controller, so the cluster can be rebuilt or audited from source of truth.

Networking: From Port-Groups and NSX to Cilium

Migrating from VMware networking constructs (VLAN-backed port-groups, distributed switches, and NSX) to Kubernetes-native networking required a careful redesign. The target state uses Cilium as the CNI, CiliumIngress for north–south traffic, and a custom DNS operator to keep DNS in sync with dynamic IPs.

Cilium CNI: eBPF-Powered Networking

Cilium serves as the Container Network Interface (CNI) for the cluster, replacing traditional iptables-based networking with eBPF (extended Berkeley Packet Filter) programs that run directly in the Linux kernel. This provides significant advantages for a bare-metal KubeVirt environment:

- **High-performance datapath** – eBPF bypasses the traditional Linux networking stack for pod-to-pod traffic, reducing latency and CPU overhead. This is especially important for VM workloads that generate significant network I/O.
- **Network policy enforcement** – Cilium enforces Kubernetes NetworkPolicy and its own CiliumNetworkPolicy CRDs at the kernel level, providing microsegmentation between VMs and containers without the overhead of userspace proxies.
- **Identity-based security** – Rather than relying solely on IP addresses (which are dynamic), Cilium assigns cryptographic identities to workloads. Policies can reference these identities, making security rules portable across node migrations.
- **Observability** – Cilium's Hubble component provides deep visibility into network flows, DNS queries, and HTTP transactions without requiring application instrumentation.

BGP Integration for Bare-Metal Routing

On bare metal, there is no cloud provider load balancer or overlay network provided by a hypervisor. Cilium's BGP integration solves this by peering directly with the upstream network infrastructure:

How it works

- Each Kubernetes node runs a BGP speaker (built into Cilium) that peers with Top-of-Rack (ToR) switches or a BGP route reflector.
- Cilium advertises the **pod CIDR** assigned to each node, so upstream routers know how to reach pods on that node.
- For **LoadBalancer Services**, Cilium advertises the external IP (VIP) from all nodes where the service has healthy endpoints, enabling ECMP (Equal-Cost Multi-Path) load balancing at the network layer.

Configuration highlights

```
none
# CiliumBGPPeeringPolicy (simplified example)
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPPeeringPolicy
metadata:
  name: cluster-bgp-policy
spec:
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: "true"
  virtualRouters:
    - localASN: 65001
      exportPodCIDR: true
      neighbors:
        - peerAddress: "10.0.0.1/32"   # ToR switch
          peerASN: 65000
      serviceSelector:
        matchExpressions:
          - key: someLabel
            operator: NotIn
            values: ["never-advertise"]
```

Benefits over VMware networking

VMware Approach	Cilium + BGP Approach
Static VLAN assignments per port-group	Dynamic pod CIDRs advertised per node
Manual IP planning and reservation	IPAM handled by Kubernetes, routed via BGP
NSX or hardware load balancers for VIPs	LoadBalancer VIPs advertised via BGP with ECMP
Complex distributed switch configuration	Declarative CRDs managed via GitOps

Table 1.

CiliumIngress for North-South Traffic

For HTTP/S ingress, CiliumIngress replaces traditional ingress controllers and VMware NSX load balancers:

- **Gateway API support** – Cilium implements the Kubernetes Gateway API, providing a standardized way to define ingress routes, TLS termination, and traffic splitting.

- **Envoy-based data plane** – Under the hood, Cilium uses Envoy proxies for L7 processing, providing advanced features like header-based routing, rate limiting, and observability.
- **Stable VIPs** – Ingress VIPs are advertised via BGP, so external clients reach applications through stable, routable addresses without relying on external load balancer appliances.

For operators migrating from VMware, CiliumIngress provides the same "stable VIP + hostname" model they're used to, but defined declaratively in Kubernetes and managed via GitOps.

Pod Masquerade Networking for KubeVirt VMs

KubeVirt VMs use **pod masquerade networking**:

- Each VM runs inside a pod that owns the real IP address on the Kubernetes network.
- Inside the VM, a virtual NIC is presented with an internal IP; the pod performs NAT (masquerading) for outbound and inbound traffic.
- This model lets VMs participate in the same network and policy model as pods, while preserving familiar L2/L3 behavior for the guest OS.

For operators coming from VMware, pod masquerade plays the role that port-groups and distributed switches did previously, but in a Kubernetes-native way.

Trade-offs: Static IPs vs. Live Migration

In VMware, networking was straightforward: VMs attached directly to VLAN-backed port-groups via bridge networking on host interfaces, giving each VM a stable, static IP. However, this model does not translate directly to KubeVirt when using Cilium as the CNI.

Why the trade-off exists

- With Cilium, pod subnets are advertised over BGP to the upstream network. VMs (running as pods) receive dynamic IPs from the pod network, not static IPs from a traditional VLAN.
- This dynamic IP model is what enables **live migration**: because the VM's network identity is tied to the pod abstraction rather than a specific host interface, KubeVirt can move the VM between nodes without breaking connectivity.
- If VMs were assigned static IPs via bridge networking (as in VMware), live migration would not work with Cilium because the IP would be bound to a specific node's network interface.

