REFERENCE ARCHITECTURE: Oct 2026



Redis Enterprise on VMware Cloud Foundation 9.1 with vSphere Kubernetes Service

Reference Architecture

vmware by Broadcom

Redis

Table of Contents

Executive Summary 4
Audience 4
Technology Overview 4
VMware Cloud Foundation (VCF) 5
vSAN Express Storage Architecture (ESA) 5
vSphere Kubernetes Service 6
Intel Xeon Platforms 6
Redis Enterprise on VCF with VKS 7
Redis Enterprise deployment through Redis Enterprise operator 7
Core Kubernetes resources 7
Disk Persistence in Redis 7
Active-Active (multi-site) databases 8
Solution Configuration 8
Architecture Diagrams 8
Hardware Resources 10
Software Resources 10
Kubernetes Environment 11
VKS Cluster Configuration 11
Networking Configuration 12
Storage Configuration 12
Deploy Redis Enterprise on VKS 12
End-to-End Manifest Deployment Sequence 13
Solution Validation 14
Validation Approach and Testing Tools 14
Testing Methodology 15
Tuning Context 15
Performance Tuning and Operational Validation 15
Key Findings and Architecture Decisions 15
VCF v9.0.2 with vSAN ESA (Deduplication Disabled) vs. v9.1.1 with vSAN ESA Deduplication 16
TuneD Profile and Antrea Hybrid Encapsulation Optimization 18
VKS Worker VM Sizing 19
Persistence Scenarios: Caching Only vs. RDB vs. AOF 20
90-Minute RDB Endurance Benchmark 20
Shard Density Analysis 21
Low-Latency Workload SLA Validation and Scale-Out 21
Multi-Tenant Fair Sharing 22
Online Shard Expansion 22
High Availability, Failover & Resiliency Validation 25

Best Practices....27
Compute & NUMA Architecture 27
Storage & vSAN ESA Design 27
Networking & Antrea CNI 27
Redis Enterprise Configuration & Scaling 27
Cross-Layer Bottleneck Diagnostics 28
Conclusion....29
Appendix....30

Executive Summary

Redis Enterprise on VMware Cloud Foundation (VCF) with vSphere Kubernetes Service (VKS) provides a production-grade foundation for mission-critical real-time applications. VCF/VKS standardizes the infrastructure; Redis Enterprise turns it into a reliable, scalable, Kubernetes-native real-time data service.

Redis Enterprise provides performance, high availability, persistence, security, automated failover, and operational capabilities required to run Redis as a production data platform—not only as a cache. For site-level resilience, Active-Active replicates data across clusters on separate VCF environments with automatic conflict resolution. The Redis Enterprise Operator enables platform teams to provision, scale, upgrade, and manage Redis databases using Kubernetes-native resources and GitOps workflows.

VMware Cloud Foundation (VCF) offers a unified, software-defined infrastructure platform that extends the public cloud operating model to on-premises and hybrid environments. Standardizing Redis deployments on VCF helps enterprises harmonize developer needs with IT operations. VCF ensures extreme performance through NVMe-optimized storage (vSAN ESA), provides zero-trust security via software-defined networking (NSX), and delivers the flexibility to consume infrastructure exactly as required. This approach dramatically lowers the Total Cost of Ownership (TCO) while accelerating time-to-market.

This reference architecture provides validated deployment guidance for worker sizing, OS and network tuning, persistence, shard placement, and operational resilience, helping architects build and scale Redis Enterprise on VCF with VKS to meet application performance, capacity, and availability requirements.

The architecture combines low-latency reads and writes, high throughput, multi-tenant database consolidation, and online expansion to support growing application demands. Its principal advantages are:

- **Efficient Scale-Out**: Right-sized worker VMs and balanced shard placement improve host resource utilization. Horizontal scaling expands capacity while keeping worker sizes aligned with the platform's NUMA core capacity.
- **Integrated Performance and Persistence**: TuneD and Antrea Hybrid Encapsulation improve throughput and tail latency. vSAN ESA provides persistent storage for RDB and AOF, with data reduction improving storage efficiency.
- **Multi-Tenant Efficiency**: Consolidate multiple Redis Enterprise databases on a shared cluster while maintaining database-specific configurations. Balanced shard placement supports efficient resource use and fair sharing among tenants with comparable workloads.
- **Operational Resilience**: Replication, automated failover, and configurable persistence support service recovery. Kubernetes-native lifecycle management integrates database operations into established platform workflows.
- **Online Capacity Growth**: Expand database shards while continuing to serve requests. Declarative scaling enables platform teams to increase capacity as application demand grows.

Audience

This reference architecture is tailored for technical professionals responsible for architecting, deploying, securing, and operating enterprise in-memory data infrastructure on VMware Cloud Foundation:

- **Enterprise Infrastructure & Cloud Architects**: Designing resilient, scalable VMware Cloud Foundation and Tanzu/VKS platforms optimized for low-latency databases.
- **Principal Database Administrators & Engineers**: Sizing Redis Enterprise clusters, configuring persistence policies, tuning proxy concurrency, and managing shard placement.
- **VMware / VCF System Administrators**: Configuring worker VM resources, vSAN ESA storage policies, and physical-host placement.
- **Site Reliability Engineers (SREs) & DevOps Architects**: Automating Kubernetes database deployments, managing GitOps lifecycle operations, and planning failover and persistence recovery.

Technology Overview

Solution technology components are listed below:

- VMware Cloud Foundation (VCF)
  - vSAN Express Storage Architecture (ESA)
  - vSphere Kubernetes Service (VKS)
- Intel Xeon Platforms
- Redis Enterprise

VMware Cloud Foundation (VCF)

VMware Cloud Foundation (VCF) is a premier private cloud platform designed to deliver public cloud scale and agility alongside on-premises security, resilience, and performance. It accelerates digital innovation by providing a consistent, unified cloud experience across customer-managed data centers, edge locations, and cloud endpoints. Engineered to support both traditional infrastructure and modern workloads—such as containers and AI/ML—VCF helps bring modern applications to market faster while delivering measurable total cost of ownership (TCO) savings.

To achieve this, VCF delivers a modernized, full-stack private cloud by integrating vSphere compute, vSAN Express Storage Architecture (ESA), and NSX networking into a single, automated software-defined data center (SDDC). This cohesive architecture provides the extreme I/O performance and low-latency network paths essential for high-performance distributed workloads and data-intensive applications.

VCF combines vSphere compute, vSAN ESA persistent storage, NSX networking, fleet management, and native Kubernetes integration to provision and operate the Redis Enterprise platform described in this reference architecture.

VCF 9.1 builds on the unified platform introduced in VCF 9.0 with capabilities that directly benefit data-intensive workloads such as Redis Enterprise. vSAN ESA deduplication and compression are extended across more cluster types and workload profiles, now including support for data-at-rest encryption, lowering the raw storage footprint per logical terabyte. Enhanced NVMe Memory Tiering keeps active pages in DRAM while moving cold pages to local NVMe devices, expanding effective memory capacity without additional DRAM and adding software-based mirroring. For Kubernetes, VCF 9.1 adds multi-network support for VKS cluster nodes, allowing application, storage, and management traffic to be isolated on separate vNICs. Operationally, live patching now extends to TPM-enabled ESX hosts and vCenter supports Quick Patching, so security fixes can be applied without workload disruption, while fleet management functions are consolidated into a single VCF Management Services layer. This reference architecture was validated on VCF 9.1.1.

vSAN Express Storage Architecture (ESA)

vSAN ESA is an all-NVMe, single-tier storage architecture designed to maximize high-performance flash device efficiency. vSAN ESA employs an advanced, inline data reduction mechanism that applies deduplication and compression globally across the storage cluster. Engineered specifically for high-performance NVMe flash storage, this architecture identifies and eliminates redundant data blocks and compresses the remaining payloads before writing them to physical media.

vSAN ESA provides high-performance persistent storage for Append-Only File (AOF) and RDB snapshots and integrated compression/deduplication. Persistent volume claims are provisioned via the vSAN ESA CSI driver configured with the vSAN ESA Auto RAID policy and late volume binding (WaitForFirstConsumer), allowing vSAN ESA to dynamically select the optimal erasure-coding or mirroring resilience layout based on cluster topology while maximizing storage efficiency.

vSAN ESA in VCF 9.1 adds several capabilities relevant to persistent Redis workloads. Global Deduplication, previously offered only under limited availability, is now generally available and can be enabled directly in the UI. It runs as a cluster-wide, post-process operation in the background, so it does not sit in the write path, which protects application performance. It now supports clusters of up to 64 hosts and is compatible with data-at-rest encryption. Compression has also been improved, delivering greater space savings than previous releases without affecting performance. The Auto-RAID policy used in this reference architecture now automatically reconfigures associated objects when the cluster size changes or the cluster is stretched or unstretched, removing the need to manually adjust and reapply storage policies. Additional 9.1 enhancements include native S3-compatible object storage, a new Effective Capacity view for simpler capacity planning, and the ability for vSAN storage clusters to mount both vSAN OSA and vSAN ESA datastores.

vSphere Kubernetes Service

VMware Cloud Foundation with vSphere Kubernetes Service (VKS) provides an enterprise-grade Kubernetes runtime built directly into VMware Cloud Foundation (VCF). With CNCF certified Kubernetes, VKS enables platform engineers to deploy and manage Kubernetes clusters while leveraging a comprehensive set of cloud services in VCF. Cloud admins benefit from the support for N-2 Kubernetes versions, enterprise grade security, and simplified lifecycle management for modern apps adoption.

Key benefits include:

- **Built-in Governance and Security**: VKS as part of VMware Cloud Foundation, provides centralized policy enforcement, role-based access control, and network micro-segmentation through NSX, ensuring Kubernetes workloads meet enterprise security standards.
- **Consistent Infrastructure Management**: VKS leverages the same vSphere infrastructure that enterprises already trust, eliminating the operational complexity of managing separate Kubernetes and VM stacks.
- **Unified Networking and Storage**: VKS inherits vSphere's mature networking and storage capabilities, simplifying connectivity between cloud-native applications and existing enterprise systems.
- **Operational Familiarity**: IT teams can manage Kubernetes clusters using familiar vSphere tools and workflows, reducing the learning curve and operational risk associated with adopting container platforms.

By building on VMware Cloud Foundation, organizations establish a standardized, supportable Kubernetes foundation that bridges cloud-native development practices with enterprise operational requirements.

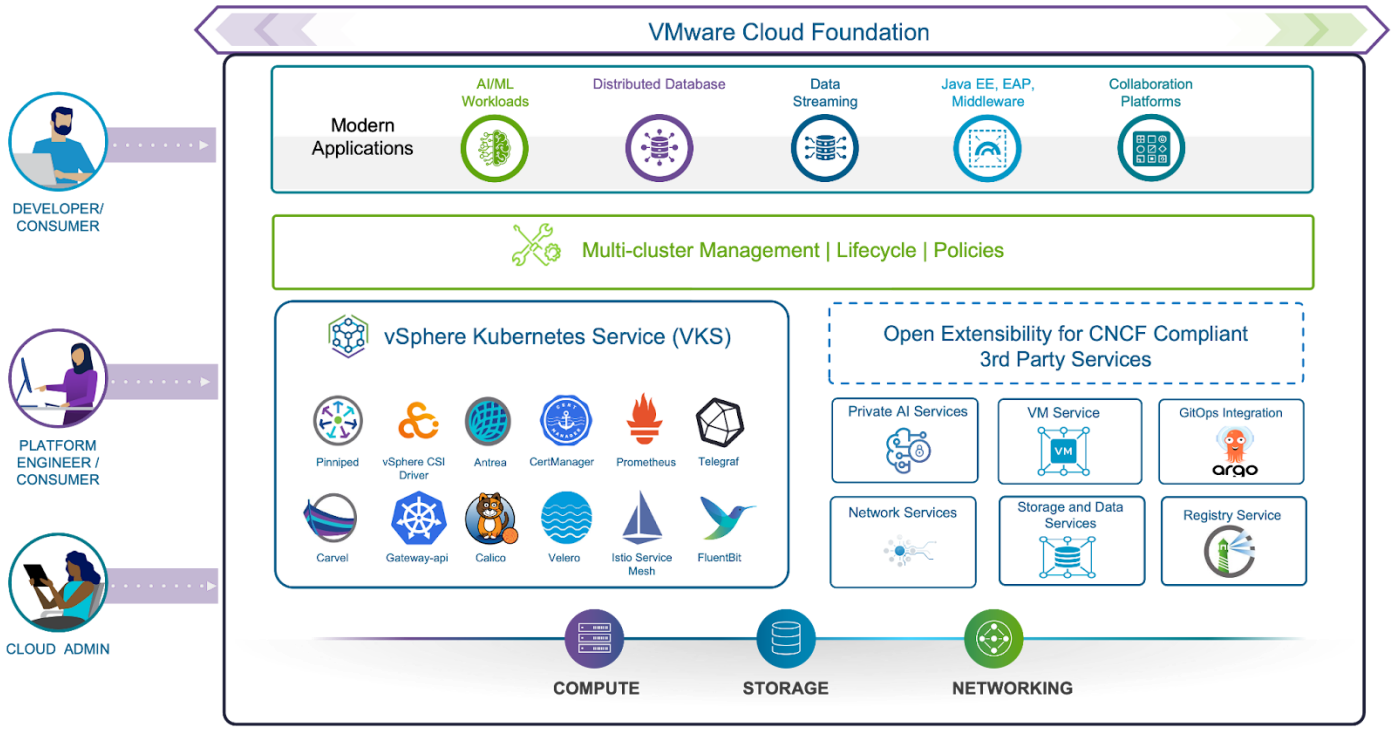


Figure 1: VMware Cloud Foundation with vSphere Kubernetes Service

Intel Xeon Platforms

Intel® Xeon platform provides built-in hardware acceleration, security, and AI.

- Intel® QuickAssist Technology (QAT) accelerates encrypted vMotion in VMware Cloud Foundation by offloading encryption from CPU cores, reducing migration overhead and enabling faster workload mobility without impacting application performance.
- Intel® AMX and VMware Cloud Foundation deliver cost-effective, CPU-based AI at enterprise scale, allowing organizations to run generative AI workloads securely and efficiently on their existing infrastructure.

- Intel® Trust Domain Extensions (TDX) and VMware Cloud Foundation enable confidential VMs with hardware-isolated memory protection, allowing organizations to securely run sensitive and regulated workloads on shared cloud infrastructure without compromising operational simplicity.

Together, they deliver a faster, more secure, and more cost-efficient private cloud platform for modern enterprise workloads.

Redis Enterprise on VCF with VKS

Redis Enterprise on VMware Cloud Foundation combines Redis' real-time performance with the operational consistency of VCF and Kubernetes. Organizations can run Redis as a resilient production data platform—not only as a cache—for operational databases, application state, streams, search, vectors, and real-time analytics.

When Redis becomes mission-critical, Redis Enterprise provides the production capabilities required for enterprise operations, including automatic failover, high availability, Active-Active geo-distribution, enterprise security, support, and simplified cluster management.

This model allows infrastructure teams to standardize platform operations on VCF and VKS, while application teams consume Redis databases through Kubernetes-native APIs. Redis therefore becomes a first-class data service within the enterprise Kubernetes environment, coexisting with application, integration, and observability workloads.

Redis Enterprise deployment through Redis Enterprise operator

Redis Enterprise for Kubernetes brings Redis Enterprise into Kubernetes environments through the Redis Enterprise Operator. Platform teams can deploy, scale, and manage Redis Enterprise clusters and databases using native Kubernetes resources and workflows. The Operator provides custom resource definitions (CRDs) for Redis Enterprise clusters (REC) and Redis Enterprise databases (REDB), enabling Kubernetes-native and GitOps-based operations.

Redis Enterprise for Kubernetes is designed to run on CNCF-conformant Kubernetes platforms. CNCF software conformance promotes API consistency and interoperability across Kubernetes distributions.

Core Kubernetes resources

- **Redis Enterprise Cluster (REC)**: The top-level custom resource defining the cluster topology, licensing, node count, and persistent volume specifications for the cluster nodes.
- **Redis Enterprise Database (REDB)**: Defines individual tenant databases, including memory quota, shard count, replication, persistence policy (RDB/AOF), and client connection limits.
- **Redis Enterprise Operator**: Watches REC and REDB resources and reconciles them into the corresponding Redis Enterprise cluster and database state. High availability is achieved through the configured cluster topology, replicas, persistence, and failover policies.
- **Dynamic Multi-Cluster Proxy (DMC Proxy)**: A high-performance, multi-threaded proxy layer that sits in front of database shards. The DMC proxy routes client commands directly to the appropriate primary shard, provides connection multiplexing, and supports zero-downtime cluster topology changes.

Disk Persistence in Redis

In production environments, disk persistence is a critical architectural consideration. Redis provides two primary persistence mechanisms:

- **Redis Database (RDB)**: Captures point-in-time snapshots of the dataset.
- **Append Only File (AOF)**: Maintains a continuous log of every Redis operation.

RDB Tradeoffs: RDB creates compact snapshots optimized for routine backup operations. This backup process has a minimal impact on overall Redis performance. Furthermore, in disaster recovery scenarios, RDB allows for faster boot times than AOF due to its smaller file size. However, because RDB relies on point-in-time snapshots, any data modified or created after the most recent snapshot will be permanently lost in the event of a system failure.

AOF Tradeoffs: AOF logs every operation, providing significantly greater durability than RDB. It can be configured to synchronize (fsync) data on every second or after every write operation. Redis Enterprise offers two AOF fsync policies: every second or every write. The default, every second, yields excellent write performance because the synchronization runs on a background thread. In the event of an outage, AOF restores the dataset by replaying

the operation log. To mitigate unbounded file growth, Redis can safely and automatically rewrite the AOF in the background when it becomes too large.

Active-Active (multi-site) databases

For disaster recovery and geo-distribution, Redis Enterprise Active-Active links databases on two or more Redis Enterprise clusters into a single logical database. Every site accepts reads and writes locally, so applications keep local latency. Changes replicate asynchronously between sites, and conflicts are resolved automatically with conflict-free replicated data types (CRDTs). If a site is lost, applications reconnect to a surviving site. There is no failover event or promotion step to run.

Validation scope. This reference architecture validates a single Redis Enterprise cluster on VCF with VKS. Active-Active was not tested here. The description above is general product guidance, and a multi-site deployment should be validated against its own latency, networking and recovery requirements.

Solution Configuration

This section introduces the resources and configurations:

- Architecture Diagrams
- Hardware Resources
- Software Resources
- Kubernetes Environment
- VKS Cluster Configuration
- Networking Configuration
- Storage Configuration

Architecture Diagrams

Figure 2 shows Redis Enterprise on VKS within the VMware Cloud Foundation architecture. Dedicated Redis and benchmark worker pools run on separate groups of three physical ESXi hosts in the six-host cluster. The baseline topology places three Redis worker VMs on one host group and three client worker VMs on the other, with one worker VM per host. The scale-out topology distributes five Redis worker VMs across the Redis host group and six client worker VMs across the benchmark host group. This separation ensures that client-to-database benchmark traffic traverses the physical 25 GbE network.

