TECHNICAL WHITE PAPER: **October 2026**



KubeDB on vSphere Kubernetes Service on VMware Cloud Foundation

Reference Architecture

Table of Contents

Executive Summary 3
vSphere Kubernetes Service 4
KubeDB by AppsCode 4
Solution Architecture 7
VKS Cluster Configuration 9
Storage Design 11
Networking Design 12
High Availability and Placement 12
Security and Multi-Tenancy 12
Data Protection and Disaster Recovery 13
Monitoring and Day-2 Operations 13
Onboarding Existing Databases 14
Kubernetes Packages 15
Solution Validation 16
Install KubeDB on VKS 16
Deploy a Highly Available PostgreSQL Database 17
Validate KubeDB on VKS 19
Conclusion 24

Executive Summary

Enterprises want the speed of a cloud Database-as-a-Service (DBaaS) without giving up control of where their data lives, what it costs, or how it is governed. Hyperscaler-managed databases deliver self-service, but they tie critical data to one provider, make costs hard to predict, and are often not an option for regulated, sovereign, or air-gapped environments. Running databases by hand on Kubernetes avoids the lock-in but brings back slow, ticket-driven provisioning and fragile day-2 operations.

This reference architecture shows how **KubeDB by AppsCode** running on **vSphere Kubernetes Service (VKS)** in **VMware Cloud Foundation (VCF)** closes that gap. VKS supplies CNCF-conformant Kubernetes clusters that are provisioned, upgraded, and governed with the same vSphere tools, skills, and policies already used for virtual machines. KubeDB complements VKS by adding a Kubernetes-native DBaaS layer on top: production-grade PostgreSQL, MySQL, MariaDB, MongoDB, Redis, Kafka, Elasticsearch, Microsoft SQL Server, and many other engines are provisioned, scaled, upgraded, backed up, and recovered through declarative Kubernetes APIs. The result is a private-cloud DBaaS that runs on infrastructure the customer already owns, using vSAN for persistent storage and NSX for networking and isolation.

Containers provide packaging; KubeDB provides the database operating model. It turns databases into governed, automated platform services on VKS, so application teams get production-ready databases without becoming database administrators, and platform teams run more than 30 database engines with one set of tools, policies, and processes.

Benefits of using VKS with modern applications:

Lower TCO: With VKS organizations can reduce silos, leverage existing tools and skill sets without having to retrain staff and/or change existing processes. Utilizing unified lifecycle management across infrastructure components to stay up to date with the most recent patches and minimizing security risks.

Operational Simplicity: VKS is engineered for unparalleled operational simplicity, leveraging the familiarity of existing vSphere tools, skills, and workflows. This design philosophy significantly reduces the learning curve for IT teams and streamlines management processes. With VKS, organizations benefit from automated cluster provisioning, which accelerates deployment times and minimizes manual configuration errors. Furthermore, its robust capabilities extend to automated upgrades and comprehensive lifecycle management. This integrated approach ensures consistent operations, reduces overhead, and frees up valuable resources to focus on innovation rather than infrastructure maintenance.

Run and Manage Kubernetes at Scale: Effortlessly deploy and manage Kubernetes clusters at scale, leveraging a built-in, Cloud Native Computing Foundation (CNCF) certified Kubernetes distribution. VKS provides fully automated lifecycle management, streamlining operations from initial setup to ongoing maintenance and upgrades. This comprehensive approach ensures that organizations can harness the power of Kubernetes for their containerized applications with unparalleled efficiency and reliability, without the complexities typically associated with large-scale Kubernetes deployments.

Benefits of running KubeDB on VKS:

Consistent DBaaS wherever VKS runs: KubeDB delivers the same DBaaS experience on VKS clusters in data centers, edge sites, sovereign regions, and air-gapped environments. The same manifests work on every cluster, giving one database operating model across the VCF estate.

Automated database lifecycle: Provisioning, high availability, failover, version upgrades, horizontal and vertical scaling, volume expansion, TLS, and reconfiguration are expressed as Kubernetes resources and executed by the KubeDB operators. This replaces runbooks and one-off scripts with repeatable, auditable workflows.