Guidance for migration planning

Requirement	Recommended Approach
Live migration is critical	Use Cilium with masquerade networking. Rely on DNS-based service discovery.
Live migration + static IP required	Use Cilium (masquerade) as primary and attach a Multus secondary bridge interface for static IP assignment.
Highly legacy/static-IP dependent workloads	Use a dual-interface model and register a secondary interface in DNS.

Table 2.

For most workloads, the shift from static IPs to DNS-based service discovery is a worthwhile trade-off that unlocks the full benefits of KubeVirt's live migration and Kubernetes-native networking.

Automated DNS with a Custom DNS Operator

Because IPs and endpoints in Kubernetes are more dynamic than in traditional VMware environments, DNS updates must be event-driven rather than ticket-driven.

DNS management is implemented through an event-driven controller that integrates with our existing DNS infrastructure.

- The **controller** watches selected Kubernetes and KubeVirt resources (e.g., annotated VMs or specific Services).
- When relevant IPs or virtual IPs change (due to live migration, rescheduling, or Service updates), the controller triggers DNS updates through the corporate DNS integration layer
- This removes the manual, ticket-driven DNS updates that were common with VMware port-groups and static IP assignments, while ensuring that application FQDNs remain stable even as workloads move.

The result is a networking model that provides the same predictability and DNS stability VMware teams expect, while embracing Kubernetes-native constructs and automation.

KubeVirt: Matching VMware Features

For VMware customers, the biggest question is usually not “Can KubeVirt run VMs?” but “Can KubeVirt support our VMware workflows?” Below are the most important challenges and how we addressed them.

Replacing vCenter VM Lifecycle Management

Challenge

vCenter provides a unified console for creating, powering, cloning, and resizing VMs. Moving to KubeVirt means these operations are expressed as Kubernetes resources (VirtualMachine, VirtualMachineInstance), which are not familiar to most VMware admins or application teams.

Approach

KubeVirt as the VM control plane

KubeVirt CRDs represent VM specifications: CPU, memory, disks, networks, and policies. These specifications are fully declarative Kubernetes objects. In this deployment, they are created and managed programmatically by the homegrown UI rather than being stored as Git-tracked manifests.

Homegrown VM Lifecycle portal (KIF)

We built a UI that exposes familiar VM operations:

- Create VMs from a library of templates (OS images, sizes).
- Power on/off, reboot, and delete VMs.
- Resize CPU/RAM within allowed guardrails.
- Create and revert snapshots (through Everpure-backed storage under the hood).
- Access a web-based console to the guest for troubleshooting and day-2 operations.

Under the hood, the UI interacts with Kubernetes and KubeVirt APIs, translating user actions into updates to VirtualMachine resources. This mimics vCenter workflows while leveraging Kubernetes semantics.

Example: VirtualMachine Manifest

Below is a representative VirtualMachine resource from a production cluster. Note the key elements that enable enterprise VM operations:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: team-namespace
spec:
  runStrategy: Always
  dataVolumeTemplates:
  - metadata:
      name: example-vm-disk-0
    spec:
      pvc:
        accessModes:
        - ReadWriteMany              # RWX enables live migration
        resources:
          requests:
            storage: 1000Gi
        storageClassName: sc-fada-rwx-immediate  # Portworx FADA storage class
        volumeMode: Block
      source:
        pvc:
          name: example-vm-disk-0    # Source PVC (from migration or clone)
          namespace: team-namespace
  template:
    spec:
      domain:
        cpu:
          cores: 20
        memory:
          guest: 64Gi
        devices:
          disks:
          - disk:
              bus: virtio
            name: rootdisk
          - disk:
              bus: virtio
            name: cloudinitdisk
          interfaces:
          - masquerade: {}           # Pod masquerade networking
            name: default
        machine:
          type: q35
      networks:
      - name: default
        pod: {}
      volumes:
```