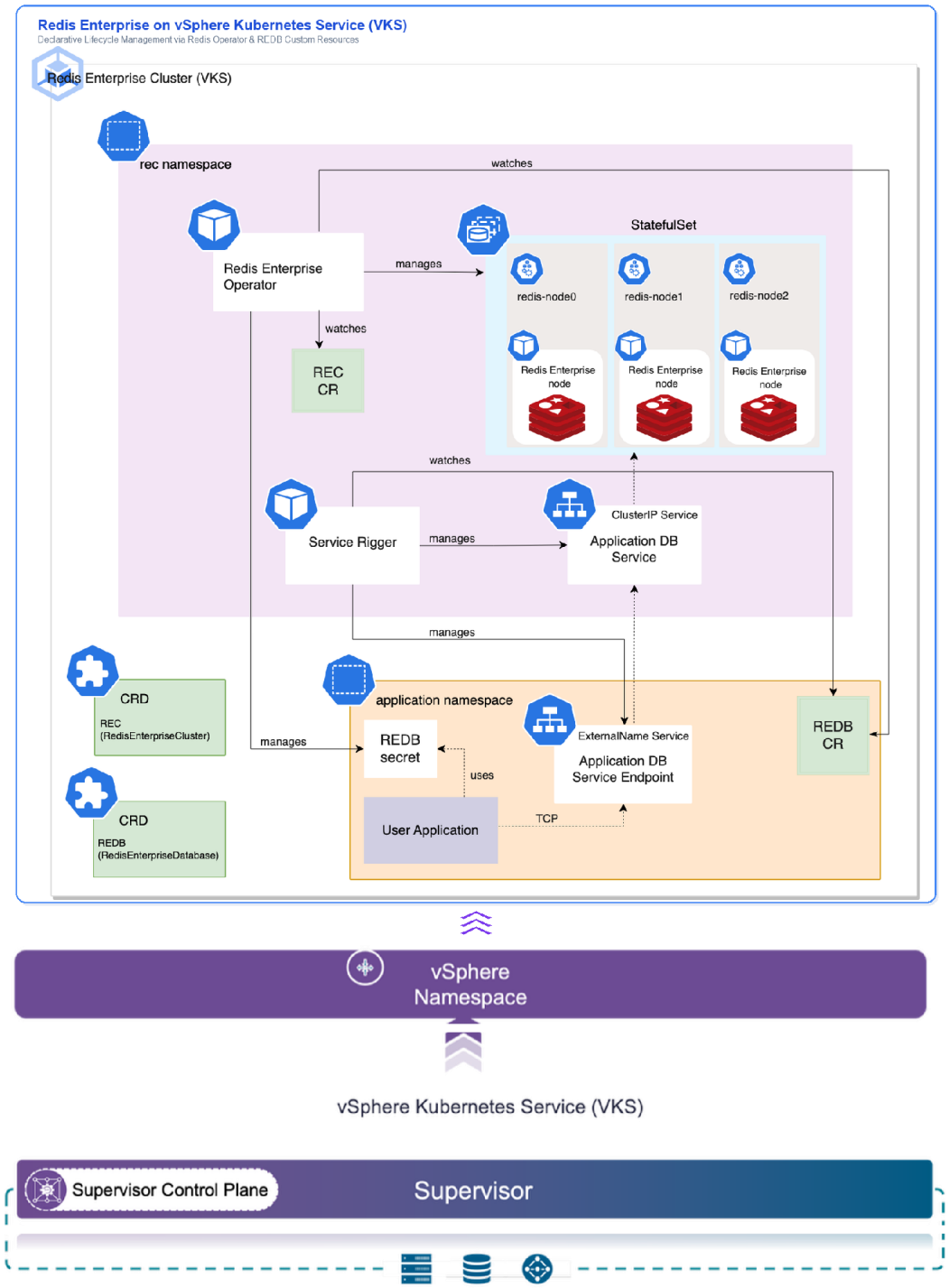


Figure 2: Cloud-Native Redis Enterprise Solutions Architecture on VMware Cloud Foundation and VKS

The following screenshot displays the live cluster shard placement across the worker nodes, illustrating the symmetrical distribution of primary and replica instances.

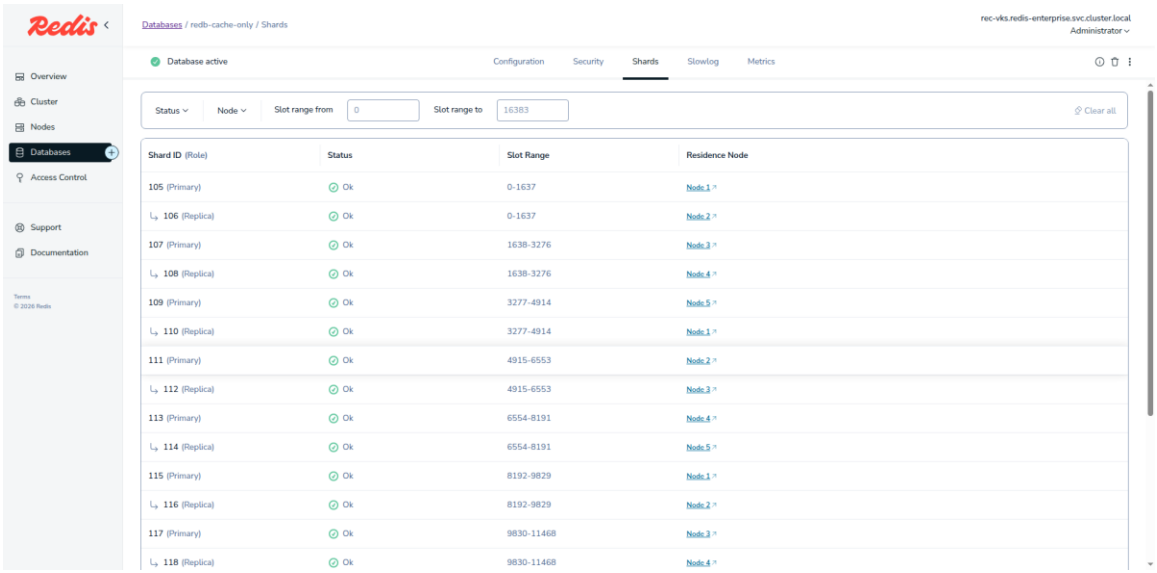


Figure 3: Cloud-Native Redis Enterprise Multi-Shard Placement and Primary/Replica Distribution

For multi-zone resilience, the VKS worker pool can span three Availability Zones. Redis Enterprise's rack-zone awareness (configured on the REC through rackAwarenessNodeLabel, using the Kubernetes zone node label) ensures primary shards and their corresponding Replica shards are placed in different zones. Kubernetes podAntiAffinity and topologySpreadConstraints complement this by spreading the cluster pods themselves across nodes and zones.

For site-level disaster recovery, the same architecture can be deployed on two or more VCF/VKS environments and joined with Active-Active. Each site hosts its own REC, and databases replicate between sites through Kubernetes ingress or route endpoints. This extends the zone-level resilience described above to full site loss.

Hardware Resources

The solution was validated on a cluster of six identically configured Dell PowerEdge R770 servers. Each ESXi host is equipped with Intel Xeon 6740P processors, 512 GB of RAM, two Broadcom quad-port 25 GbE network adapters, and eight 3.84 TB NVMe drives that provide capacity for vSAN ESA. Table 1 lists the per-host hardware configuration.

Table 1: Hardware Configuration

PROPERTIES	SPECIFICATION
Server model name	PowerEdge R770
CPU	Intel(R) Xeon(R) 6740P, 2 sockets × 2 dies, 24 cores per die, with SNC2 enabled
RAM	512GB
Network adapter	2x Broadcom Adv. Quad 25Gb Ethernet
Storage adapter	BOSS-N1 DC-MHS
Disks	8x Dell DC NVMe CD8P E3.S 3.84TB on each host

Software Resources

Table 2: Software Configuration

COMPONENT	VERSION	NOTES
VCF	9.1.1/9.0.2	Complete SDDC software-defined cloud platform
Supervisor	v1.34.9+vmware.1-vsc9.1.1.0-25712839/ v1.32.9+vmware.2-fips-vsc 9.0.2.0-25129014	

COMPONENT	VERSION	NOTES
Redis Enterprise Software	8.0.20-44	
memtier_benchmark	V2.5.1	redislabs/memtier_benchmark:2.5.1

Kubernetes Environment

Table 3: Supervisor Control Plane and vSphere Kubernetes Service

COMPONENT	VERSION	NOTES
vSphere Kubernetes Service	3.7/ 3.6	Control plane Service, CNCF-conformant Kubernetes workload clusters

VKS Cluster Configuration

A single VKS cluster comprises three control plane VMs and two dedicated worker pools: redis-node-pool for Redis Enterprise components and benchmark-node-pool for memtier_benchmark clients. The validated configurations use three Redis worker VMs with three client worker VMs for baseline low-latency and endurance tests, and five Redis worker VMs with six client worker VMs for scale-out, multi-tenant, and online expansion tests. These counts refer to worker VMs. In both configurations, the Redis pool occupies one group of three ESXi hosts and the client pool occupies the other three. Kubernetes node selectors and pod anti-affinity control pod placement on worker VMs; vSphere placement rules maintain the separation between the two physical-host groups.

The baseline worker VMs use the guaranteed-4xlarge VM class with 16 vCPUs, 128 GB of RAM, and full compute reservations.

The VKS Worker VM Sizing section presents the sizing comparison and scale-out rationale; Best Practices provides guidance on NUMA locality and host capacity.

Cluster Size and Availability: Redis recommends an odd number of cluster nodes, with a minimum of three; 3-, 5-, and 7-node clusters are common sizing examples. This reference architecture validates three and five Redis Enterprise nodes. For larger deployments, select node count and shard placement together with host capacity and quorum requirements. Where multiple Redis worker VMs share an ESXi host, align primary/replica placement and Redis rack-zone awareness with the physical failure domains; separate worker VMs alone do not provide separate host-failure domains.

Table 4: Kubernetes Cluster Configuration Summary

COMPONENT	CONFIGURATION	NOTES
Kubernetes Control Plane	3 Replicas — 4x vCPU 16GB RAM	VM Class: “guaranteed-large” Storage Class: “vsan-esa-auto-raid-policy”

COMPONENT	CONFIGURATION	NOTES

Kubernetes Worker Nodes	Two deployment layouts and validated topologies:
	• Baseline SLA and endurance scenario config:
	  ○ redis-node-pool (Redis Enterprise cluster nodes): 3 worker VMs
	  ○ Benchmark-node-pool (memtier_benchmark client pods): 3 worker VMs
	• Scale-out, multi-tenant and expansion scenario config:
	  ○ redis-node-pool (Redis Enterprise cluster nodes): 5 worker VMs
	  ○ Benchmark-node-pool (memtier_benchmark client pods): 6 worker VMs
	Per Worker VM:
	16 vCPUs, 128 GB RAM (guaranteed-4xlarge, 100% reservation; 16 vCPUs within the die's 24-core capacity)
	Physical placement:
	• Redis worker VMs use three ESXi hosts
	• Benchmark worker VMs use the other three
		VM Class: "guaranteed-4xlarge"
		Storage Class: "vsan-esa-auto-raid-policy"

Kubernetes Release / VKr Identifier	VCF 9.1.1 (Production Reference): • Kubernetes Release: v1.36.1+vmware.4-vkr.5 • VKR / TKR Name: v1.36.1---vmware.4-vkr.5 VCF 9.0.2 (Baseline): • Kubernetes Release: v1.35.2+vmware.1-vkr.3 • VKR / TKR Name: v1.35.2---vmware.1-vkr.3	Deploys CNCF-conformant Kubernetes workload clusters. The VKR name is referenced in declarative cluster manifests (e.g., redis-vks.yaml), while runtime nodes and apiserver report upstream Kubernetes versions (v1.36.1+vmware.4 for VCF 9.1.1; v1.35.2+vmware.1 for VCF 9.0.2).
OS Image	Ubuntu 24.04	
Cluster Domain	cluster.local	
CIDR Blocks	Pods: 192.168.0.0/16 Services: 10.96.0.0/12	

The complete declarative definition of the VKS cluster—including dual redis-node-pool and benchmark-node-pool specifications—is defined in the redis-vks.yaml file in Appendix. As previously outlined, applying this manifest to the Supervisor triggers the creation of the cluster, which is then managed and reconciled by Cluster API.

Networking Configuration

The network architecture relies on VMware vSphere Distributed Switches (VDS), supported by redundant 25GbE physical uplinks in the validated hardware configuration across the vSphere cluster. To establish a secure, multi-tenant software-defined networking fabric, the underlying vSphere Supervisor is deployed in VPC mode, powered by VMware NSX VPC networking. Within this architecture, the Redis Enterprise vSphere namespace is provisioned with its own dedicated Virtual Private Cloud (VPC). This VPC secures an isolated Layer-3 routing domain, providing dedicated logical routers and software-defined gateway services exclusively for Redis Enterprise nodes, database instances, and client benchmark workloads.

Storage Configuration

Storage is provided by vSAN ESA under the vSAN ESA Auto RAID policy. The examples use vsan-esa-auto-raid-policy for the default storage class and vsan-esa-auto-raid-policy-latebinding for Redis persistent volumes. The latter uses WaitForFirstConsumer to coordinate volume provisioning with pod scheduling and available storage topology.

Deploy Redis Enterprise on VKS

Deploying Redis Enterprise on VMware Cloud Foundation (VCF) with vSphere Kubernetes Service (VKS) leverages a fully declarative, GitOps-aligned Kubernetes Operator pattern. Rather than relying on manual virtual machine administration or imperative configuration scripts, the entire platform lifecycle—from underlying infrastructure provisioning and OS kernel tuning to cluster formation, proxy routing, and multi-tenant database provisioning—is orchestrated through Kubernetes custom resources and declarative manifests.

A foundational design principle of this architecture is the strict separation of concerns between the infrastructure management plane and the application workload plane, centered on a dedicated redis-enterprise namespace across both the vSphere Supervisor control plane and the VKS workload cluster:

- **vSphere Supervisor Control Plane (redis-enterprise Namespace):** Defines the infrastructure boundaries within VMware Cloud Foundation. It provides tenant isolation, a dedicated VMware NSX Virtual Private Cloud (VPC) with software-defined routing and distributed firewalling, enterprise storage policies backed by VMware vSAN™ Express Storage Architecture (ESA), and quota management.
- **VKS Workload Cluster (redis-enterprise Namespace):** Hosts the Redis Enterprise Operator, cluster management services, DMC proxies, and database shards. Worker-pool selection, pod scheduling, and physical-host placement together define the deployment topology.

By standardizing declarative manifests across both the Supervisor and VKS workload clusters, platform engineering teams establish a version-controlled, reproducible deployment pipeline suitable for continuous integration, automated testing, and zero-downtime Day-2 operations.

End-to-End Manifest Deployment Sequence

To establish an enterprise-grade, highly available Redis Enterprise deployment on VMware Cloud Foundation, manifests must be applied in a coordinated, sequential lifecycle across the Supervisor control plane and the VKS workload cluster. The complete end-to-end workflow comprises five distinct phases:

Table 5 details the end-to-end manifest sequence, deployment target planes, configuration files, and architectural objectives across the complete lifecycle:

Table 5: End-to-End Manifest Deployment Sequence and Lifecycle Workflow

DEPLOYMENT PHASE	TARGET CONTEXT & NAMESPACE	MANIFEST / TOOL	KEY CONFIGURATION & ARCHITECTURAL OBJECTIVE
Phase 1: Supervisor Setup & Day-0 Tuning	Supervisor Plane (redis-enterprise NS)	redis-tuned-profile.yaml antrea-hybrid-encap.yaml	Registers declarative TuneD Profile (throughput-performance reference baseline; latency-performance is an alternative to evaluate; 16MB TCP buffers, THP disabled) and configures Antrea trafficEncapMode: hybrid for direct L2 container routing.
Phase 2: VKS Workload Cluster Deployment	Supervisor Plane (redis-enterprise NS)	redis-vks.yaml	Provisions 3-node HA control plane and dual dedicated worker pools (guaranteed-4xlarge, 16 vCPUs): redis-node-pool (5 or 3 VMs for Redis Enterprise nodes) and benchmark-node-pool (6 or 3 VMs for memtier_benchmark client pods).
Phase 3: Redis Enterprise Operator Installation	VKS Workload Cluster (redis-enterprise NS)	Version-specific Operator installation procedure	Installs Redis Enterprise Operator controller, RBAC roles, admission webhooks, and custom resource definitions (REC, REDB, RERC) for automated lifecycle management.
Phase 4: Redis Enterprise Cluster Provisioning	VKS Workload Cluster (redis-enterprise NS)	redis-enterprise-cluster.yaml	Provisions Redis Enterprise StatefulSet (spec.nodes: 5 for scale-out/expansion, or spec.nodes: 3 for baseline latency/endurance). Uses worker-node anti-affinity, vSphere placement rules for physical-host separation, vSAN ESA Auto RAID persistence, and the proxy thread tuning described in Measurement Procedure.
Phase 5: Redis Enterprise Database Creation	VKS Workload Cluster (redis-enterprise NS)	redb-cache-only.yaml redb-cache-aof.yaml redb-cache-rdb.yaml	Provisions tenant databases with memory quota (e.g., 70 GB), HA replication, TLS 1.3, and targeted persistence (None, AOF, or RDB). For balanced initial placement, use S = k × N, where S is the primary shard count, N is the Redis node count, and k is an integer. Use the reported counts: 5 or 10 primary shards on 5 nodes, and 6 or 9 primary shards on 3 nodes; account for replica resources as well.

Following manifest deployment, platform administrators verify operational readiness via kubectl get rec -n redis-enterprise (Status: Valid) and kubectl get redb -n redis-enterprise (Status: Active). The web-based Redis Enterprise Cluster Management Console provides a centralized dashboard illustrating cluster quorum, symmetric primary and replica shard placement across the worker nodes, and operational database endpoints, as shown in Figure 4.