Built-in data protection: KubeDB integrates with KubeStash for scheduled, application-consistent backups to S3-compatible object storage, with point-in-time recovery for supported engines. The same tool backs up the Kubernetes manifests of namespaces or the whole cluster, and vSphere CSI volume snapshots add fast volume-level rollback.

Governance and multi-tenancy: vSphere Namespaces, VM classes, storage policies, and quotas set the guardrails for each team. Inside the cluster, Kubernetes RBAC, namespaces, and network policies give developers secure self-service access to the databases they own and nothing else.

Resilience and recovery you can trust: Automated failover, self-healing, encrypted incremental backups, point-in-time recovery, and multi-zone placement are standardized across every database engine, rather than rebuilt for each one.

Extends VKS for modernization and AI: Courier brings existing databases under KubeDB management on VKS with minimal downtime whenever a team chooses to. Vector databases such as Qdrant, Milvus, and Weaviate run on VKS alongside the AI and application workloads that use them.

Predictable cost and existing storage: Databases consume vSAN capacity through standard Kubernetes StorageClasses, so organizations keep full data ownership and choose the performance, resilience, and cost profile for each workload.

vSphere Kubernetes Service

vSphere Kubernetes Service (VKS) is the 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.

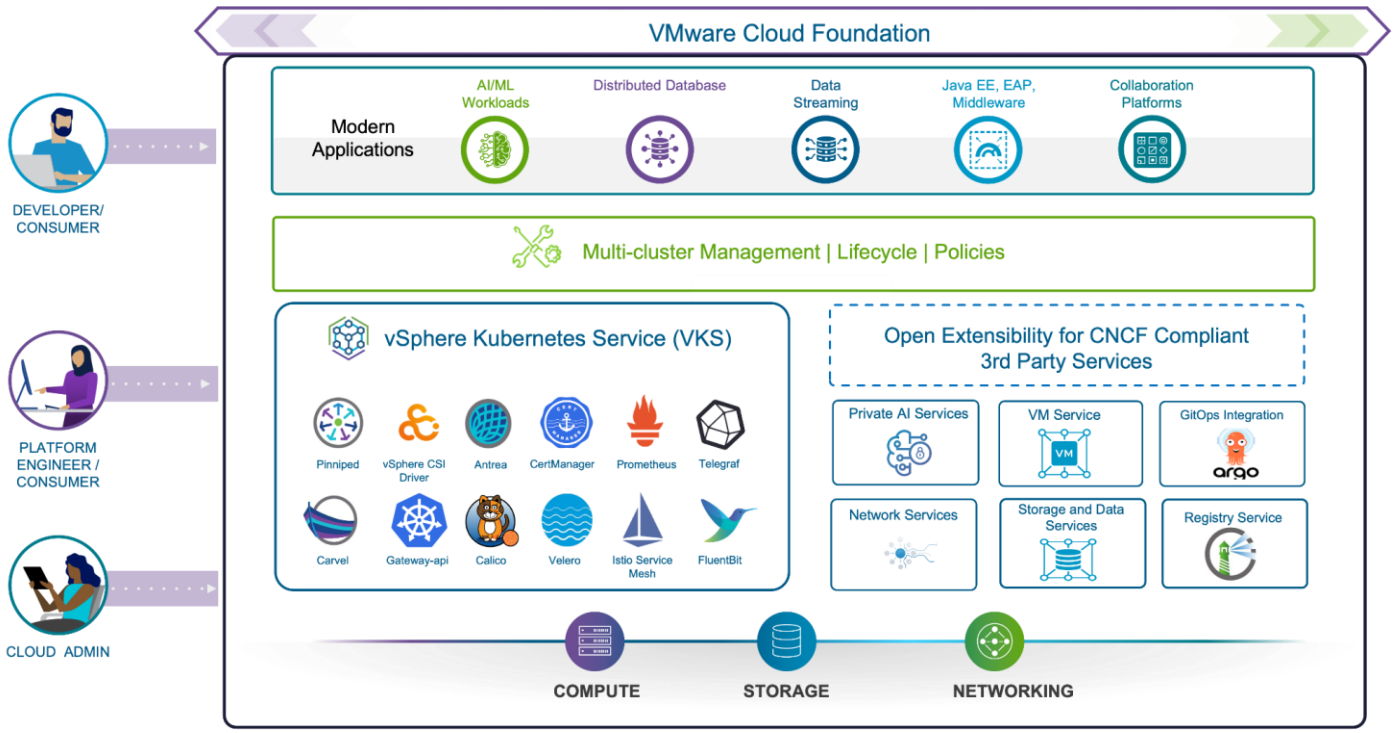


Figure 1: VKS on VCF Ecosystem

KubeDB by AppsCode

KubeDB by AppsCode is a Kubernetes-native Database-as-a-Service platform. It extends the Kubernetes API with a custom resource for each supported database engine and a set of operators that automate the full database lifecycle. Platform teams use KubeDB to standardize and govern production-ready databases across clusters and environments, while developers get self-service access to databases through the same Kubernetes workflows they already use for applications: kubectl, Helm, GitOps tools such as Argo CD and Flux, or the KubeDB web console.

KubeDB supports more than 30 SQL, NoSQL, caching, search, streaming, and vector databases from a single control plane, including PostgreSQL, MySQL, MariaDB, Percona XtraDB, Microsoft SQL Server, Oracle, MongoDB, Redis, Memcached, Kafka, RabbitMQ, Elasticsearch, OpenSearch, Solr, ClickHouse, Druid, SingleStore, Cassandra, Neo4j, Qdrant, Milvus, and Weaviate. Key capabilities used in this reference architecture are:

- **Provisioning and high availability**: standalone, replicated, and clustered or sharded topologies with automatic leader election and failover.

- **Day-2 operations through OpsRequests**: version updates, horizontal and vertical scaling, online volume expansion, reconfiguration, TLS rotation, credential rotation, and restarts, each executed as a tracked Kubernetes object.
- **Autoscaling**: compute and storage autoscaling and a recommendation engine for right-sizing.