```
    - dataVolume:
        name: example-vm-disk-0
      name: rootdisk
    - cloudInitNoCloud:
        networkDataBase64: <base64-encoded-netplan>
        userDataBase64: <base64-encoded-cloud-init>
      name: cloudinitdisk
```

Key observations

- **RWX access mode** with `sc-fada-rwx-immediate` storage class enables live migration
- **DataVolumeTemplate** provisions the disk from a source PVC (migrated or cloned)
- **Masquerade networking** provides pod-based connectivity with NAT
- **Cloud-init** configures hostname, networking, and first-boot scripts
- **virtio devices** for optimal I/O performance

Git-backed templates and policies

Policies and guardrails for the platform—such as storage classes, RBAC, monitoring, and security agents—are stored as templates in Git and applied with GitOps. VM lifecycle operations are handled through the UI, which manages the underlying Kubernetes/KubeVirt resources.

Outcome

The operations experience feels similar to vCenter, while the underlying VM resources are Kubernetes-native objects. Infrastructure and policies remain declarative and GitOps-driven, and the UI provides a familiar interface for day-to-day VM lifecycle operations.

High Availability, Maintenance, and Live Migration

Challenge

VMware vMotion and HA set a high bar for uptime during host maintenance and failures. An alternative platform must provide comparable resilience and controlled maintenance workflows.

Approach

- **Kubernetes HA and node pools** – Multiple control-plane and worker nodes provide resilient scheduling and capacity. KubeVirt VMs are scheduled like pods, respecting node labels for hardware specialization (e.g., virtualization-enabled nodes, SR-IOV).
- **KubeVirt live migration (vMotion-like behavior)** – KubeVirt live migration is enabled for VMs that use RWX Portworx volumes backed by FlashArray Direct Access (FADA). VM disks stay on the same shared RWX volume; only CPU and memory state are streamed to the target node. Because the storage layer is already accessible from every eligible worker node, migrations are fast and do not involve copying disks, closely mirroring the vMotion model. Migration policies can be tuned (e.g., timeouts, bandwidth constraints) to balance speed and workload safety.
- **Portworx-backed storage for resilience** – VM disks are backed by Portworx volumes on FlashArray, ensuring that storage is highly available independent of any single node. This allows VMs to restart or migrate without data loss.

Outcome

Planned maintenance and unplanned node failures can be handled without major downtime, satisfying the same operational expectations teams had with VMware.

Storage, Snapshots, and Backup/Restore

Challenge

VMware customers are used to powerful storage workflows—VM snapshots, clones, and integrations with backup frameworks. A modern alternative must provide similar or better data services.

Approach

- **Portworx Enterprise as data fabric** – Portworx Enterprise acts as the CSI driver and data management layer for both containers and VMs. Persistent volumes for VMs are created via standard Kubernetes PersistentVolumeClaim objects.
- **FlashArray integration** – Underneath Portworx, Everpure FlashArray provides consistent low-latency performance and array-level resilience, enabling fast snapshots and clones for VM volumes and efficient replication and backup options.
- **Policy-based data protection** – Backup and snapshot policies are defined per storage class or namespace, aligning with workload tiers. This replaces the VM-level snapshot policies typically managed in VMware.

Outcome

VMs gain robust, policy-driven data protection with equal or greater flexibility than traditional VMware-integrated solutions, while containers benefit from the same data platform.

Resource Management: Auto Memory Limits and Overcommit

VMware admins are familiar with reservation/limit concepts and safely overcommitting CPU and memory on a host. KubeVirt on Kubernetes provides similar controls:

- **Automatic memory limit behavior** – The platform derives VM memory limits automatically from the requested guest memory plus a controlled overhead, instead of requiring every VM owner to hand-tune limits. Operators define standard size classes or templates (e.g., small/medium/large), and limits are enforced consistently to prevent any single VM from starving neighbors.
- **Cluster-level overcommit** – On Kubernetes, overcommit is expressed as the ratio of requested resources to physical capacity. For CPU, the platform allows moderate overcommit for general-purpose VMs, similar to VMware consolidation ratios. For memory, production VMs are configured conservatively (close to 1:1), while non-production namespaces target an approximate 1.65:1 memory overcommit ratio via automatic memory limits and namespace-level resource quotas.

Because VMs run as pods, they benefit from the same scheduling, QoS, and eviction semantics as containers, making resource behavior predictable under load.

Node Maintenance and Non-Disruptive Upgrades

VMware environments commonly use maintenance mode to drain a host of VMs before patching. KubeVirt provides an analogous workflow:

1. Before taking a node out of service, operators cordon and drain it at the Kubernetes level, triggering a node-maintenance workflow that:
    a. Migrates eligible VMs off the node using live migration (leveraging RWX FADA volumes)
    b. Evicts or reschedules container workloads according to their policies
2. Once the node is empty of critical workloads, it can be patched, rebooted, or decommissioned.
3. When it returns, the scheduler and KubeVirt place workloads back where capacity and policies allow, similar to how VMware DRS and maintenance mode interact.

From the VM owner's standpoint, node maintenance becomes a non-event: their VM keeps running on a different node with the same persistent storage underneath.

Storage & Data Platform: Portworx Enterprise + FlashArray

The combination of Portworx Enterprise and Everpure FlashArray is central to delivering enterprise-grade storage for KubeVirt VMs and Kubernetes workloads.