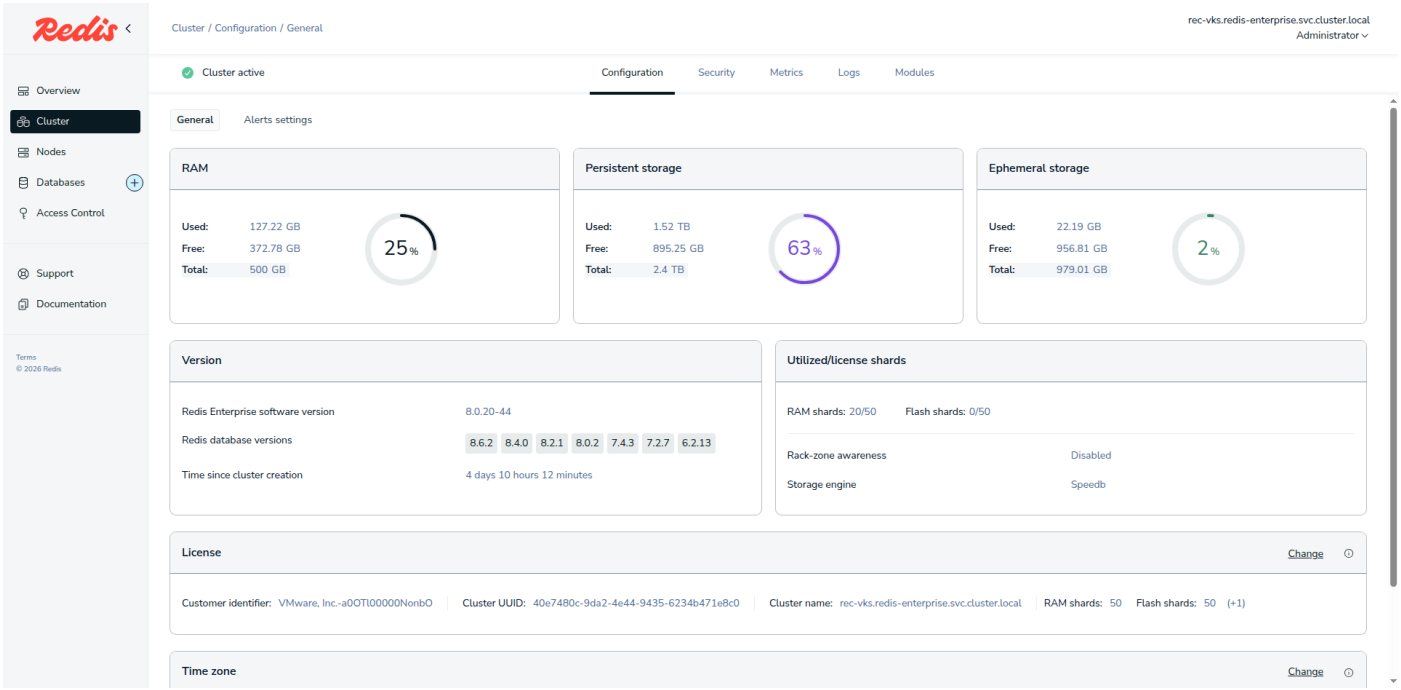


Figure 4: Redis Enterprise Cluster Management Console Overview

The following validation findings guide worker VM sizing, persistence, performance tuning, shard placement, and lifecycle operations. The Appendix provides VKS, REC, and REDB manifest examples to support deployment.

Solution Validation

Validation Approach and Testing Tools

This reference architecture presents key validation findings that inform deployment decisions for Redis Enterprise on VCF with VKS. Each section connects workload characteristics and measured performance to actionable guidance for sizing, tuning, persistence, and operations. Throughput and latency were measured using memtier_benchmark.

Shared Workload Context

- **Load Generation and Concurrency:** Client pods ran on a dedicated benchmark pool of three or six worker VMs. All connection settings, including the 30–120 range, are per client pod. Aggregate concurrency depends on the active client-pod count and each pod's connection configuration—not the worker VM count. Any endpoint-level connection totals are reported separately.
- **Latency Interpretation**: Latency percentiles are labeled GET, SET, or Combined (Totals). An unqualified P99 for a mixed workload refers to the combined command distribution. Compare results at the same pipeline depth, because deeper pipelines increase outstanding requests and can raise queueing latency.
- **Physical Placement**: Redis worker VMs occupy three ESXi hosts and benchmark worker VMs occupy the other three. The 5-Redis/6-client topology shares hosts within each pool, while client-to-Redis traffic crosses the physical 25 GbE fabric between the host groups. Kubernetes scheduling controls pod placement; vSphere placement controls VM placement.
- **Uniform Random (R: R) vs Gaussian (G: G)**: Workloads evaluated uniform key distribution across 24GB raw data pre-populated keys (R: R) as well as Gaussian bell-curve hot-key distributions (G: G) during benchmark testing.
- **Data size**: 512B,1KB, 2KB, 4KB to isolate the impact of packet size on performance.

- **Pipeline Depth**: Depths 1–4 characterize latency-oriented workloads; depths 8–32 characterize batching and aggregate throughput. Each section states the depths relevant to its finding.
- **TLS**: Enabled, standard secure database connectivity.
- **Standard Test Duration**: 300 seconds per standard run (with documented exceptions: 90 minutes for sustained endurance validation and 600 seconds for online expansion monitoring).
- **Cluster-Aware Clients**: Enable memtier_benchmark cluster mode (--cluster-mode) to discover database endpoints and route requests across the shard topology.

Testing Methodology

The benchmark procedure prepared the dataset, verified load-generator capacity, varied workload settings, and measured throughput and latency. Each section presents the conditions and findings that inform the architectural comparison and deployment guidance.

- **Dataset Preparation**: Preload the working dataset to minimize initialization effects and ensure a consistent starting point for each benchmark.
- **OS Kernel and Network Tuning**: Apply TuneD Profile to increase socket-buffer limits and connection backlogs and enable Antrea Hybrid Encapsulation to bypass encapsulation for eligible same-subnet node traffic. Verify both configurations on worker VMs before benchmarking.
- **Load-Generator Capacity**: Monitor client CPU and network utilization as load increases to ensure the load generators can sustain the target workload without limiting measured database performance.
- **Pipeline Sweep**: Compare throughput and P99 together as pipeline depth increases; select the operating point that meets the application's latency objective.
- **DMC Proxy Tuning**: Configure 10 active proxy threads with a maximum of 14 using `rladmin tune proxy all threads 10 max_threads 14`. Balance proxy concurrency with primary and replica shard processing and management activity within the pod's shared 14.5-vCPU limit to meet throughput and latency targets.
- **Resource Monitoring**: Correlate throughput plateaus with worker VM and host CPU utilization, CPU throttling, proxy activity, and network usage to identify bottlenecks and guide tuning or scaling decisions.
- **Configuration Comparison**: Evaluate throughput and latency across complete deployment configurations, accounting for differences in worker VM count, client capacity, persistence, and platform release to guide sizing and scale-out decisions.

Tuning Context

The documented reference configuration combines VCF 9.1.1 with Antrea Hybrid Encapsulation and redis-optimized-profile, which inherits throughput-performance and adds Redis-oriented memory and network settings. Use this configuration as the scale-out reference. The platform comparison separately identifies VCF 9.0.2 and VCF 9.1.1, while the OS and network tuning comparison evaluates the optimized stack against default Linux settings and overlay networking. For latency-sensitive applications, evaluate a latency-performance-based profile at the intended concurrency and pipeline depth, then select the profile that meets the application's throughput and latency targets.

Performance Tuning and Operational Validation

Key Findings and Architecture Decisions

The selected validation findings guide the following architecture decisions for platform selection, workload sizing, and operations:

- **Platform Configuration**: The VCF 9.1.1 configuration combined higher throughput, lower GET and SET P99 latency relative to the VCF 9.0.2 comparison configuration. And storage reduction (measured 6.44:1 data reduction ratio on the benchmark dataset), evaluate performance and capacity efficiency together and size storage using representative application data.
- **OS and Network Tuning**: The throughput-performance-based redis-optimized-profile and Antrea Hybrid Encapsulation improved throughput and GET/SET P99 latency as a combined configuration. Use this stack as the documented scale-out reference and evaluate alternative profiles against the application's concurrency, pipeline depth, and latency targets.

- **Worker VM Sizing**: Larger worker VMs provided limited gains or reduced performance in the sizing comparison. Start with 16-vCPU worker VMs on this platform and expand horizontally, checking host capacity, ESXi placement, and memory locality as the deployment grows.
- **Persistence and Endurance**: The short-run comparisons characterize cache-only, RDB, and AOF performance, while the 90-minute RDB run demonstrates sustained throughput and sub-millisecond combined P99 for the tested workload with hourly snapshots configured. Select persistence to meet recovery requirements and use both comparisons to guide sizing.
- **Shard Density**: Two primary shards per 16-vCPU worker VM provide a starting point for the tested shallow-pipeline workloads; three per worker VM increased throughput under deeper pipelines. Select shard density for the intended workload and budget CPU and memory for replicas, proxies, and persistence.
- **Low Latency and Scale-Out**: The three-node, six-primary-shard configuration met sub-millisecond combined P99 targets in the reported shallow-pipeline workloads. The five-node, ten-primary-shard configuration, with increased client capacity, delivered higher throughput at a higher measured P99. Choose the operating point that meets the application's throughput and latency requirements.
- **Multi-Tenant Consolidation**: Concurrent databases shared throughput approximately evenly and exhibited comparable latency under symmetric load. Use balanced shard placement to consolidate tenants with similar demand and assess per-database performance as the workload mix changes.
- **Online Shard Expansion**: The database expanded from five to ten primary shards under active traffic, with zero connection errors recorded by the client workload. Use declarative REDB scaling to increase database parallelism, with sufficient resources for the expanded shard set.
- **HA Operations**: Pod deletion and worker-node drain-and-return tests demonstrated pod restoration and client workload completion under active load. Build recovery and maintenance procedures around pod readiness, database availability, and application recovery checks.

VCF v9.0.2 with vSAN ESA (Deduplication Disabled) vs. v9.1.1 with vSAN ESA Deduplication

This comparison evaluates Redis Enterprise on two platform configurations: VCF 9.0.2 with vSAN ESA deduplication disabled and VCF 9.1.1 with deduplication enabled. It examines the combined effects of the platform upgrade and deduplication on throughput, P99 latency, and storage efficiency, providing practical guidance for balancing application performance with storage capacity savings. Refer to Next-Gen Topology-Aware CPU Scheduler Performance Improvements in VCF 9.1.

To quantify the empirical performance differences and evaluate the benefits of these architectural advancements, head-to-head benchmarking was conducted between VCF 9.1.1 (Kubernetes Release v1.36.1+vmware.4-vkr.5) and baseline VCF 9.0.2 (Kubernetes Release v1.35.2+vmware.1-vkr.3) on the reference 5-Node / 10-Shard cluster (16 vCPU worker nodes, 10 primary + 10 replica shards = 20 total shards, TLS 1.3 enabled, 900 concurrent client connections, 1:10 write:read ratio).

The release comparison measures the complete deployment configuration: VCF 9.0.2 with deduplication disabled versus VCF 9.1.1 with deduplication enabled. It demonstrates the combined performance and storage-efficiency outcome of that configuration change.

Analysis of Results

As Figure 5 shows performance improvement by comparing VCF 9.0.2 baseline with VCF 9.1.1:

- **Throughput**: The VCF 9.1.1 configuration delivered 7.2–41.8% higher throughput than the VCF 9.0.2 configuration across the tested payloads. The RDB and AOF configurations also performed close to cache-only in the short-run comparisons.
- **P99 Latency**: The VCF 9.1.1 configuration reduced GET P99 latency by 9.9–47.4% and SET P99 latency by up to 43.1% relative to the baseline configuration across the evaluated concurrency spectrum. The result captures the combined platform and storage-configuration changes.

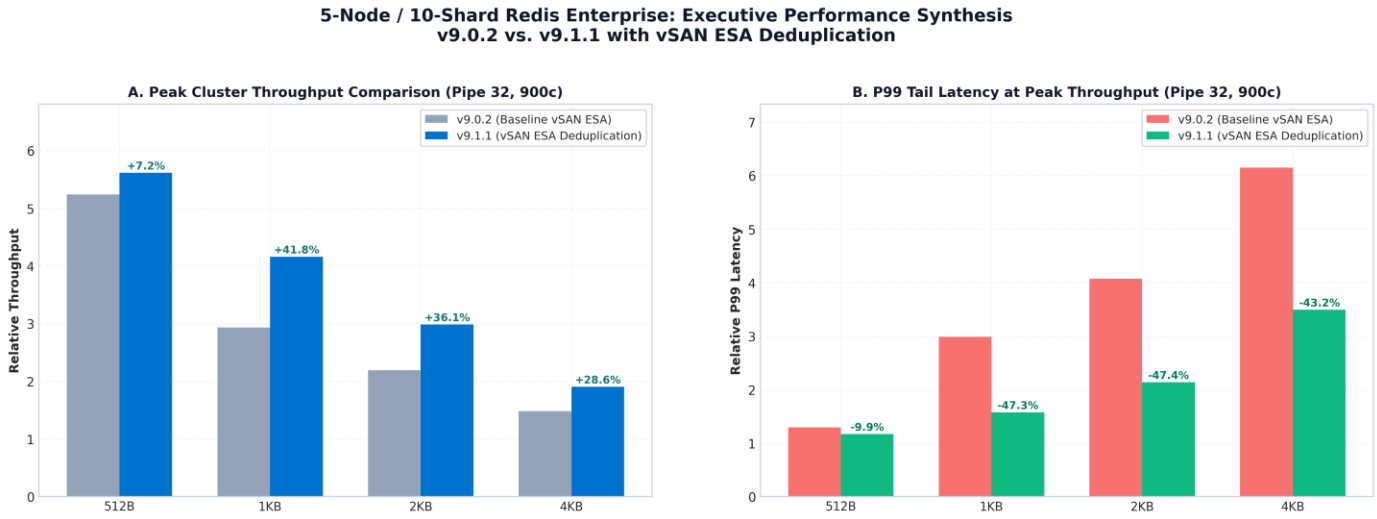


Figure 5: VCF 9.0.2 baseline vs. VCF 9.1.1 with vSAN ESA Deduplication Performance Comparison

Storage Efficiency: The VCF 9.1.1 configuration achieved 6.44:1 data reduction with deduplication and compression enabled for the benchmark dataset. Use representative application data to translate this capacity-saving opportunity into a deployment-specific storage budget.

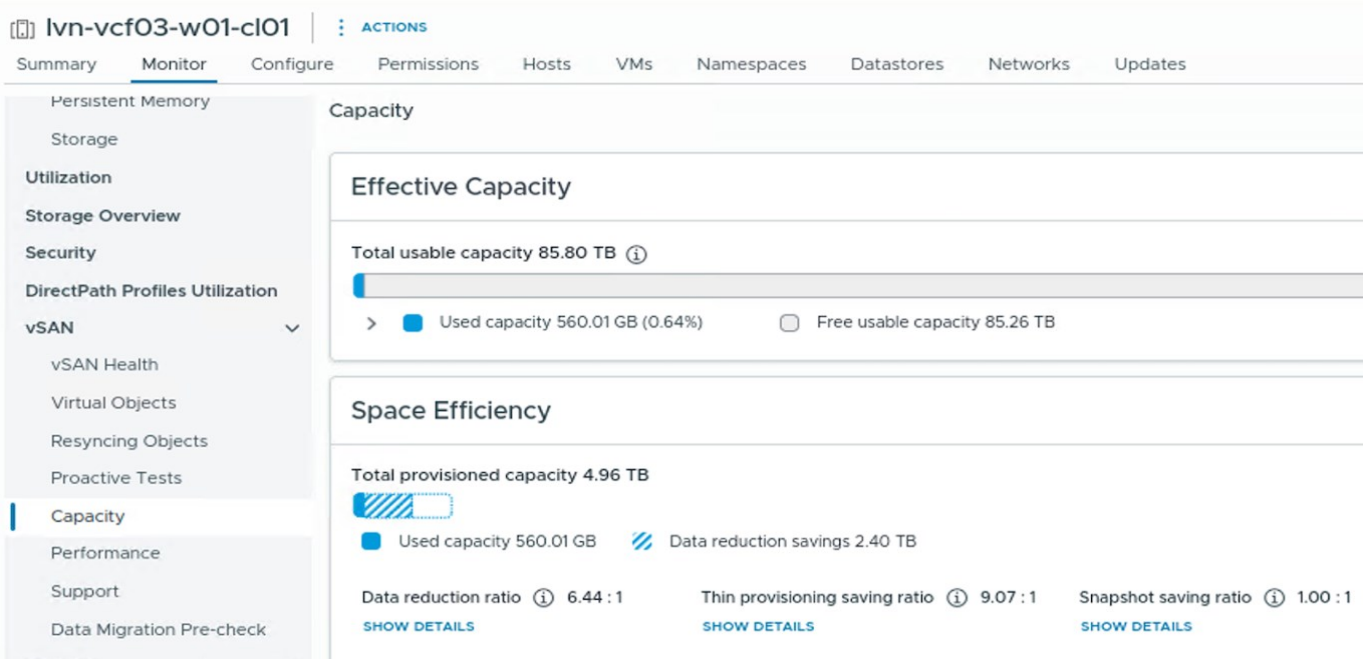


Figure 6: Data Reduction Ratio during Benchmark Testing

Conclusion

The VCF 9.1.1 configuration delivered higher throughput, lower P99 latency, and reduced storage consumption for the benchmark dataset, demonstrating a strong balance of application performance and storage efficiency. With vSAN ESA deduplication and compression enabled, the tested configuration combined responsive Redis Enterprise processing with efficient persistent storage. These findings support VCF 9.1.1 as a foundation for scaling Redis Enterprise workloads while managing storage capacity, with deployment sizing guided by application-specific data characteristics and performance requirements.

Linux OS-Level Tuning via TuneD Profiles

VKS introduces the TuneD Profile declarative API, a declarative API for Kubernetes OS tuning. It replaces imperative scripts and unsafe sysctls with a supported, Kubernetes-native approach.

The redis-optimized-profile example combines the throughput-performance base with Redis-oriented memory and network settings. Use it as the documented scale-out reference configuration. For latency-sensitive applications, compare a latency-performance-based profile at the intended concurrency and pipeline depth, then select the profile that best meets the application's operating target.

Create redis-optimized-profile in the redis-enterprise Supervisor namespace, then reference it in the worker-pool osConfiguration block of redis-vks.yaml. The TunedProfile API and profileRef pattern follow the VMware VKS 3.6 OS tuning reference. Verify the resource against the installed Supervisor API and confirm the active worker settings with tuned-adm active and tuned-adm verify.

```
#redis-tuned-profile.yaml

apiVersion: os.kubernetes.vmware.com/v1alpha1
kind: TunedProfile
metadata:
  name: redis-optimized-profile
  namespace: redis-enterprise
spec:
  content: |
    [main]
    summary=Optimized profile for Redis Enterprise on VMware VCF 9.x
    # Base Profile Selection:
    # - throughput-performance: Default tested baseline; optimizes CPU throughput and
I/O pipeline bandwidth
    # - latency-performance: Alternative to evaluate for latency-sensitive workloads
    include=throughput-performance

    [sysctl]
    net.core.somaxconn=65535
    net.ipv4.tcp_max_syn_backlog=65535
    net.core.netdev_max_backlog=100000
    net.ipv4.tcp_rmem=4096 87380 16777216
    net.ipv4.tcp_wmem=4096 65536 16777216
    net.ipv4.tcp_keepalive_time=120
    net.ipv4.tcp_fin_timeout=30
    net.ipv4.tcp_tw_reuse=1
    vm.overcommit_memory=1
    vm.swappiness=1

    [vm]
    transparent_hugepages=never
```

Antrea Hybrid Encapsulation CNI

Overlay CNI networking encapsulates traffic between nodes. Antrea Hybrid Encapsulation selects direct routing for eligible same-subnet node traffic and encapsulation for traffic that crosses subnets.