CSI and Dynamic Provisioning

Kubernetes workloads request storage via PersistentVolumeClaims. Portworx implements the CSI interface, dynamically provisioning volumes that are ultimately backed by FlashArray pools.

Storage classes define different performance and protection profiles—similar to VMware storage policies. For KubeVirt, the VM's disks are simply Portworx-backed PVCs mounted into KubeVirt VirtualMachineInstances.

Unified Data Services for VMs and Containers

One of the most significant benefits vs. VMware is that the same data platform is used for both VMs and containers:

- Stateful workloads that migrated from VMware VMs now run either as VMs under KubeVirt or as containerized services.
- In both cases, their data lives on Portworx volumes on FlashArray, simplifying:
    - Operations and monitoring—one storage stack to manage.
    - Data protection—common policies across VM and containerized workloads.
    - Future replatforming—moving from VM-based to container-based stacks doesn't require a storage migration.

Data Protection and Resilience

Portworx Enterprise on FlashArray enables:

- **Snapshots and clones** – As described in Section 4.3, FlashArray provides fast, space-efficient snapshots for backups and environment copies.
- **Backup and DR integration** to meet RPO/RTO targets, matching or exceeding what many VMware environments achieve with vSphere-integrated storage, but in a cloud-native, Kubernetes-aligned way.

Performance: FADA Delivers FlashArray Performance to VMs

A common concern when moving VMs to Kubernetes is storage performance. Traditional hypervisors often introduce an "I/O Blender" effect where many VMs compete for storage controller resources, limiting per-VM throughput.

With the recent addition of **FlashArray Direct Access (FADA)** to Portworx Enterprise, VMs on KubeVirt bypass these bottlenecks entirely. FADA uses NVMe fabric to connect each node directly to the FlashArray backend, delivering performance that rivals—and often exceeds—local SSD storage.

Performance benchmarks

To validate that KubeVirt on bare metal could match or exceed legacy virtualization performance, we conducted stress tests using **fio (Flexible I/O Tester)** with a 3GB dataset and 128KB block sizes on a standard VM:

Metric	Test	IOPS	Bandwidth
Write	Sync Write (fdatasync=1)	3,306	413 MiB/s
Write	Async Write (Buffered)	18,000+	2,370 MiB/s
Read	Random Read	28,200	3,523 MiB/s

Table 3.

Analysis

- **Synchronous writes** (critical for database integrity) sustain over **400 MiB/s**, rivaling local SSD performance. This is a dramatic improvement from the ~150 MB/s often seen with traditional shared storage approaches.
- **Asynchronous writes** saturate the available bandwidth at **2.3 GB/s**, proving the solution can handle burst workloads like code compilation or large database restores.
- **Random reads** deliver over **3.5 GB/s per VM**, effectively eliminating the I/O Blender effect. Each VM gets direct FlashArray performance without contention at a hypervisor storage controller layer.

Why this matters for VMware migrations

Many VMware environments are constrained by shared storage controllers that become bottlenecks under load. With FADA:

- There is no hypervisor storage layer adding latency
- Each VM's I/O goes directly to FlashArray over NVMe fabric
- RWX volumes enable live migration without sacrificing performance
- Storage performance scales with the number of nodes, not a single controller

The result is enterprise storage performance that exceeds what most VMware deployments achieve, delivered through a fully Kubernetes-native stack.

GitOps Operating Model with ArgoCD

Rather than configure clusters and add-ons manually, everything is defined as code and applied declaratively through ArgoCD.

Top-Level ArgoCD Application (App-of-Apps Pattern)

Each Kubernetes cluster is associated with a top-level ArgoCD “app-of-apps” that points at a directory of rendered ArgoCD Applications. Those Applications describe all cluster add-ons and platform components.

This pattern means that:

- All cluster-level add-ons are defined once in Git.
- ArgoCD continuously reconciles those definitions against the live cluster.
- Drift is detected and corrected automatically.

Cluster-Specific Add-ons and Config

Cluster profiles (region, environment, and addon groups) are described in values files that drive manifest rendering. Rendered ArgoCD Applications include, among others:

- **Virtualization stack** – KubeVirt core, VM image import and migration tooling, VM management dashboards, and console proxy components.
- **Data platform** – Portworx Enterprise operators and storage clusters that expose FlashArray-backed volumes (including RWX FADA) for both VMs and containers.
- **Observability** – monitoring and alerting stacks, cluster state exporters, event routing, and log forwarding agents that integrate with the enterprise monitoring and logging backends.
- **Security and governance** – secret management integration, centrally defined RBAC policies with automated reconciliation, and vulnerability scanning connectors parameterized by environment.

These add-ons are versioned, parameterized (for environment, region, and service type), and continuously reconciled by the GitOps controller, ensuring that every cluster converges to the same policy and operations baseline.

GitOps for the Platform, UI for VM Lifecycle

Platform infrastructure (storage, security, observability, KubeVirt control plane) is Git-managed as described above. Individual VM lifecycle operations—provisioning, power management, snapshots, consoles—are handled through the homegrown UI (see Section 4.1), not through GitOps.

This separation keeps the infrastructure layer declarative and auditable while giving VMware administrators a familiar interface for day-to-day VM operations.

Bare-Metal and Cluster Lifecycle

KubeVirt's power on bare metal is only fully realized when the underlying hardware and cluster are equally automated.

Hardware Sizing Guidance

Running VMs on Kubernetes requires careful hardware selection. The following guidance ensures sufficient resources for production workloads:

Worker nodes (VM Hosts)

Component	Minimum Spec	Recommended Spec	Remarks
Server	2U rackmount	2U rackmount	Must accommodate FC/NVMe adapters and NICs
CPU	x64 with 8 cores	x64 with 48+ cores	Plan for 20+ VMs per node
RAM	64 GB	768 GB or more	(Median RAM per VM) × (VMs per node)
Network	2× 10 Gbps	4× 25 Gbps	Pod overlay and VM traffic on data
Storage	2× 16/32 Gbps FC or NVMe-oF	2× 32 Gbps FC or NVMe-oF	Multipath for FlashArray connectivity
Boot	Local SSD	Local NVMe	Typically 250 GB

Table 4.

Control plane nodes

Control plane nodes do not run VM workloads and can be lighter:

Cluster Size	Namespaces	CPU Cores	Memory
10 workers	100	4	8 GB
25 workers	500	4	16 GB
100 workers	1000	8	32 GB
250 workers	2000	16	64 GB

Table 5.

Cluster sizing considerations

- **Minimum 10 worker nodes** – Ensures sufficient distribution for storage failover (VMs cannot run on nodes hosting their storage replicas)
- **Storage topology awareness** – The Stork scheduler ensures pods land on nodes with fast access to their data
- **Everpure FlashArray//C or FlashArray//X** – Both models are compatible with Portworx and suitable for VM workloads

Bare-Metal Provisioning

Bare metal servers are treated as disposable, reprovisionable resources:

- **Inventory as code** – Hosts, roles, and groups are defined in inventory files per cluster, specifying node hostnames and IPs, group membership (control-plane, worker, storage-heavy, etc.), and any node-specific labels or hardware characteristics.
- **Ansible playbooks** – Playbooks handle PXE boot configuration, OS installation (Ubuntu 22.04 LTS), base hardening, and any firmware or BMC configuration required.
- **GitHub Actions pipelines** – CD workflows encapsulate the playbook runs. Triggering a pipeline with a specific inventory for a given cluster results in a repeatable, idempotent node provisioning run.

Network bonding and VLANs

Production deployments typically configure:

Interface	Type	Purpose	Example VLAN
mgmt	Physical 1G (1 NIC)	SSH	Native or VLAN 200
bond	Bond 25G (4 NICs)	Kubernetes API, pod overlay, VM traffic, external access	VLAN 3999
bond.100	Bridge on bond	Master interface for Multus VLANs	VLAN 100

Table 6.

Switchports are configured as trunks: mgmt NICs allow management VLANs; data NICs allow all VM VLANs. This mirrors the familiar VMware pattern of separating management from vMotion/VM traffic.

Multipath configuration for FlashArray

Nodes connecting to Everpure FlashArray require multipath-tools with device-specific settings:

- **Path selector**: `service-time 0` for SCSI, `queue-length 0` for NVMe
- **Path grouping**: `group_by_prio` with ALUA/ANA prioritization
- **Failback**: `immediate` for rapid path recovery
- **No path retry**: `0` (fail fast to upper layers)
- **udev rules**: `Set scheduler=none`, `add_random=0`, `rq_affinity=2` for optimal SSD performance

These settings ensure Portworx sees stable multipath devices and can manage LUN assignments during node failures or upgrades.

Cluster Lifecycle with Kubespray

Once nodes are provisioned, Kubespray manages:

- **Initial cluster creation** – Control-plane nodes, etcd layout, CNI plugin (Cilium), core add-ons.
- **Upgrades** – Kubernetes version upgrades tested in lower environments, then applied consistently across clusters. Kubespray supports rolling upgrades that drain and cordon nodes one at a time.
- **Configuration drift prevention** – Consistent cluster configurations across environments by reusing Kubespray definitions and inventory.

Portworx behavior during node upgrades

When nodes are drained for upgrades (as described in Section 4.5), Portworx handles storage gracefully by reassigning LUNs to remaining nodes rather than rebuilding data—eliminating the I/O storm typical of distributed storage during node replacement.

Outcome

Any cluster can be rebuilt from scratch using only Git (inventory and Kubespray config) and GitHub Actions. Teams are no longer dependent on "golden images" or snowflake servers.

Migration Strategy and Automation

Migrating from VMware to KubeVirt is both a process and a technology problem. On this platform, migrations are driven by Forklift – an open-source migration toolkit under the KubeVirt/Konveyor umbrella — that orchestrates vCenter, KubeVirt, storage, DNS, and the VM lifecycle UI end-to-end.

How VM Disks Move: Forklift with FlashArray Storage Offload

Forklift is the primary migration engine for moving VM workloads from VMware to KubeVirt at scale. It supports VMware, oVirt, OpenStack, and OVA sources, and provides warm migration via **Change Block Tracking (CBT)**, automated guest conversion via **virt-v2v**, and **FlashArray XCOPY** storage offload. Under the hood, Forklift builds on two proven KubeVirt primitives: **CDI (Containerized Data Importer)** and **VDDK (VMware Virtual Disk Development Kit)**.

CDI (Containerized Data Importer)

CDI is a KubeVirt companion project that handles importing, cloning, and uploading VM disk images into Kubernetes. It introduces the **DataVolume** custom resource, which declaratively describes where disk data should come from and where it should be stored.

When a DataVolume is created, CDI:

1. Spins up an importer pod that reads the source disk image
2. Streams the data into a PersistentVolumeClaim (PVC)
3. Reports progress and completion status on the DataVolume object

VDDK (VMware Virtual Disk Development Kit)

VDDK is VMware's official library for reading and writing VMDK disk files. Forklift integrates with VDDK to import disks directly from vCenter datastores:

- The importer pod uses a VDDK init container that provides the proprietary VMware libraries
- Forklift connects to vCenter, authenticates, and reads the VMDK via the VDDK API
- Disk data streams directly from the VMware datastore into the target PVC—no intermediate staging required

What Forklift Orchestrates