Antrea Hybrid Encapsulation optimizes CNI routing within VMware Cloud Foundation (VCF) environments by intelligently adapting to topology:

- **No Encapsulation (Direct Routing):** Automatically applied to all pod-to-pod communication occurring on nodes within the same Layer 2 physical subnet.
- **Encapsulation:** Leveraged only when traffic must cross distinct Layer 3 subnet boundaries.

Configuring Antrea in Hybrid Encapsulation mode provides substantial architectural advantages:

- **Geneve / VXLAN Header Bypass**: For all intra-subnet pod-to-pod and pod-to-proxy communications, Antrea bypasses Geneve encapsulation entirely. Packets flow over native routed Layer 2 Ethernet frames without encapsulation headers.
- **CPU Datapath Efficiency**: Bypassing encapsulation and decapsulation for eligible same-subnet traffic reduces the work required on that network path.
- **Encapsulation & MTU Efficiency:** Bypassing overlay encapsulation for eligible same-subnet node traffic reduces packet-header overhead. Configure MTU and TCP Maximum Segment Size (MSS) consistently with the physical and virtual network paths.
- **Observed Combined Result**: TuneD plus Antrea Hybrid Encapsulation improved throughput and reduced GET/SET P99 latency in the reported comparison.

Apply the antrea-hybrid-encap.yaml file to enable Hybrid Encapsulation before deploying the VKS cluster.

```
apiVersion: addons.kubernetes.vmware.com/v1alpha1
kind: AddonConfig
metadata:
  name: redis-vks-antrea-config
  namespace: redis-enterprise
spec:
  clusterName: redis-vks
  values:
    antrea:
      config:
        trafficEncapMode: hybrid
```

Analysis of Optimization Results: Baseline vs. TuneD + Antrea Hybrid Encap

The benchmark results compare an un-optimized baseline (default Linux kernel parameters and standard overlay CNI encapsulation) against the optimized configuration (TuneD redis-optimized-profile + Antrea Hybrid Encapsulation CNI) across increasing concurrency levels.

Deploying TuneD in conjunction with Antrea Hybrid Encapsulation yields a 94.8% increase in throughput at a concurrency of 384 connections and successfully sustains peak throughput as connection counts scale. Furthermore, this configuration effectively reduces P99 GET and SET latencies, exhibiting a widening performance delta—meaning greater latency reductions—under higher connection loads.

Refer to Appendix Figure A-1: Redis Enterprise Normalized Throughput and GET/SET P99 Tail Latencies across Concurrency: Baseline vs. TuneD + Antrea Hybrid Encap for more details.

Conclusion

Combining TuneD optimization with Antrea Hybrid Encapsulation increased throughput by 94.8% at 384 aggregate client connections and reduced GET and SET P99 latency. Together, kernel buffer scaling and same-subnet encapsulation bypass support high-concurrency workloads. The separate low-latency validation identifies the workload and topology combinations that achieved sub-millisecond combined P99 latency.

VKS Worker VM Sizing

The worker-sizing comparison used a 3-node Redis Enterprise cluster with 16, 24, and 32-vCPU worker VMs. The first two sizes are within the 24-physical-core capacity of an SNC2 NUMA domain; 32 vCPUs exceed that capacity. The cluster hosted 9 primary and 9 replica shards, with TLS 1.3 enabled and payloads from 512 B to 4 KB.

Analysis of Results

The Intel Xeon 6740P hosts use SNC2 mode, exposing two NUMA domains per socket, each with 24 physical cores. The 16 and 24-vCPU worker configurations remain within this per-domain core capacity, while the 32-vCPU configuration exceeds it.

The benchmark results favor smaller worker VMs. Increasing vCPU count beyond the NUMA core boundary delivered lower throughput and higher P99 latency in the tested workloads.

- **16 to 24 vCPUs**: Throughput improved by only 1.6–3.8%, indicating limited benefit from additional CPU capacity with the tested nine-primary-shard configuration.

- **32 vCPUs**: Throughput for 512 B payloads decreased by 15.7%. For 2 KB payloads at pipeline depth 32, P99 latency reached 243.0 ms, an increase of 101.3%; at pipeline depth 4, it increased by 355%. These results favor worker VM sizes within the NUMA domain's core capacity.
- **4 KB payloads**: Additional vCPUs provided little throughput benefit, indicating that CPU capacity alone did not limit performance at this payload size.

Conclusion

Use 16-vCPU worker VMs as the baseline on this platform, with vCPU sizing guided by the 24-core SNC2 domain capacity. Expand throughput and memory capacity horizontally by adding Redis worker VMs, balancing shard placement and physical host resources as the cluster grows.

Refer to Appendix Figure A-2: Redis Enterprise Scale-up and NUMA Trap: Normalized Throughput Comparison and Figure A-3: Normalized P99 Tail Latency Escalation & The NUMA Scale-Up Trap for more details.

Persistence Scenarios: Caching Only vs. RDB vs. AOF

Redis Enterprise supports three operational persistence models configured via the REDB custom resource. Benchmark testing evaluated throughput overhead and latency impact across payload sizes (512B to 4KB):

- **Caching Only (No Persistence):** Operates purely in RAM. Delivers maximum raw IOPS and lowest latency, ideal for ephemeral session caches and distributed compute buffers.
- **Caching + RDB Snapshots**: Configures periodic point-in-time snapshots with snapshotEvery1Hour. In the replicated configuration, replica-side persistence is intended to reduce interference with primary request processing.
- **Caching + AOF (Every Second):** Configures aofEverySecond to persist the write log at a one-second fsync interval. Recovery behavior depends on the failure and the persistence and replication state.

Analysis of Results

Payload Size Impact: Across payloads from 512 B to 4 KB, throughput decreases as each operation transfers and processes more data. The results reflect the increasing demands on network transport, memory bandwidth, and TLS encryption as payload size grows.

RDB Performance: The 300-second comparison shows throughput close to cache-only with hourly RDB persistence configured. The dedicated 90-minute endurance validation evaluates sustained operation across the hourly persistence interval and provides the longer-duration basis for the RDB deployment guidance.

AOF Performance: The aofEverySecond configuration maintains throughput close to cache-only in the comparison, providing frequent write persistence with a small, measured throughput difference. Select it when the application benefits from a finer recovery interval than periodic snapshots.

Refer to Appendix Figure A-4: Normalized Throughput and P99 Tail Latencies by Payload Size across Persistence Scenarios for more details.

Conclusion

Redis Enterprise provides flexible persistence options: cache-only for rebuildable data, RDB for periodic snapshots, and AOF for more frequent write persistence. Select the mode that matches application recovery requirements, using the short-run comparisons to guide performance sizing and the 90-minute RDB endurance results to assess sustained operation with hourly snapshots configured.

90-Minute RDB Endurance Benchmark

The 90-minute RDB endurance validation was designed to assess sustained performance across the hourly snapshot interval. It used three Redis nodes with six primary and six replica shards on 16-vCPU worker VMs, with 60 reported connections, 512 B payloads, pipeline depth 4, a 1:10 write:read ratio, and TLS 1.3.

Analysis of Results

The RDB endurance workload maintained stable throughput and combined P99 tail latency (Totals) below 1.0 ms throughout the 90-minute run (both GET P99 SET P99 remaining strictly sub-millisecond). The benchmark operated at an aggregate concurrency of 60 client connections (generated by 3 client pods configured with 4 threads and 5 connections per thread). Its throughput was approximately 8.8% below the separate 300-second cache-only comparison, confirming that replica-offloaded hourly RDB snapshots sustain sub-millisecond tail latency without cumulative degradation.

Conclusion

The endurance result supports hourly RDB persistence for the tested latency-sensitive workload. Combined with the short-run persistence comparison, it provides a practical basis for sizing a replicated Redis deployment that balances sustained performance with periodic recovery images.

Shard Density Analysis

The shard-density comparison evaluated two configurations on three-node clusters with 16-vCPU worker VMs: six-primary shards and nine- primary shards configurations. These configurations average two and three primary shards per worker VM, respectively, with replica shards sharing the available CPU and memory.

Analysis of Results

- **Shallow Pipelines**: At pipeline depths 1–4 and the stated 60-connection setting, the six-primary and nine-primary configurations differed in throughput by −2.5% to +5.5%. Both achieved sub-millisecond P99 for the tested small-payload workloads. The six-primary configuration provides an efficient starting point when the latency objective takes priority.
- **Deep Pipelines**: Nine primary shards with nine replicas on three nodes increased throughput by 9.6–17.4% at pipeline depths 8–32. This higher-density configuration uses additional shard parallelism to support throughput-oriented workloads.

Conclusion

Shard Sizing: Use two primary shards per 16-vCPU worker for the tested shallow-pipeline workloads and three per worker to increase throughput under deeper batching. Include replica and proxy demand in the resource budget and scale out when additional density no longer improves the target operating point.

Refer to Appendix Figure A-5: Normalized Peak Throughput Comparison to evaluate the impact of Shard Density and Figure A-6: Normalized Low Latency Comparison across shallow pipeline to evaluate the impact of Shard Density (Combined P99 Totals, 30–120 Conns per Client Pod) for more details.

Low-Latency Workload SLA Validation and Scale-Out

The low-latency and scale-out comparison evaluated a three-node cluster with six primary shards and a five-node cluster with ten primary shards, both with replication and TLS 1.3. The workloads covered 512 B, 1 KB, and 2 KB payloads; 1:10 and 3:7 write:read ratios; 30–120 connections per client pod; and pipeline depths 1 and 2 across cache-only, AOF-every-second, and RDB persistence configurations.

Analysis of Results

The three-node, six-primary-shard and five-node, ten-primary-shard configurations provide two useful operating points for balancing throughput and tail latency:

Low-Latency Configuration (3 Nodes / 6 Primary Shards): Across shallow pipeline depths (Pipeline 1–2), both 1:10 and 3:7 write:read workloads sustained combined P99 tail latency (Totals) below 1.0 ms (e.g., 0.802 ms at Pipeline 2 for 3:7 write:read, with operation-specific SET P99 at 0.820 ms and GET P99 at 0.791 ms). Client concurrency was evaluated across a range of 30–120 connections per client pod. These results provide a sizing reference for applications with similar payloads, concurrency, and persistence requirements.

Scale-Out Configuration (5 Nodes / 10 Primary Shards): Increasing Redis worker VMs from three to five and client VMs from three to six delivered 90–140% higher throughput, reaching approximately 2.0M Ops/sec at about 1.0–1.5 ms P99. This is a scale-out result for the combined server and client configuration.

Write-Ratio Sensitivity (3:7): With 60 connections per client pod and a 1 KB payload, the three-node configuration maintained P99 below 1.0 ms. The five-node configuration delivered higher throughput at approximately 1.4 ms P99, providing a scale-out option for applications that prioritize additional capacity within that latency range.

Conclusion

The three-node, six-primary-shard configuration provides the validated starting point for the tested sub-millisecond workloads. The five-node, ten-primary-shard configuration expands throughput within the measured 1.0–1.5 ms P99 range. Select between these operating points using the application's payload, concurrency, persistence, and pipeline requirements.

Refer to Appendix Figure A-7: Normalized Throughput and P99 Latency Comparison — Three Nodes / Six Primary Shards and Five Nodes / Ten Primary Shards and Figure A-8: Normalized Write-Ratio Sensitivity Impact Comparison: Ratio 1:10 vs. Ratio 3:7 across 3-Node and 5-Node Clusters for more details.

Multi-Tenant Fair Sharing

To evaluate multi-tenant performance predictability, fair-share resource allocation, and resource consolidation efficiency, empirical benchmarking executed concurrent Cache-Only and Cache-AOF tenant databases on a shared 5-node Redis Enterprise cluster across pipeline depths (Pipeline 8 to 32) under symmetric concurrent client load.

Analysis of Results

Throughput Sharing: Under symmetric load, both databases split throughput evenly (~50% each), scaling combined output from 3.4 to 4.7 units as pipeline depth increased from 8 to 32.

Latency under Symmetric Load: Both tenant databases tracked matching combined GET/SET p99 curves across pipeline depths of 8, 16, and 32. The result demonstrates comparable service behavior under symmetric deep-pipeline demand and complements the separate shallow-pipeline latency findings.

Refer to Appendix Figure A-9: Multi-Tenant Dual Database Normalized Throughput and P99 Tail Latency for more details.

Shard-Density Comparison

Balanced Placement Rule: S = k × N, where S is the primary shard count per database, N is the Redis node count, and k is an integer. In the density comparison, five primaries per database on five nodes delivered higher throughput and 30–46% lower P99 than six primaries per database. Use balanced allocation to distribute work across nodes and inspect primary and replica placement as part of deployment verification.

Conclusion

Redis Enterprise on VCF delivers enterprise-grade multi-tenancy. The symmetric-load results support shared-cluster sizing for tenants with comparable demand. Use balanced shard placement as a starting point and measure each tenant's throughput and latency as the workload mix changes.

Online Shard Expansion

Enterprise database platforms must accommodate growing workloads while maintaining application availability. Redis Enterprise on VCF with VKS supports online shard expansion while the database continues serving requests. Managed declaratively through the Redis Enterprise Operator and REDB resource, this capability gives platform teams a consistent Kubernetes workflow for increasing database parallelism as demand grows.

Mechanics of In-Flight Dynamic Resharding

When an administrator increases the shardCount field in the RedisEnterpriseDatabase (REDB) custom resource, the Redis Enterprise Operator initiates a coordinated zero-downtime resharding workflow:

- **Declarative CRD Update**: The user or GitOps pipeline applies an updated REDB manifest specifying the new target shard count (e.g., shardCount: 10).
- **Parent-to-Child Shard Forking**: The cluster orchestration plane creates new child Redis engine processes on target worker nodes. Each existing parent primary shard establishes an in-memory replication link to its newly assigned child shard.
- **Hash Slot Redistribution**: Redis redistributes the existing slot ranges across the target primary shards. In the tested 5-to-10 expansion, each original shard's range is divided between two shards.
- **Live Migration and Routing**: Data moves to the expanded shard set while proxy routing is updated. The following workload validation describes client behavior during the tested expansion.

Expansion Multiple Rule

For online shard expansion, Redis Enterprise follows the Expansion Multiple Rule, where the target primary shard count is an integer multiple of the current count (for example, doubling from 5 to 10 primary shards, a 2x multiple; 3x and 4x multiples are also supported). In this mode, the system splits existing hash slot ranges equally among the expanded shards (for example, splitting each shard's assigned slots in half during a 2x expansion). This ensures balanced key redistribution and prevents hash-slot skew across the cluster.

Table 6: Expansion Modes and Operational Constraints

EXPANSION MODE	SUPPORTED SHARD RATIOS	OPERATIONAL CHARACTERISTICS & CONSTRAINTS
Online Live Expansion (CRD Update)	Integer multiples, for example 2x: 5->10; 3x: 5->15; 4x: 5->20. This paper reports 5->10.	Zero downtime; parallel fork-and-split; continuous client traffic without disconnection.
Initial Database Creation	Any integer (5, 10, 15, 20, etc.)	Uniform distribution across 16,384 slots calculated prior to key ingestion.
Fractional Expansion (e.g. 10->15)	Unsupported for Online Live Modification	Online resharding only increases the shard count by an integer multiple of the current count. A non-multiple target is rejected.

Live Workload Validation

A 600-second memtier_benchmark run exercised a Caching + AOF database while it expanded from five primary and five replica shards to ten primary and ten replica shards. The workload used 2 KiB payloads, a 1:1 write:read ratio, pipeline depth 1, and TLS 1.3. The raw run metadata reports 500 total connections across five endpoints, with four threads and 25 clients per thread.

Figures 7 and 8 show the shard configuration before and after expansion. Figure 9 provides a Redis UI traffic view, and Figure 10 plots the recorded client workload across the full benchmark run.

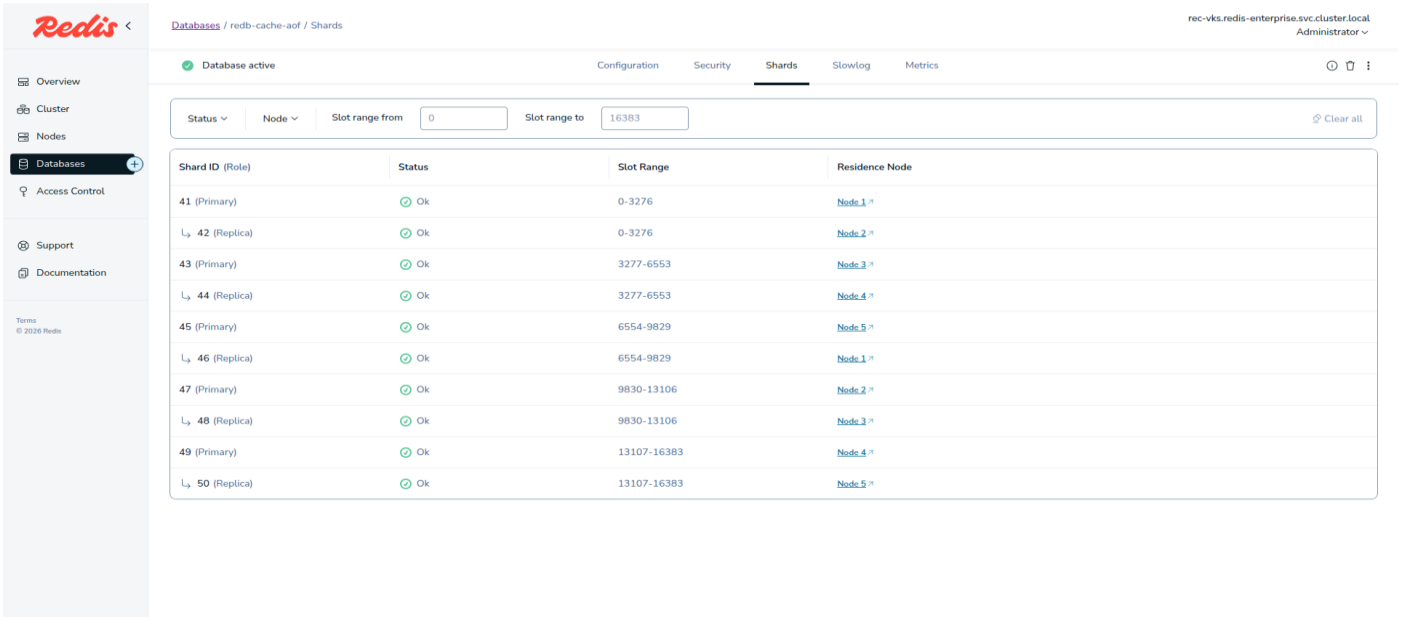


Figure 7: Pre-Expansion Database State with 5 Primary Shards and 5 Replica Shards (1 primary + 1 Replica per Node)