Forklift automates the end-to-end migration workflow, layering warm migration via CBT and automated guest conversion on top of the CDI/VDDK import pipeline:

1. **Pre-flight** – Gather VM specs from vCenter (CPU, memory, disk sizes, UUIDs)
2. **Guest conversion** – virt-v2v installs virtio/QEMU-KVM drivers, enables cloud-init, and configures the serial console — no SSH access to the guest required
3. **Cutover** – Power off the VM for a consistent disk state
4. **Disk transfer** – Forklift creates a DataVolume per disk; XCOPY storage offload (if FlashArray backs both source and target)

or CDI/VDDK streaming completes the transfer into RWX PVCs

5. **VirtualMachine creation** – Create the KubeVirt VirtualMachine referencing those DataVolumes
6. **Validation** – Boot the VM, verify connectivity, update DNS

The result is a VM running on KubeVirt with its disks stored as RWX PVCs on Portworx/FlashArray—fully Kubernetes-native, ready for live migration and standard data protection workflows.

FlashArray XCOPY Storage Offload

For environments where Everpure FlashArray backs both the VMware datastores and the target Portworx volumes, Forklift enables a storage-level offload that dramatically accelerates migrations using FlashArray's native XCOPY capability. Instead of streaming disk data through the host, the storage array performs the copy operation directly:

Everpure Contributions to Forklift

Everpure has contributed storage-offload capabilities to the Forklift project, enabling array-level operations that complement the standard CDI/VDDK pipeline:

Feature	Description
XCOPY Populator	FlashArray provider for storage-level copy operations
vVol Migration	Support for VMware Virtual Volumes backed by FlashArray
FC Host Matching	Fibre Channel WWN lookup for proper LUN mapping
Token Authentication	Secure API authentication for FlashArray management

Table 7.

Benefits of storage offload:

- **Faster migrations** – Array-to-array copies bypass host CPU and network bottlenecks
- **Reduced resource consumption** – Minimal impact on source VMware hosts during migration
- **Consistency** – Storage-level snapshots provide crash-consistent disk states
- **Scalability** – Migrate many VMs simultaneously without saturating network bandwidth

When to Use XCOPY Offload:

Forklift with FlashArray XCOPY storage offload is the preferred migration path when a single FlashArray backs both the VMware datastores and the target Portworx volumes. It is particularly well-suited for:

- Large VM disks (100+ GB) where host-based transfer is slow
- High-density migration waves where network bandwidth is constrained
- Environments already standardized on Everpure FlashArray

Migration Challenges and Lessons Learned

During the migration from VMware to KubeVirt, several real-world challenges emerged that required architectural decisions and targeted solutions. These lessons are documented here to help teams anticipate and plan for similar issues.

Challenge: IP Address Stability During Live Migration

With pod masquerade networking, VMs receive dynamic IPs from the pod network. During live migration, the VM's pod is recreated on the target node with a new IP. Users reported losing SSH connections and service disruptions when VMs migrated between nodes, because clients and monitoring systems were connected to the original pod IP.

Solution

To restore IP stability, we introduced secondary NICs using Multus CNI. Each VM that required a stable, reachable IP was configured with a Multus secondary interface attached to a bridge network on a dedicated VLAN. The secondary interface receives a stable IP that persists across live migrations. The custom DNS operator was updated to register the secondary interface IP (rather than the pod IP) as the VM's DNS record, ensuring that SSH connections, monitoring agents, and application clients connect through the stable address. The primary masquerade interface continues to handle Kubernetes-native service traffic and network policy enforcement via Cilium.

Challenge: Minimizing Downtime for Critical VM

Some business-critical VMs had very tight downtime windows that standard CDI/VDDK streaming could not meet, particularly for VMs with large disks (500+ GB). Streaming data through the host network introduced unacceptable cutover times during migration waves.

Solution

For these critical workloads, we leveraged Forklift's FlashArray XCOPY storage offload to perform array-level disk copies that bypass the host entirely. Additionally, we configured FlashArray-to-FlashArray replication between the source array (backing VMware datastores) and the target array (backing Portworx volumes). This replication pre-staged the bulk of the data before the cutover window, reducing the final sync to only the changed blocks. The combination of XCOPY offload and FA replication brought cutover times for large, critical VMs from hours down to minutes.

Challenge: Static-to-Dynamic IP Transition at Scale

The majority of VMs in the VMware environment were configured with static IP addresses via netplan or /etc/network/interfaces. After migration to KubeVirt with pod masquerade networking, these VMs could not obtain network connectivity automatically — they would boot with a hardcoded static IP that did not match the pod network, leaving them unreachable without manual intervention.

Solution

We developed cloud-init scripts that are injected during the Forklift migration process (as part of virt-v2v guest conversion). These scripts reconfigure the guest networking from static to DHCP-based dynamic assignment on first boot in the KubeVirt environment. The cloud-init userdata overwrites the existing netplan configuration with a DHCP-enabled config, ensuring the VM picks up its pod network IP automatically. For VMs requiring stable reachability, this was combined with the Multus secondary NIC approach described above. This eliminated the need for manual, per-VM network reconfiguration and allowed migration waves to proceed without hands-on guest OS remediation.