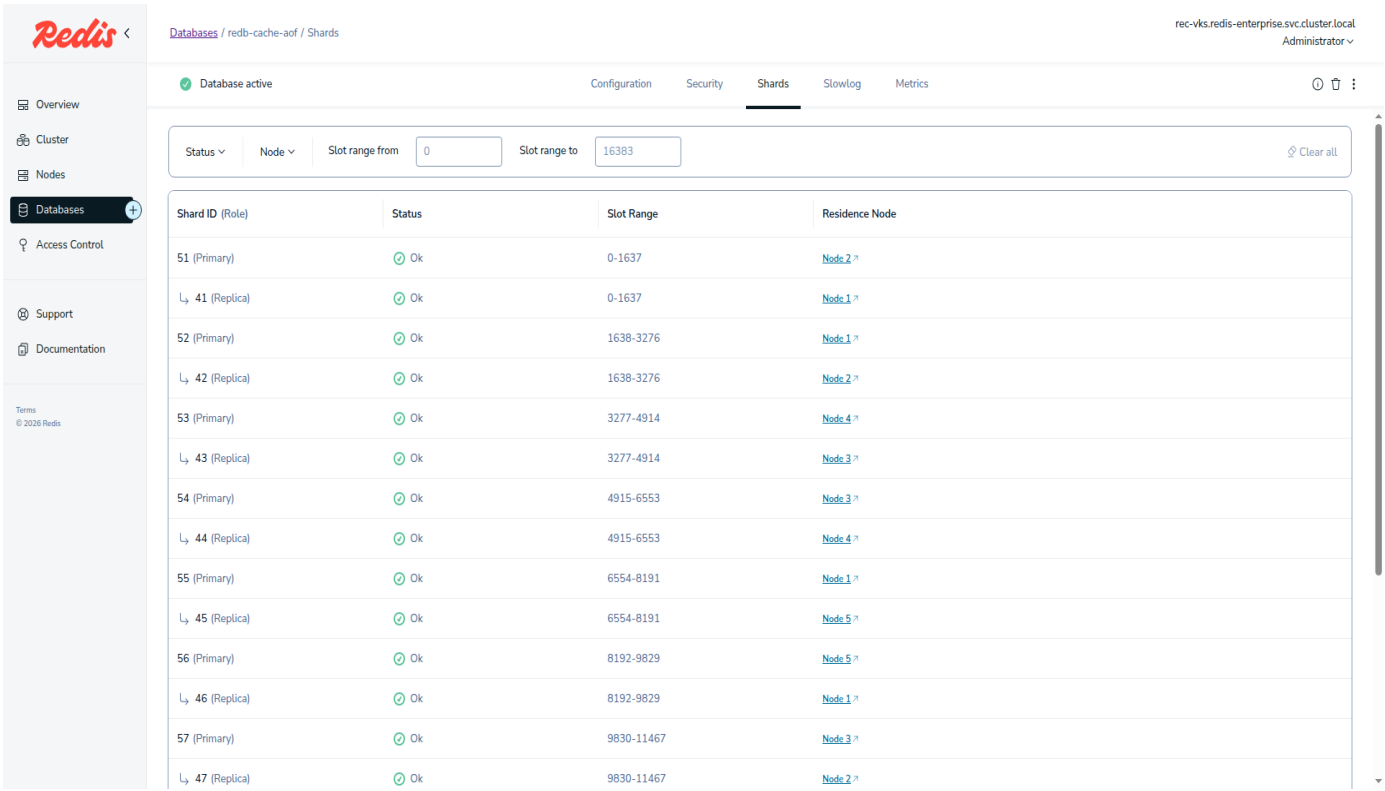


Figure 8: Post-Expansion Database State with 10 primary Shards and 10 Replica Shards (2 primary + 2 Replica per Node)

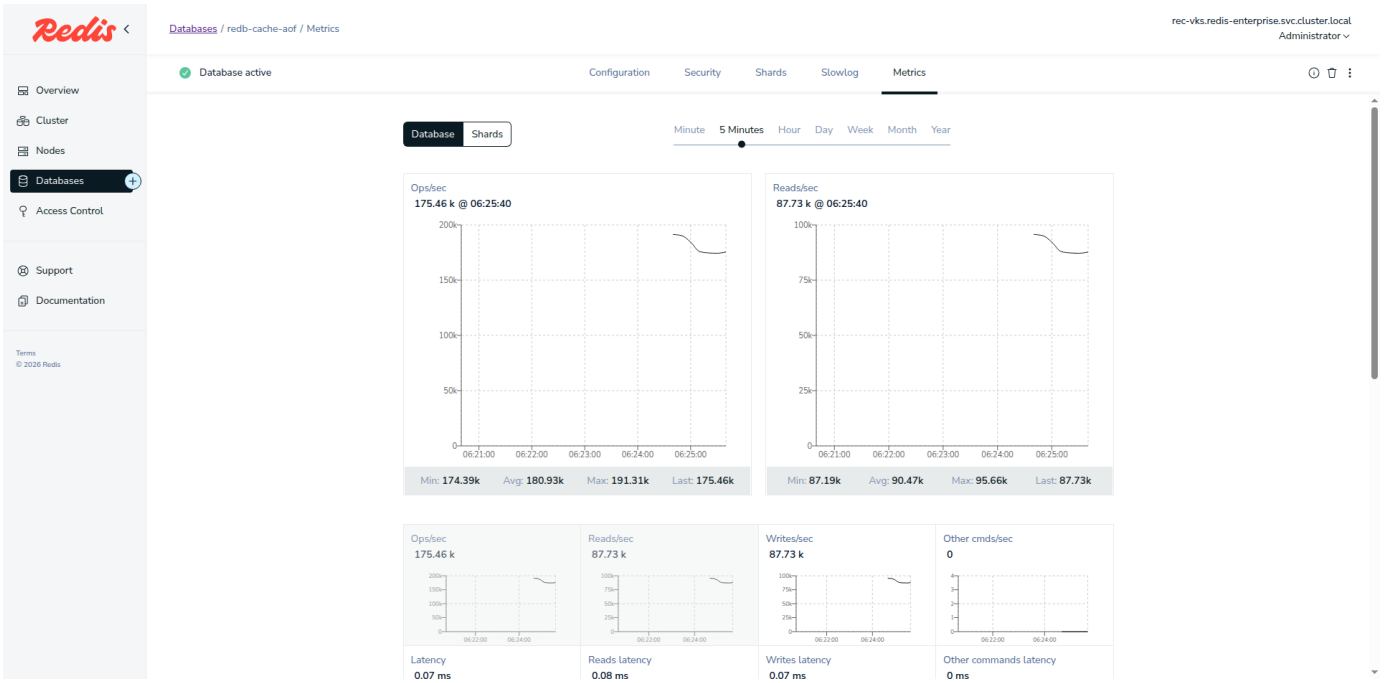


Figure 9: Redis UI Traffic Metrics Captured after Shard Expansion

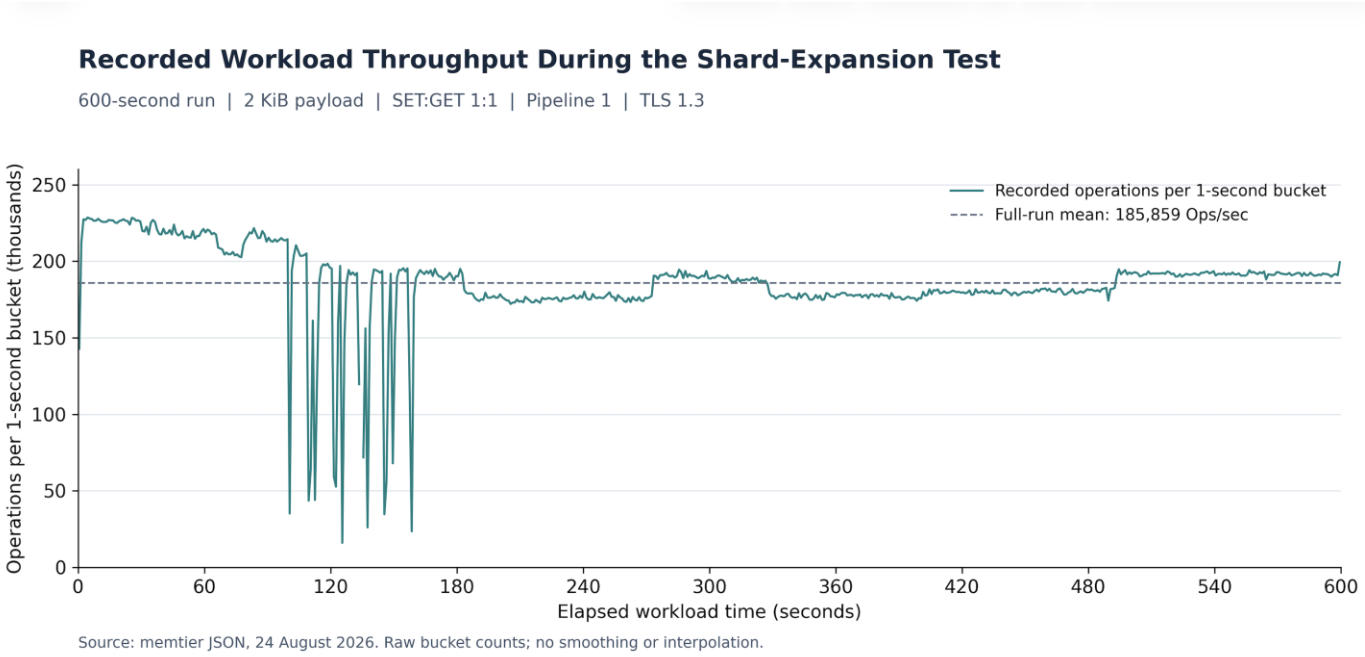


Figure 10: Recorded Workload Throughput during the Online Shard-Expansion Test

Expansion Timing and Workload: The execution report records 56 seconds from submission of the REDB shard-count change to completion, when all shards reported OK. Figure 10 shows the recorded 600-second memtier workload, which averaged approximately 185,859 Ops/sec and recorded zero connection errors. The curve plots completed operations in each recorded one-second interval.

Conclusion

Online shard expansion provides a practical capacity-growth mechanism while the database remains in service. The test expanded the primary shard count from five to ten under active traffic, with zero connection errors recorded by the client workload. The execution report documents completion in 56 seconds.

High Availability, Failover & Resiliency Validation

The HA validation exercised Redis Enterprise pod deletion and worker-node drain under active client workloads against an AOF-enabled database. These scenarios assessed automated pod recovery and operational resilience during faults and planned maintenance.

- **Pod Failure Recovery:** Redis pods were deleted while client workloads were active. Kubernetes restored the deleted pod to Ready, and the separate forced-deletion benchmark completed its configured workload with zero connection errors recorded in the client JSON summary. These findings demonstrate pod restoration and workload completion under the tested conditions.
- **Worker-Node Maintenance:** A worker node was cordoned, drained, and subsequently returned to service. All Redis pods reached Ready, and the associated benchmark completed with zero recorded connection errors. This validates the tested drain-and-return maintenance workflow under active load.

Figure 11 plots recorded client throughput for the initial-suite drain and forced-pod-deletion runs. Each panel uses its own elapsed workload time. Figure 12 provides a Redis UI view captured during the initial pod-deletion test; it provides complementary server-side telemetry.

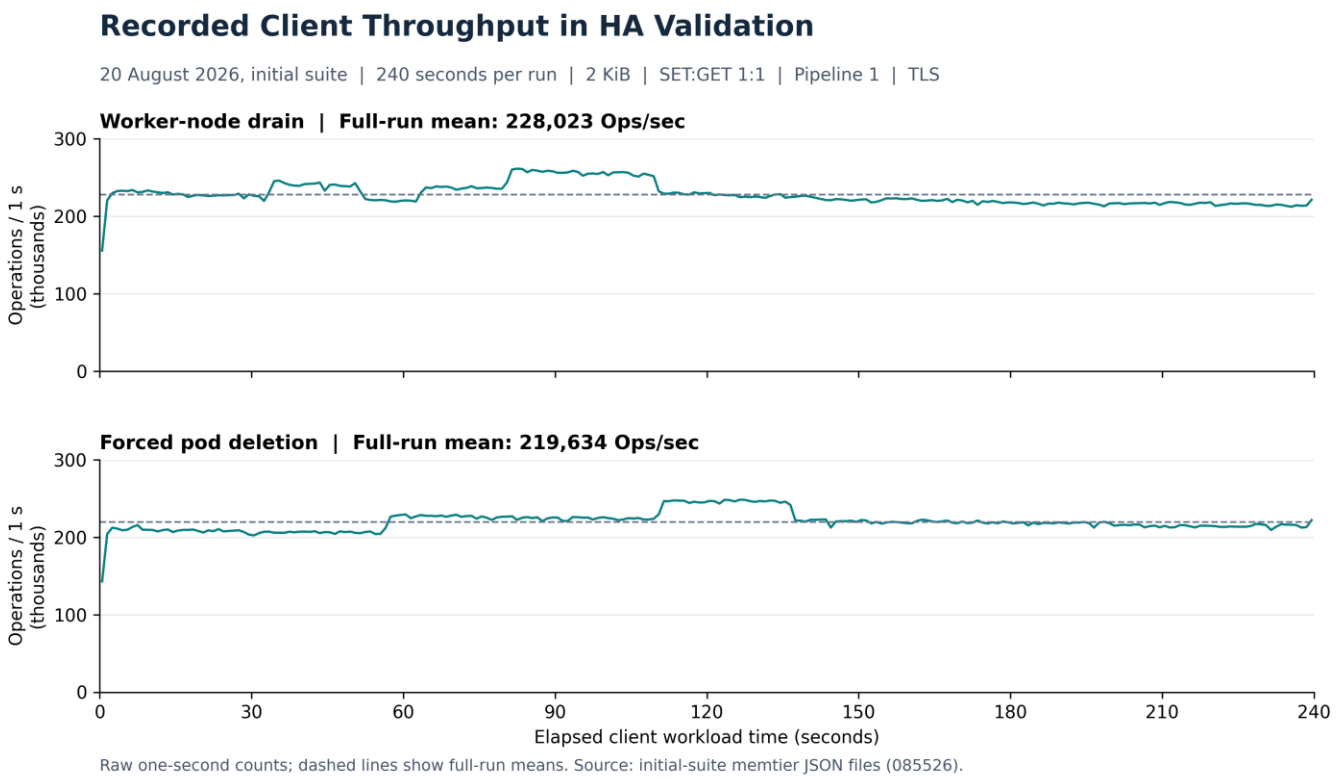


Figure 11: Recorded Client Throughput — Worker-Node Drain and Forced Pod Deletion (Initial Suite)

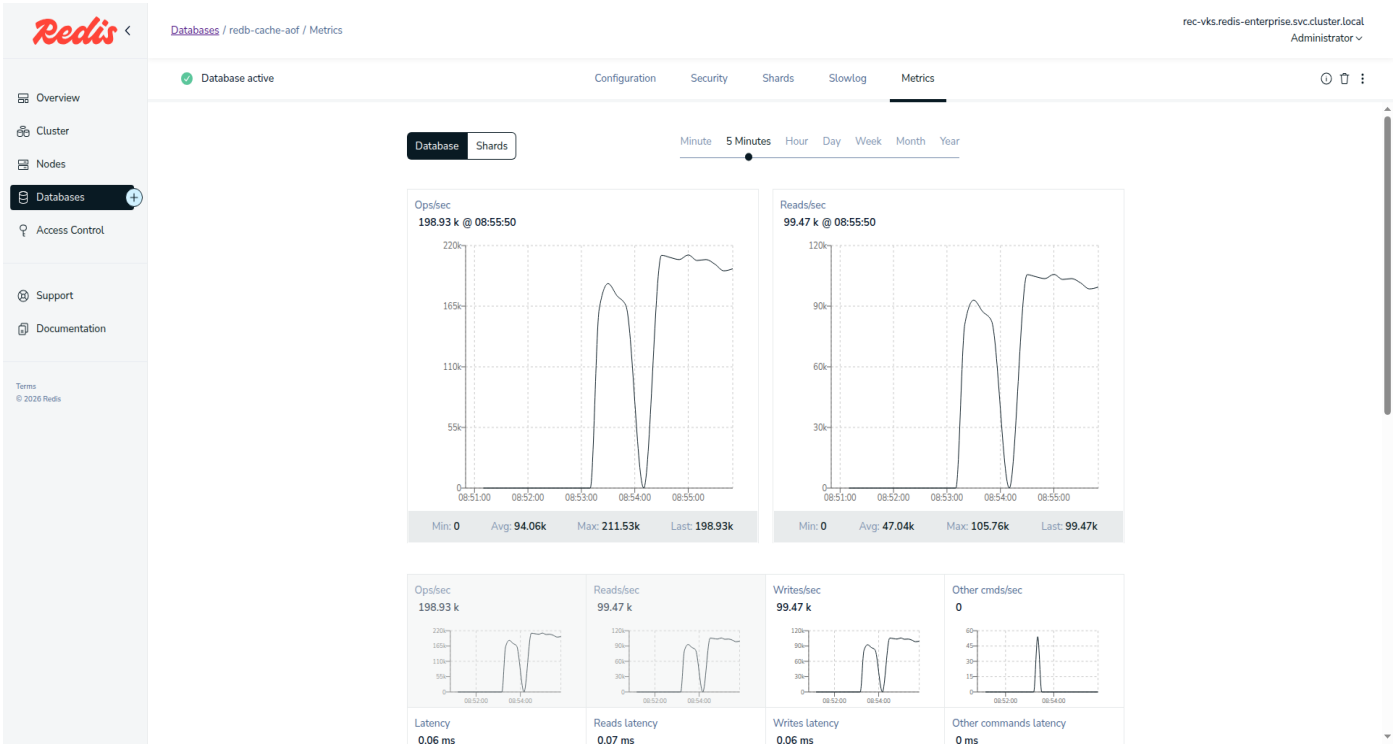


Figure 12: Redis Enterprise UI Metrics Captured during the Initial Pod-Deletion Test

Conclusion

The validation supports automated pod recovery and planned worker-node maintenance under active workloads. Incorporate pod readiness, database availability, and application recovery checks into operational procedures to confirm successful service restoration.

Best Practices

To achieve optimal Redis Enterprise performance on VKS on VMware Cloud Foundation, consider the following recommendations:

Compute & NUMA Architecture

- **Worker VM Sizing**: Use 16-vCPU worker VMs as the baseline on this platform, keeping vCPU counts within the 24-core SNC2 domain capacity. Favor horizontal scaling to reduce the need for cross-NUMA scheduling and verify actual placement and memory locality through ESXi metrics.
- **Host Capacity**: Budget for all co-located worker VMs, ESXi, NSX, and vSAN activity. A worker VM's unused share of a NUMA domain does not reserve dedicated cores for hypervisor services.

Storage & vSAN ESA Design

- **Use vSAN ESA Auto RAID Late-Binding**: Provision persistent volume claims with the late-binding storage class.
- **Persistent Storage**: Enable persistentSpec.enabled in the REC and select the intended vSAN ESA storage class. Size persistent volumes for the database persistence workload; when volumeSize is omitted, inspect the capacity provisioned by the installed Operator. The Appendix uses vsan-esa-auto-raid-policy-latebinding to coordinate provisioning with pod scheduling.
- **Storage Efficiency:** The benchmark dataset achieved a 6.44:1 storage reduction. Size deployment capacity using representative application data to align expected deduplication and compression savings with the workload's data characteristics.

Networking & Antrea CNI

- **Antrea Hybrid Encapsulation**: Bypass encapsulation for eligible same-subnet node traffic and retain encapsulation where required by the network topology.
- **Pair with TuneD Profile**: Deploy redis-optimized-profile TuneD CRD on all database worker nodes to expand TCP buffer depths and backlog queues.

Redis Enterprise Configuration & Scaling

- **Balanced Placement Rule**: Use S = k × N, where S is the primary shard count, N is the Redis node count, and k is an integer, as the starting point for balanced placement. Ten primaries across five nodes gives two per node. Confirm actual primary and replica placement and size the node resources for both roles.
- **Shard Density**: Use two to three primary shards per 16-vCPU worker as the tested starting range, then account for replicas, proxy activity, and persistence before increasing density.
- **DMC Proxy Concurrency**: On 16-vCPU worker VMs, the reference setting is 10 active threads with a maximum of 14: `rladmin tune proxy all threads 10 max_threads 14`. Threads share the Redis pod's 14.5-vCPU limit with shard and management processes. Check proxy CPU, shard CPU, throttling, and P99 under load before changing this setting.
- **Expansion Multiple Rule**: Choose a target primary shard count that is a supported integer multiple of the existing count. The validated example doubles five primaries to ten; balance the expanded placement against available node capacity.
- **Persistence Selection**: Use cache-only for rebuildable state where loss on restart is acceptable, RDB for periodic recovery images, and AOF every second for more frequent persistence. Select and test the policy against the required recovery point and recovery time; the fsync interval alone is not a universal loss bound.

Cross-Layer Bottleneck Diagnostics

Table 7 translates the validation findings into practical diagnostic guidance. Correlate CPU, network, storage, and latency metrics to identify bottlenecks and guide targeted tuning or scale-out decisions. Measured results are complemented by qualitative operational indicators.

Table 7: Bottleneck Reference and Architectural Tuning Guidance

Operating Regime / Bottleneck Category	Primary Limiting Subsystem & Root Cause	Diagnostic Indicators & Metrics (What to Observe)	Architectural Tuning & Remediation Guidance
Balanced Compute Allocation	Shard and proxy CPU demand are balanced for the workload.	Compare shard and proxy utilization, throughput, and P99 as concurrency changes.	Start with the tested two to three primary shards per 16-vCPU worker. Budget for replicas and proxies within the pod CPU limit; validate at the target pipeline depth.
Host or Container CPU Limits	Host contention or container CPU limits can restrict execution.	Inspect ESXi CPU ready time, host utilization, container CPU usage, and throttling counters over the same interval.	Budget capacity across co-located VMs and all container processes. Match requests and limits to the intended workload; verify Kubernetes QoS and CPU-manager eligibility separately before relying on dedicated CPUs.
Proxy CPU Saturation	Proxy CPU can limit throughput before shard engines reach capacity.	Look for high proxy CPU with lower shard utilization and a throughput plateau.	Evaluate 10 active proxy threads, maximum 14, using `rladmin tune proxy all threads 10 max_threads 14`. Threads share CPU; check throttling and P99 before increasing concurrency.
Shard Density and Contention	Additional shard and replica processes compete with proxies for worker resources.	Compare run-queue pressure, context switches, CPU usage, throughput, and P99 as shard density changes.	Use the six-versus-nine-primary comparison on three worker VMs as a starting point. Scale out when more shards no longer improve the target throughput and latency.
Network Packet Rate	Small-payload traffic can increase packet-processing demand before saturating link bandwidth.	Inspect softIRQ CPU, packet rate, NIC drops, and interface bandwidth together.	Use Antrea Hybrid Encapsulation for eligible same-subnet traffic. Evaluate supported ring sizes, RSS/RPS, and pipelining against the latency objective.
TLS Processing	Encryption and connection establishment consume CPU; the reported tests used TLS.	Inspect proxy CPU, encryption workload, and handshake activity to identify TLS-related CPU demand.	Maintain TLS and size proxy concurrency from measured CPU utilization and latency at the intended connection load.
Network Bandwidth	Large payloads can increase bandwidth demand and socket backpressure.	Compare pNIC throughput with link capacity, drops, queueing, and CPU utilization.	Distribute worker load across hosts with available network capacity and tune socket settings using measured behavior. The validated physical uplinks are 25 GbE.

Operating Regime / Bottleneck Category	Primary Limiting Subsystem & Root Cause	Diagnostic Indicators & Metrics (What to Observe)	Architectural Tuning & Remediation Guidance
Worker VM Size and NUMA	The 32-vCPU worker exceeded the 24-physical-core domain capacity and performed worse in the reported comparison.	Size worker VMs with NUMA topology in mind, and review ESXi scheduling and memory locality when larger VMs deliver lower throughput or higher tail latency.	Use 16-vCPU worker VMs as the platform baseline and scale out. Confirm physical placement and memory locality through ESXi metrics.

Conclusion

Redis Enterprise on VMware Cloud Foundation with VKS provides a scalable, resilient foundation for enterprise real-time data services. Efficient worker VM sizing, coordinated TuneD and Antrea optimization, and balanced shard placement combine to support low-latency workloads, high throughput, and multi-tenant resource efficiency. Horizontal scaling gives organizations a practical growth path as application demand increases.

vSAN ESA delivers persistent storage and data reduction, while Redis replication, automated failover, and online shard expansion support recovery and capacity growth. Kubernetes-native declarative management integrates these capabilities into established platform workflows, simplifying deployment and ongoing operations. Together, these architectural advantages enable organizations to build and scale Redis Enterprise services with performance, capacity, and recovery choices aligned to their business requirements.

Appendix

redis-vks.yaml

```
apiVersion: cluster.x-k8s.io/v1beta2
kind: Cluster
metadata:
  name: redis-vks
  namespace: redis-enterprise
spec:
  clusterNetwork:
    services:
      cidrBlocks:
      - 10.96.0.0/12
    pods:
      cidrBlocks:
      - 192.168.0.0/16
    serviceDomain: cluster.local
  topology:
    classRef:
      name: builtin-generic-v3.7.0
    version: v1.36.1---vmware.4-vkr.5
    controlPlane:
      replicas: 3
      metadata:
        annotations:
          run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu,os-version=24.04
    workers:
      machineDeployments:
      - class: node-pool
        name: redis-node-pool
        # Validated Redis Worker Pool Sizing:
        # - replicas: 5: Scale-out, multi-tenant, and online expansion topology (5
worker VMs)
        # - replicas: 3: Baseline low-latency SLA and endurance topology (3 worker VMs)
        replicas: 5
        variables:
          overrides:
          - name: vmClass
            value: guaranteed-4xlarge
          - name: osConfiguration
            value:
              tuned:
                active:
                - redis-optimized-profile
                profiles:
                  redis-optimized-profile:
                    profileRef:
                      name: redis-optimized-profile
          - name: volumes
            value:
            - capacity: 200Gi
              mountPath: /var/lib/containerd
              name: containerd
        metadata:
          annotations:
            run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu,os-version=24.04
      - class: node-pool
        name: benchmark-node-pool
        # Validated Benchmark Client Pool Sizing:
        # - replicas: 6: Scale-out multi-client load (6 client VMs driving traffic
```

```
across ESXi hosts)
        # - replicas: 3: Baseline client load (3 client VMs for 1:1 client-to-Redis
node parity)
        replicas: 6
        variables:
          overrides:
          - name: vmClass
            value: guaranteed-4xlarge
          - name: osConfiguration
            value:
              tuned:
                active:
                - redis-optimized-profile
                profiles:
                  redis-optimized-profile:
                    profileRef:
                      name: redis-optimized-profile
        metadata:
          annotations:
            run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu,os-version=24.04
    variables:
    - name: vmClass
      value: guaranteed-large
    - name: storageClass
      value: vsan-esa-auto-raid-policy
    - name: vsphereOptions
      value:
        persistentVolumes:
          defaultStorageClass: vsan-esa-auto-raid-policy
```

redis-enterprise-cluster.yaml

```
apiVersion: app.redislabs.com/v1
kind: RedisEnterpriseCluster
metadata:
  name: rec-vks
  namespace: redis-enterprise
spec:
  # Validated Cluster Node Sizing:
  # - nodes: 5: Scale-out, multi-tenant, and online expansion validation (5-node
StatefulSet)
  # - nodes: 3: Baseline low-latency SLA and 90-minute endurance validation (3-node
StatefulSet)
  nodes: 5
  redisEnterpriseNodeResources:
    limits:
      cpu: "14500m"
      memory: "100Gi"
    requests:
      cpu: "14500m"
      memory: "100Gi"
  persistentSpec:
    enabled: true
    storageClassName: "vsan-esa-auto-raid-policy-latebinding"
  servicesRiggerSpec:
    databaseServiceType: cluster_ip

  # Scheduled onto redis-node-pool nodes
  nodeSelector:
vks.vmware.com/nodepool: redis-node-pool
  # ── Optional: Physical Failure-Domain & Rack Awareness ──
  # Specifies the Kubernetes node label key used by the Operator to assign
  # Redis Enterprise nodes to physical rack/host failure domains.
  # Redis uses this rack assignment to ensure primary and replica shards
  # never reside on the same physical ESXi host.
  rackAwarenessNodeLabel: topology.example.com/esxi-host
  # ── Optional: Custom Scheduling Anti-Affinity Override ──
  # By default, enabling rackAwarenessNodeLabel instructs the Operator to inject
  # hard (required) anti-affinity per rack. If running 5 Redis pods across 3 physical
  # ESXi hosts (a 2/2/1 distribution), a hard 1-pod-per-rack rule would cause
  # 2 pods to remain unschedulable (Pending).
  #
  # This custom override solves the constraint:
  # 1. requiredDuringScheduling on 'kubernetes.io/hostname':
  #    Guarantees strict 1 pod per VM (no co-located pods on the same worker VM).
  # 2. preferredDuringScheduling on 'topology.example.com/esxi-host':
  #    Soft-spreads pods across physical ESXi hosts with high priority (weight 100).
  podAntiAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchLabels:
            app: redis-enterprise
            redis.io/cluster: rec-vks
            redis.io/role: node
        topologyKey: kubernetes.io/hostname
    preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchLabels:
```

```
              app: redis-enterprise
              redis.io/cluster: rec-vks
              redis.io/role: node
          topologyKey: topology.example.com/esxi-host
license:|
----- LICENSE START -----
Xxx
 ----- LICENSE END -----
```

redb-cache-only.yaml

```
apiVersion: app.redislabs.com/v1alpha1
kind: RedisEnterpriseDatabase
metadata:
  name: redb-cache-only
  namespace: redis-enterprise
  labels:
    scenario: caching-only
    persistence: none
spec:
  # ── Memory ──
  # Database memory quota: 70 GB. Each Redis pod requests 100 GB.
  # Budget separately for replicas, persistence work, and management overhead.
  memorySize: 70GB
  # ── Sharding ──
  # Enable OSS Cluster API for client-side shard awareness.
  # The Enterprise proxy routes traffic to the correct shard transparently.
  ossCluster: true
  # Primary shard examples (one replica per primary):
  # - 5 or 10 primaries on 5 nodes
  # - 6 or 9 primaries on 3 nodes
  # Choose the count for the intended workload; these are separate configurations.
  shardCount: 10
  shardsPlacement: sparse
  # ── Proxy ──
  # Route traffic through all primary shards when ossCluster is enabled.
  proxyPolicy: all-master-shards
  # ── Replication ──
  # Even without persistence, replication provides HA. If a primary shard
  # fails, a replica can be promoted; recovery depends on replication state.
  replication: true
  # ── Persistence ──
  # DISABLED — this is a pure caching scenario.
  persistence: disabled
  # ── Security ──
  tlsMode: enabled

  # ── Cluster Reference ──
  redisEnterpriseCluster:
name: rec-vks
```

redb-cache-aof.yaml

```
apiVersion: app.redislabs.com/v1alpha1
kind: RedisEnterpriseDatabase
metadata:
  name: redb-cache-aof
  namespace: redis-enterprise
  labels:
    scenario: caching-aof
    persistence: aof
spec:
  # ── Memory ──
  memorySize: 70GB

  # ── Sharding ──
  ossCluster: true
  # Primary shard examples (one replica per primary):
  # - 5 or 10 primaries on 5 nodes
  # - 6 or 9 primaries on 3 nodes
  # Choose the count for the intended workload; these are separate configurations.
  shardCount: 10

  shardsPlacement: sparse
  # ── Proxy ──
  proxyPolicy: all-master-shards
  # ── Replication ──
  # With AOF + replication, the REPLICA handles persistence writes.
  # This is a key Redis Enterprise optimization — the primary serves
  # requests without AOF write overhead, while the replica ensures
  # durability. This architecture provides both HA and persistence
  # without the primary paying the I/O penalty.
  replication: true
  # rackAware: Instructs the cluster manager to place corresponding primary and
  #            replica shards in different rack domains (different physical ESXi
hosts).
  #rackAware: true
  # ── Persistence: AOF ──
  # Append-Only File with fsync every 1 second.
# Options: aofEverySecond, aofEveryWrite
  # aofEveryWrite changes durability and latency; validate the required recovery
behavior.
  persistence: aofEverySecond
  # ── Security ──
  tlsMode: enabled
  # ── Cluster Reference ──
  redisEnterpriseCluster:
name: rec-vks
```

redb-cache-rdb.yaml

Shard Examples: The reported 3-node configurations use 6 or 9 primary shards, with one replica per primary. The 5-node scale-out configuration uses 10 primary shards. The RDB manifest below selects 6 primaries to match the 3-node endurance configuration. The database memorySize setting is a capacity limit, distinct from the benchmark's populated dataset size.

```
apiVersion: app.redislabs.com/v1alpha1
kind: RedisEnterpriseDatabase
metadata:
  name: redb-cache-rdb
  namespace: redis-enterprise
  labels:
    scenario: caching-rdb
    persistence: rdb
spec:
  # ── Memory ──
  memorySize: 70GB

  # ── Sharding ──
  ossCluster: true
  # Primary shard examples (one replica per primary):
  # - 5 or 10 primaries on 5 nodes
  # - 6 or 9 primaries on 3 nodes
  # Choose the count for the intended workload; these are separate configurations.
  shardCount: 6
  shardsPlacement: sparse

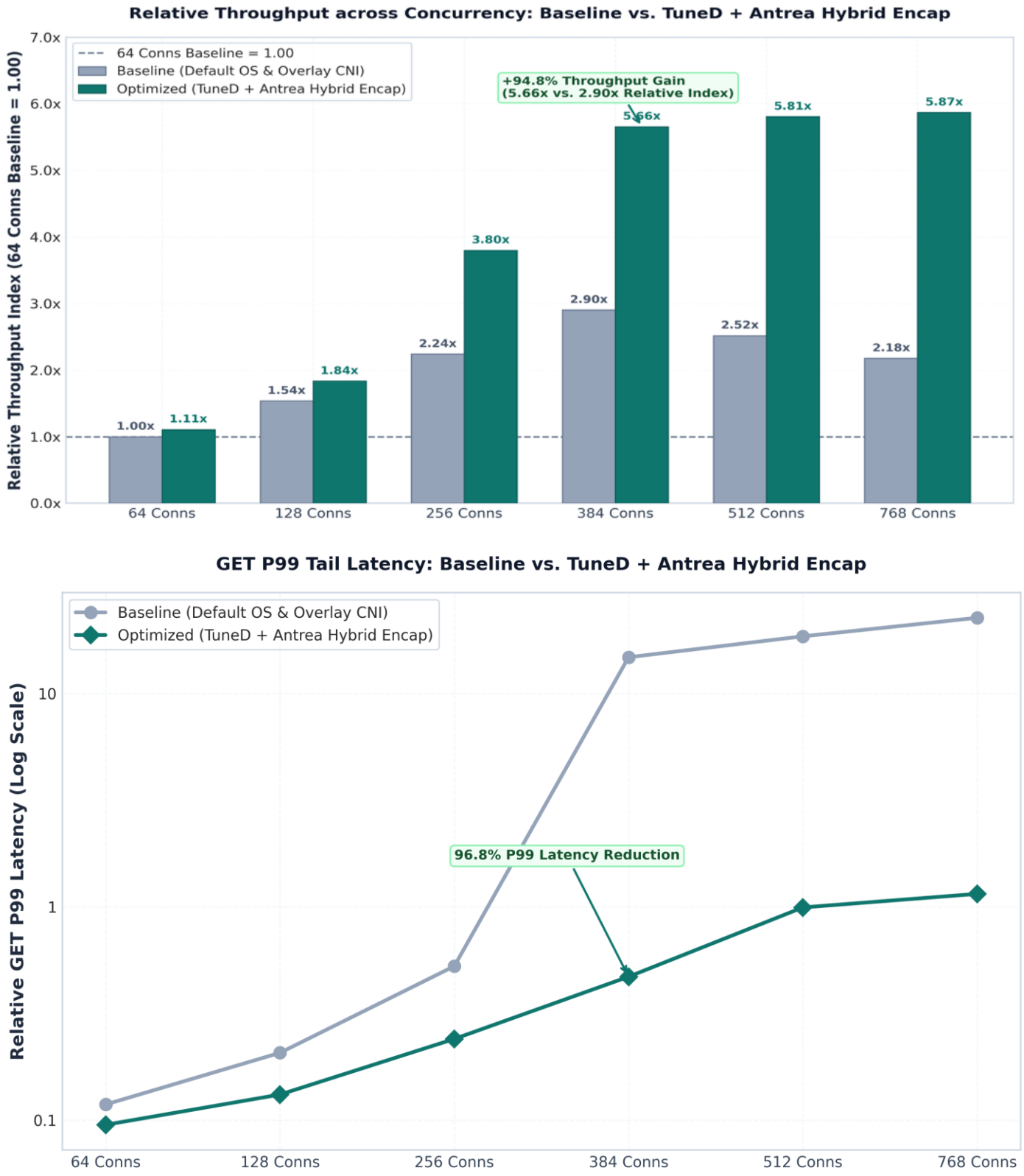
  # ── Proxy ──
  proxyPolicy: all-master-shards

  # ── Replication ──
  # With RDB, the replica handles persistence — avoids fork() impact
  # on the primary shard's request-serving performance.
  replication: true

  # ── Persistence: RDB ──
  # Point-in-time snapshots at configured intervals.
  # Options: snapshotEvery1Hour, snapshotEvery6Hours, snapshotEvery12Hours
  persistence: snapshotEvery1Hour
  # ── Security ──
  tlsMode: enabled

  # ── Cluster Reference ──
  redisEnterpriseCluster:
name: rec-vks
```