These challenges highlight that while the core migration tooling (Forklift, CDI/VDDK, XCOPY) handles the bulk of the work, real-world migrations require thoughtful handling of networking transitions, downtime constraints, and guest OS configuration. Planning for these scenarios early — during the Discovery and PoC phases — significantly reduces risk during production migration waves.

Discovery and Assessment

A thorough discovery phase prevents surprises during migration:

Inventory workloads on VMware

- **Compute profile** – vCPU count, memory allocation, CPU ready times, memory ballooning
- **Storage footprint** – Provisioned vs. actual usage, thin vs. thick provisioning, datastore dependencies
- **Network dependencies** – VLANs, port groups, NSX segments, firewall rules, load balancer VIPs
- **OS and application stack** – OS version, installed agents, licensing constraints, kernel dependencies
- **Integration points** – External dependencies (databases, APIs), backup agents, monitoring hooks

Classify workloads

Category	Characteristics	Migration Path
Lift-and-Shift	Standard x86 VMs, no special hardware, Linux/Windows	Forklift migration
Replatform	Stateless apps, microservices candidates	Containerize over time
Stay on VMware	GPU passthrough, specialized drivers, vendor appliances	Keep or evaluate alternatives
Retire	Unused VMs, obsolete systems	Decommission

Table 8.

Success criteria for discovery

- Complete inventory with resource utilization data for 30+ days
- Identified application owners and stakeholders
- Documented network topology and firewall dependencies
- Risk assessment for each workload category

Design and PoC

Stand up a PoC KubeVirt cluster in a non-production environment that mirrors the target production architecture:

Infrastructure validation

- Deploy the full stack: bare-metal provisioning, Kubespray, Cilium, Portworx, KubeVirt, ArgoCD
- Validate storage performance with fio benchmarks (target: 2+ GB/s throughput with FADA)
- Test network connectivity: BGP peering, LoadBalancer VIPs, DNS integration
- Verify multipath failover by simulating FC/NVMe path failures

Workload migration testing

Migrate a small set of representative workloads, validating:

- **Performance** – I/O latency, CPU scheduling, memory allocation
- **HA behavior** – Live migration under load, node failure simulation, storage failover
- **Operational workflows** – Provisioning via UI, snapshot/revert, console access, backup/restore

PoC exit criteria

- All representative workloads running successfully for 2+ weeks
- Performance metrics within 10% of VMware baseline (often better with FADA)
- Documented runbooks for common operations
- Stakeholder sign-off on operational readiness

Pilot and Scale-Out

Establish a pilot cluster (production-like but not business-critical):

Coexistence with VMware

- VMs can run on both platforms during transition
- DNS operator updates records as VMs move, ensuring transparent failover
- Monitoring spans both environments with unified dashboards

Gradual migration waves

Wave	Scope	Validation Focus
Wave 1	Dev/test workloads	Operational workflows, team training
Wave 2	Non-critical production	Monitoring, alerting, incident response
Wave 3	Business-critical systems	HA, disaster recovery, compliance
Wave 4	Remaining workloads	Scale, optimization, consolidation

Table 9.

Operational validation

- **Monitoring and alerting** – Prometheus alerts fire correctly; Grafana dashboards reflect cluster health
- **Incident response** – On-call runbooks tested for common failure scenarios
- **Backup and recovery** – Snapshot schedules in place; restore procedures validated
- **Change management** – GitOps promotion workflows tested across environments

Decommission and Optimize

As confidence grows, shift more workloads from VMware to KubeVirt clusters following this architecture.

Optimization areas

- **Node density** – Tune memory overcommit ratios (e.g., 1.65:1 for non-prod); consolidate underutilized nodes
- **Storage efficiency** – Right-size PVCs, leverage FlashArray deduplication, optimize replication factors
- **GitOps maturity** – Automate more workflows; reduce manual intervention
- **Cost tracking** – Compare operational costs: licensing, power, cooling, labor

Decommissioning checklist

1. Verify all VMs migrated and validated on KubeVirt
2. Confirm backup retention policies honored
3. Update CMDB and asset inventory
4. Reclaim vCenter resources (hosts, storage, licenses)
5. Archive VMware configurations for audit purposes

Long-term outcome

VMware becomes one environment among several, rather than the central dependency. Many organizations significantly reduce their VMware footprint or decommission it entirely for specific workload classes—eliminating licensing costs while gaining operational consistency across VMs and containers.

Operations, Security, and Compliance

Monitoring and Observability

Prometheus and Grafana stack

The kube-prometheus-stack provides comprehensive monitoring:

- **Cluster health dashboards** – Node status, pod scheduling, resource utilization
- **KubeVirt metrics** – VM count, vCPU usage, memory allocation, live migration status
- **Storage metrics** – Portworx volume health, I/O throughput, latency histograms
- **Network metrics** – Cilium flow logs, Hubble visibility, BGP peering status

Dashboards are parameterized per cluster, enabling consistent views across environments.

Logging and metrics collection

Elastic Agent (or similar) collects logs and metrics:

- **Node logs** – System journals, kernel messages, storage multipath events
- **Container logs** – KubeVirt, Portworx, Cilium, and workload containers
- **VM console logs** – Serial console output captured for troubleshooting
- **Tagging** – Events tagged by environment, cluster, and namespace for filtering

Snapshot Management with CSI Snapshot Controller

The CSI Snapshot Controller abstracts volume snapshots into a generic Kubernetes resource:

VolumeSnapshotClass configuration

A VolumeSnapshotClass points to the Portworx CSI driver:

```
none
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: px-csi-snapclass
driver: pxd.portworx.com
deletionPolicy: Delete
```

Snapshot workflow

1. User or automation creates a VolumeSnapshot referencing a VM's PVC
2. CSI Snapshot Controller invokes Portworx to create a point-in-time snapshot
3. Portworx leverages FlashArray's native snapshot capabilities—instantaneous and space-efficient
4. Snapshots can be used to restore PVCs or clone VMs

This provides a consistent snapshot mechanism across VMs and containers, integrated with the VM lifecycle UI.

Security

RBAC and access control

- **Namespace isolation** – Each team or project gets a namespace with RBAC boundaries
- **Role definitions in Git** – Roles and RoleBindings managed via ArgoCD, ensuring consistency
- **Service accounts** – Workloads use scoped service accounts with least-privilege access

Secrets management

- **HashiCorp Vault integration** – Secrets injected at runtime via Vault Agent or External Secrets Operator
- **Kubernetes Secrets encryption** – etcd encryption enabled for secrets at rest
- **Rotation policies** – Automated credential rotation for service accounts and certificates

Network security

- **Cilium Network Policies** – L3/L4 and L7 policies enforced at the eBPF layer
- **Identity-based security** – Policies based on pod identity rather than IP addresses
- **Encryption in transit** – WireGuard or IPsec for inter-node traffic where required

Security scanning

- Vulnerability scanning deployed per cluster, with configuration driven from Git
- Image scanning in CI/CD pipelines before deployment
- Runtime security monitoring for anomaly detection

Compliance and Audit

Because every change—cluster configuration, add-ons, policies—flows through Git and ArgoCD:

Audit trail

- **Git history as audit log** – Every configuration change has an author, timestamp, and approval record
- **ArgoCD sync history** – When changes were applied, by whom, and the resulting state
- **Prometheus metrics** – Historical data for capacity planning and SLA reporting

Compliance benefits

- **Rollback** – As simple as reverting a Git commit; ArgoCD syncs to the previous state
- **Reproducibility** – Any cluster can be rebuilt from Git; no undocumented configurations
- **Promotion workflows** – Changes tested in dev → staging → prod with gates and approvals

Regulatory alignment

This GitOps model aligns with compliance frameworks that require:

- Change management controls (SOC 2, ISO 27001)
- Audit trails and separation of duties (PCI-DSS, HIPAA)
- Disaster recovery and backup verification (NIST, FedRAMP)

Summary and Next Steps

VMware has earned its place as a trusted backbone for enterprise workloads, but the future of infrastructure is open, cloud-native, and automation-first. By deploying KubeVirt on bare metal Kubernetes, backed by Portworx Enterprise and Everpure FlashArray, migrating workloads with Forklift and FlashArray XCOPY storage offload, and operating via GitOps and automated bare-metal lifecycle, you can:

- Reduce costs associated with proprietary virtualization stacks.
- Unify operations across VMs and containers on a single, modern platform.
- Avoid vendor lock-in while preserving enterprise-grade resilience and data protection.
- Accelerate modernization, making it easier to replatform workloads over time.

For a VMware customer, the practical next steps are:

1. Run a discovery and readiness assessment of VMware workloads.
2. Stand up a PoC environment mirroring the architecture described here.
3. Use GitOps and automation from day one to ensure repeatability.
4. Carefully migrate and operate a pilot set of workloads before scaling out.