Figure A-1: Redis Enterprise Normalized Throughput and GET/SET P99 Tail Latencies across Concurrency: Baseline vs. TuneD + Antrea Hybrid Encap



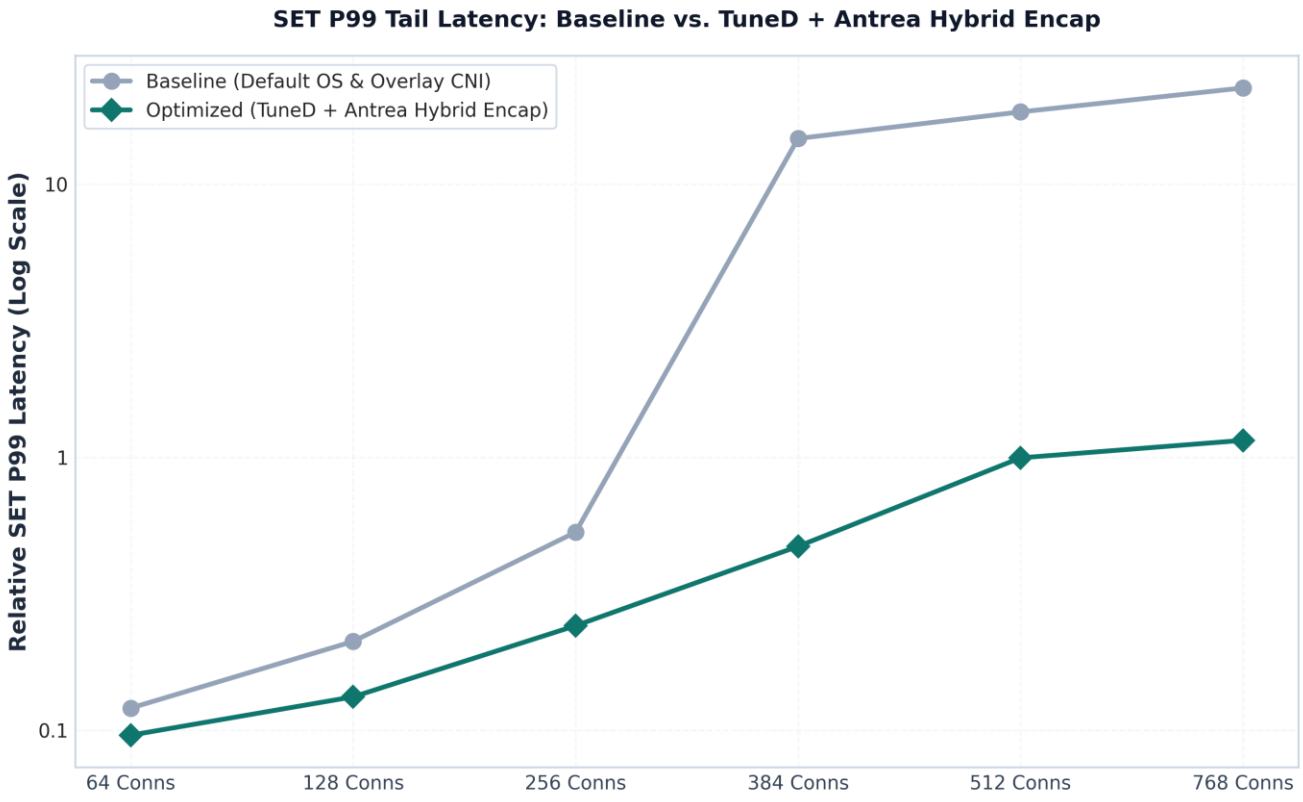

SET P99 Tail Latency: Baseline vs. TuneD + Antrea Hybrid Encap
Baseline (Default OS & Overlay CNI)
Optimized (TuneD + Antrea Hybrid Encap)
Relative SET P99 Latency (Log Scale)
10
1
0.1
64 Conns
128 Conns
256 Conns
384 Conns
512 Conns
768 Conns

Figure A-2: Redis Enterprise Scale-up and NUMA Trap: Normalized Throughput Comparison

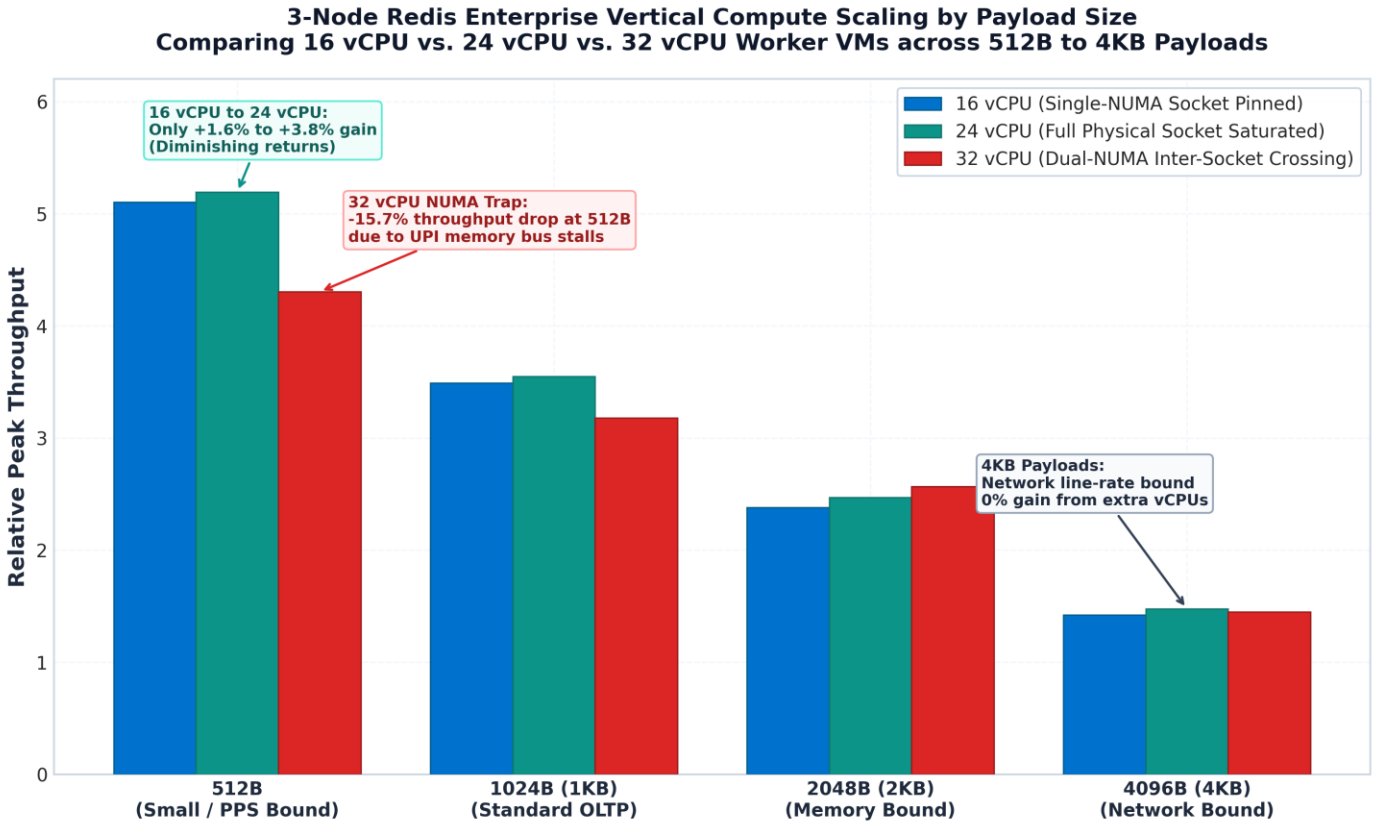


Figure A-3: Normalized P99 Tail Latency Escalation & The NUMA Scale-Up Trap

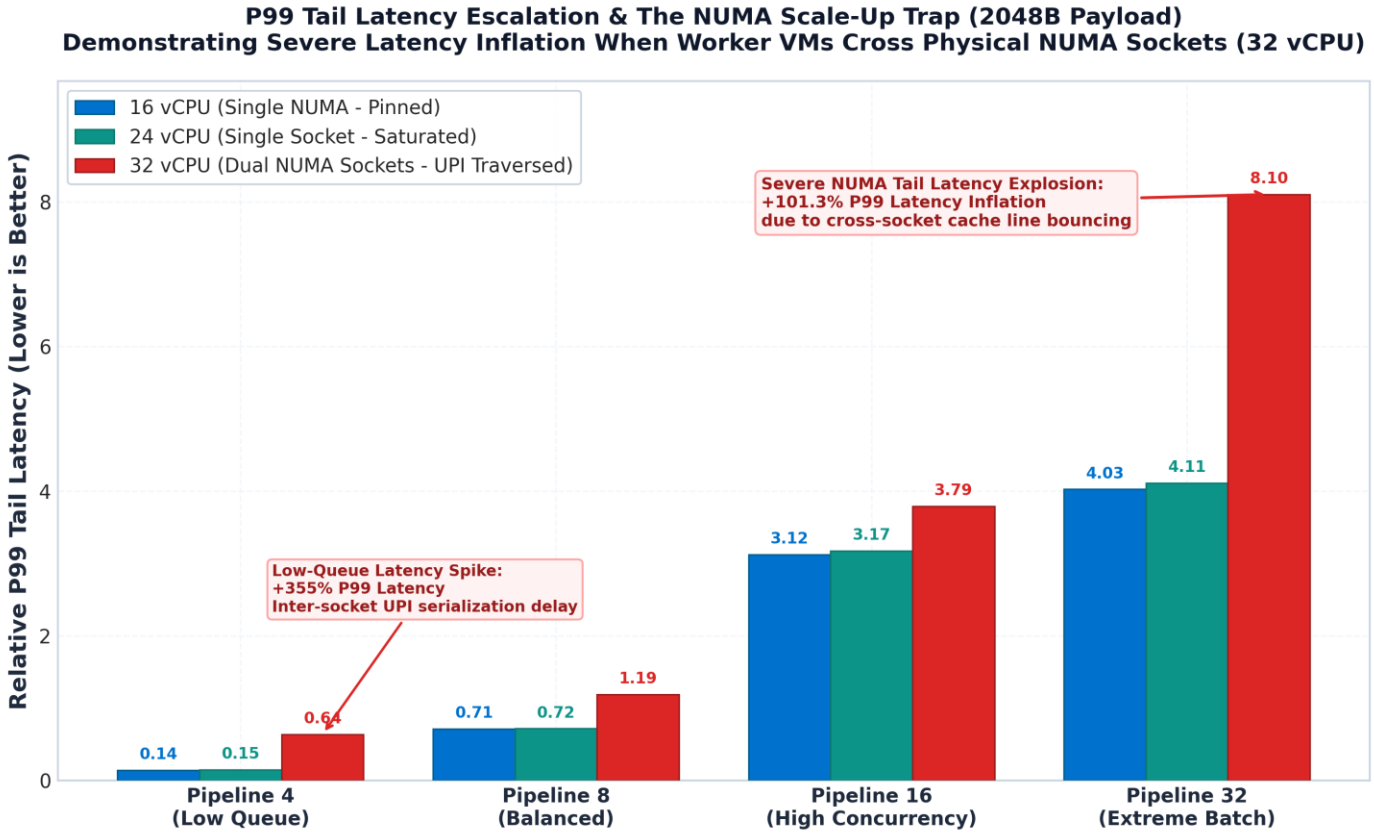
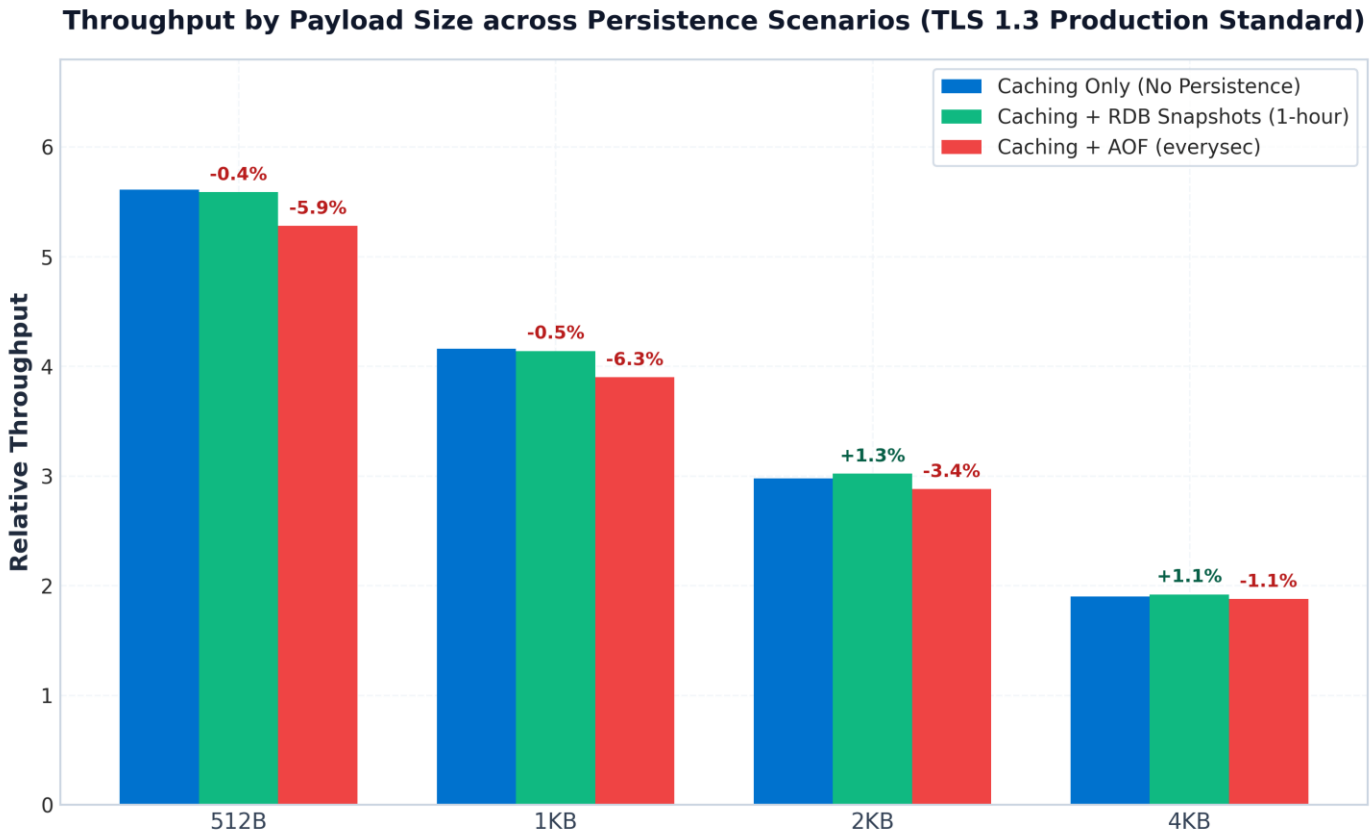

P99 Tail Latency Escalation & The NUMA Scale-Up Trap (2048B Payload)
Demonstrating Severe Latency Inflation When Worker VMs Cross Physical NUMA Sockets (32 vCPU)
16 vCPU (Single NUMA - Pinned)
24 vCPU (Single Socket - Saturated)
32 vCPU (Dual NUMA Sockets - UPI Traversed)
Relative P99 Tail Latency (Lower is Better)
0
2
4
6
8
Low-Queue Latency Spike:
+355% P99 Latency
Inter-socket UPI serialization delay
Severe NUMA Tail Latency Explosion:
+101.3% P99 Latency Inflation
due to cross-socket cache line bouncing
0.14
0.15
0.64
0.71
0.72
1.19
3.12
3.17
3.79
4.03
4.11
8.10
Pipeline 4
(Low Queue)
Pipeline 8
(Balanced)
Pipeline 16
(High Concurrency)
Pipeline 32
(Extreme Batch)

Figure A-4: Normalized Throughput and P99 Tail Latencies by Payload Size across Persistence Scenarios



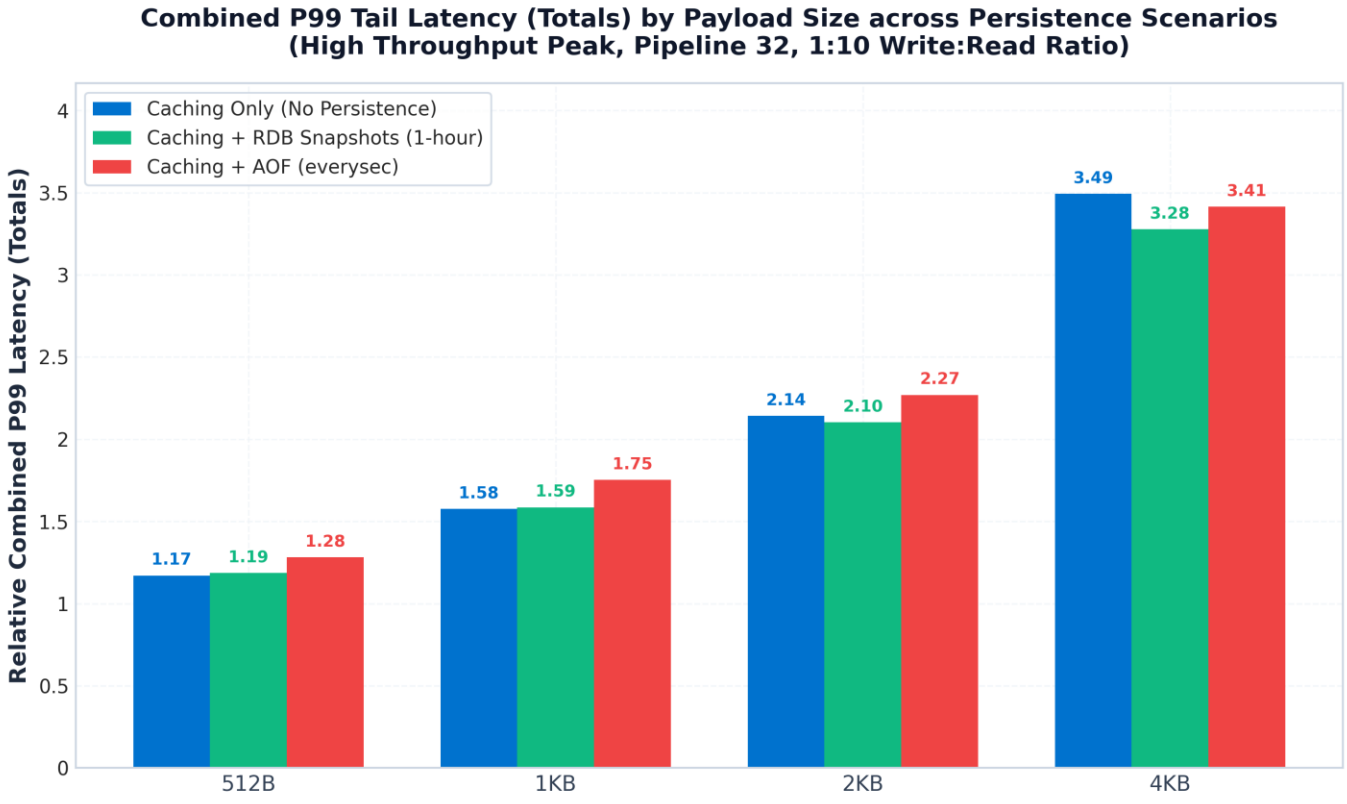


Figure A-5: Normalized Peak Throughput Comparison to evaluate the impact of Shard Density

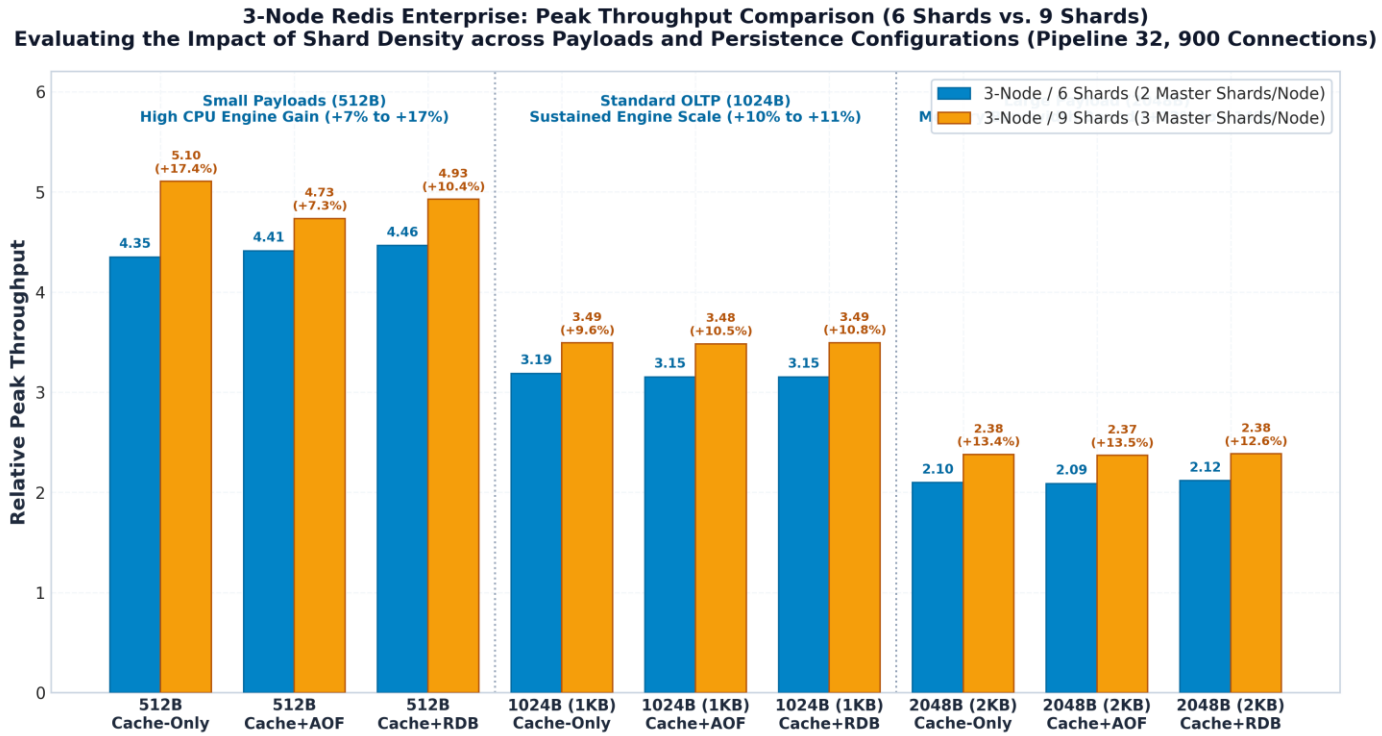


Figure A-6: Normalized Low Latency Comparison across shallow pipeline to evaluate the impact of Shard Density (Combined P99 Totals, 30–120 Conns per Client Pod)

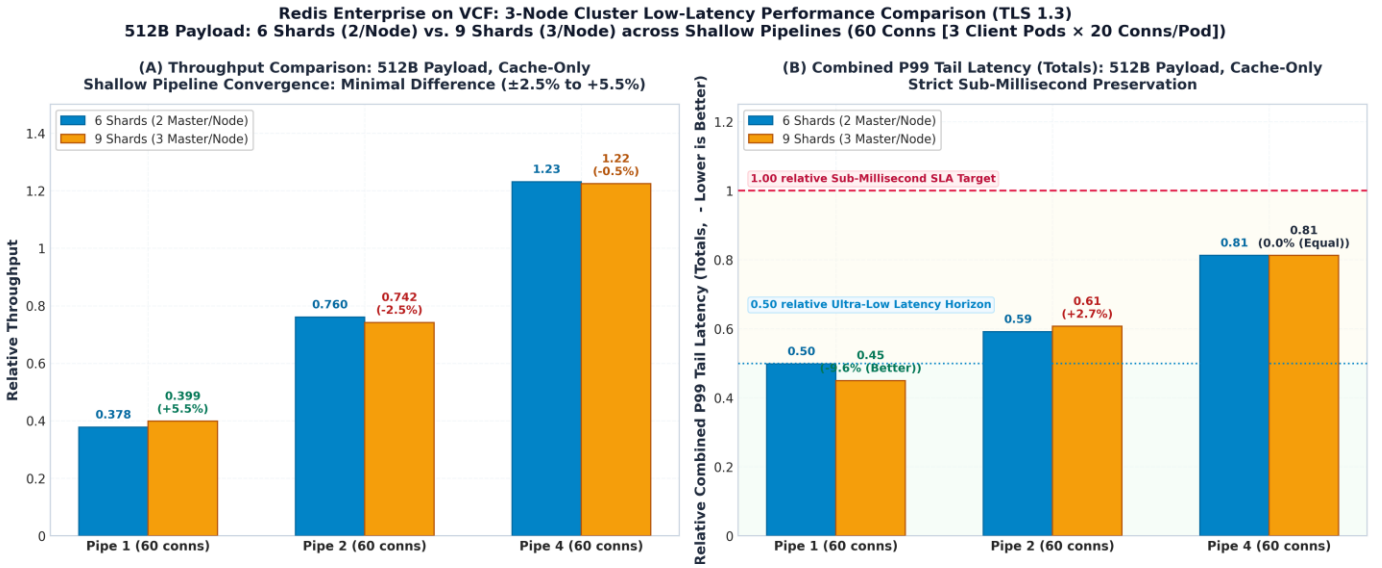


Figure A-7: Normalized Throughput and P99 Latency Comparison — Three Nodes / Six Primary Shards and Five Nodes / Ten Primary Shards

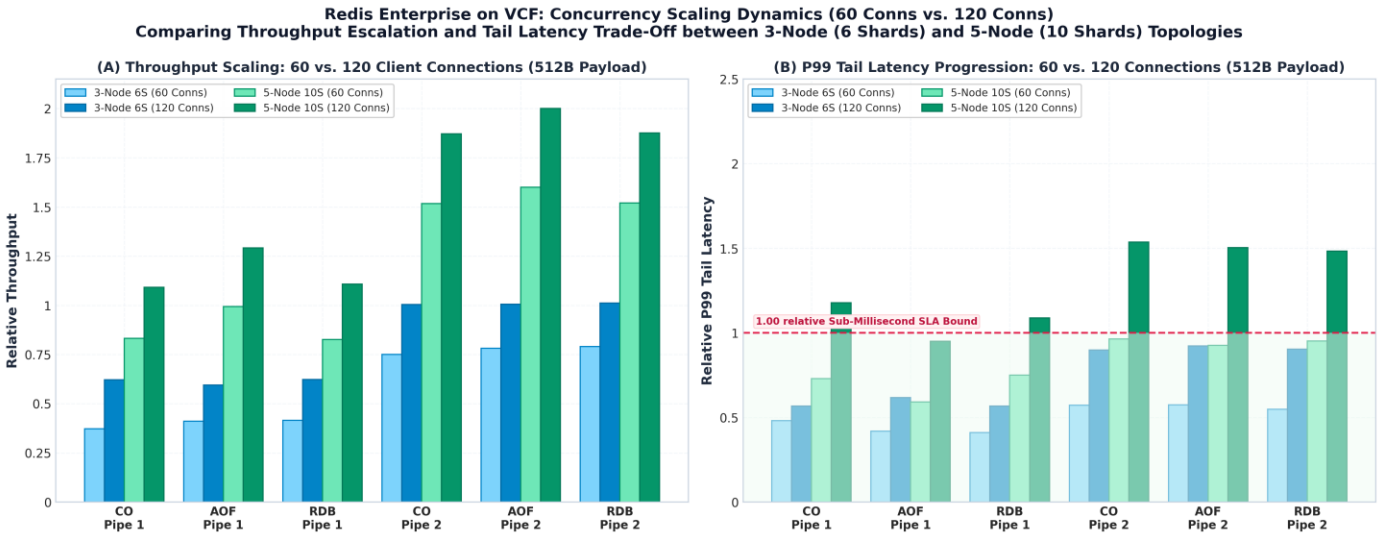


Figure A-8: Normalized Write-Ratio Sensitivity Impact Comparison: Ratio 1:10 vs. Ratio 3:7 across 3-Node and 5-Node Clusters

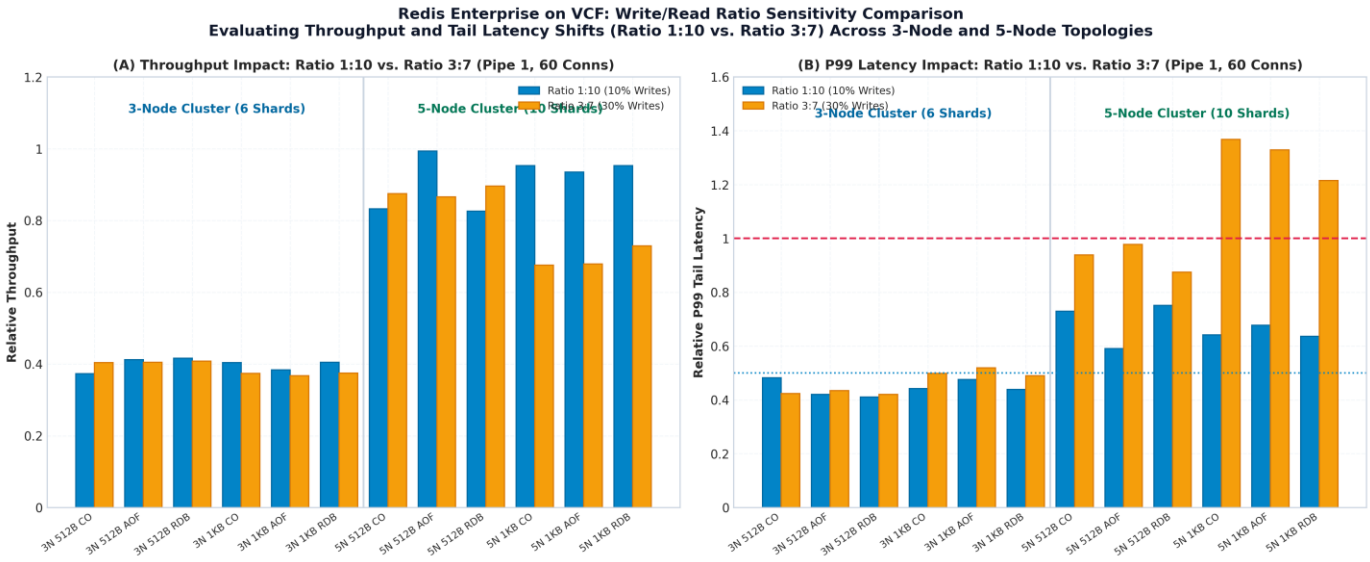
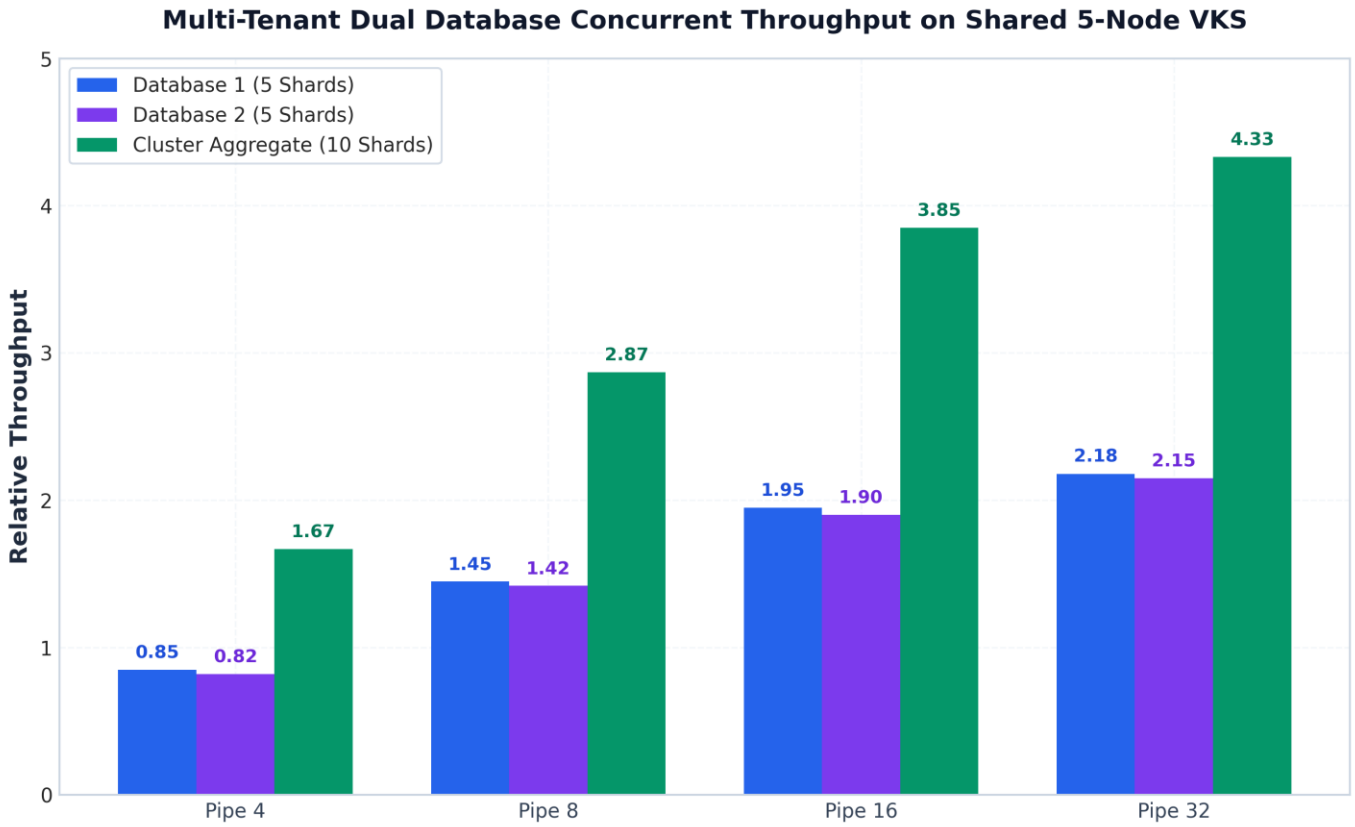
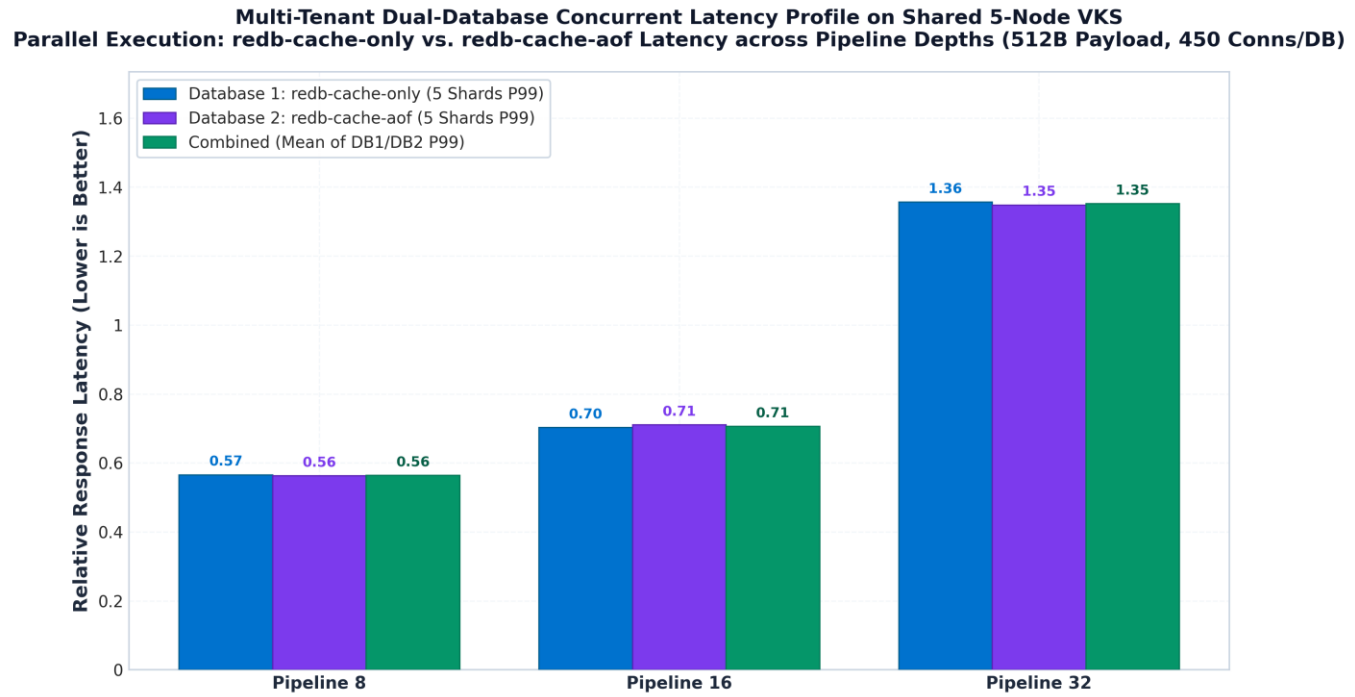


Figure A-9: Multi-Tenant Dual Database Normalized Throughput and P99 Tail Latency





Copyright © 2026 Broadcom. All rights reserved.
The term “Broadcom” refers to Broadcom Inc. and/or its subsidiaries. For more information, go to www.broadcom.com. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies. Broadcom reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design. Information furnished by Broadcom is believed to be accurate and reliable. However, Broadcom does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.
Item No: vmw-bc-wp-tech-uslet-word-2026; Oct-26