- **Data protection**: scheduled backups, restores, and point-in-time recovery through KubeStash, written to S3-compatible object storage with encryption and retention policies.
- **Security**: TLS everywhere through cert-manager, auto-generated or user-supplied credentials, and optional network policies for every database.
- **Observability**: built-in Prometheus exporters, ServiceMonitor integration, and Grafana dashboards for each engine.
- **Migration and cloning**: Courier migrates live, non-KubeDB databases into KubeDB-managed instances, and production data can be cloned safely for development and test.

KubeDB is built on the Kubernetes operator pattern, which encodes expert operational knowledge in software. A team declares the desired state, for example a three-node PostgreSQL cluster with TLS, and the operator works out how to reach it: it provisions the pods, persistent volumes, and Secrets, configures replication, and then watches the database continuously. If a pod fails or the configuration drifts, the operator restores the declared state without human intervention. Replication setup, failover, and upgrade sequences live in the operator rather than in runbooks, and day-2 changes are made by applying a YAML object rather than running manual scripts.

Run as a Database-as-a-Service on VKS, this gives each audience what it needs:

- **Developer self-service**: any supported database is provisioned through a single Kubernetes object, in seconds through GitOps rather than through a ticket queue.
- **Platform team control**: policies, quotas, approved versions, and governance are enforced centrally.
- **Operations automation**: backups, high availability, and upgrades run without manual steps.
- **Full portability**: the same interface works on-premises, in the cloud, at the edge, and in air-gapped sites.

Table 1: How KubeDB Complements VKS

Area	VKS on VMware Cloud Foundation provides	KubeDB adds
Platform	CNCF-conformant Kubernetes clusters with automated provisioning, upgrades, and lifecycle management	Database operators for 30+ engines running on those clusters
Storage	vSAN storage policies exposed as StorageClasses through vSphere CSI	Per-member persistent volumes, online volume expansion, and storage autoscaling
Networking	NSX and VPC networking, load balancing, and Antrea	Primary and standby Services and per-database network policies
Security	vSphere Namespaces, vCenter Single Sign-On, and cert-manager	Database TLS, credential management and rotation, and RBAC on database resources
Availability	vSphere HA, vSphere Zones, and automatic node replacement	Database replication, automatic failover, and self-healing
Data protection	vSphere CSI volume snapshots	Database-aware, encrypted backups with point-in-time recovery through KubeStash
Operations	Supervisor services such as Argo CD and Harbor, and VKS standard packages	Declarative day-2 OpsRequests, autoscaling, and per-engine monitoring dashboards

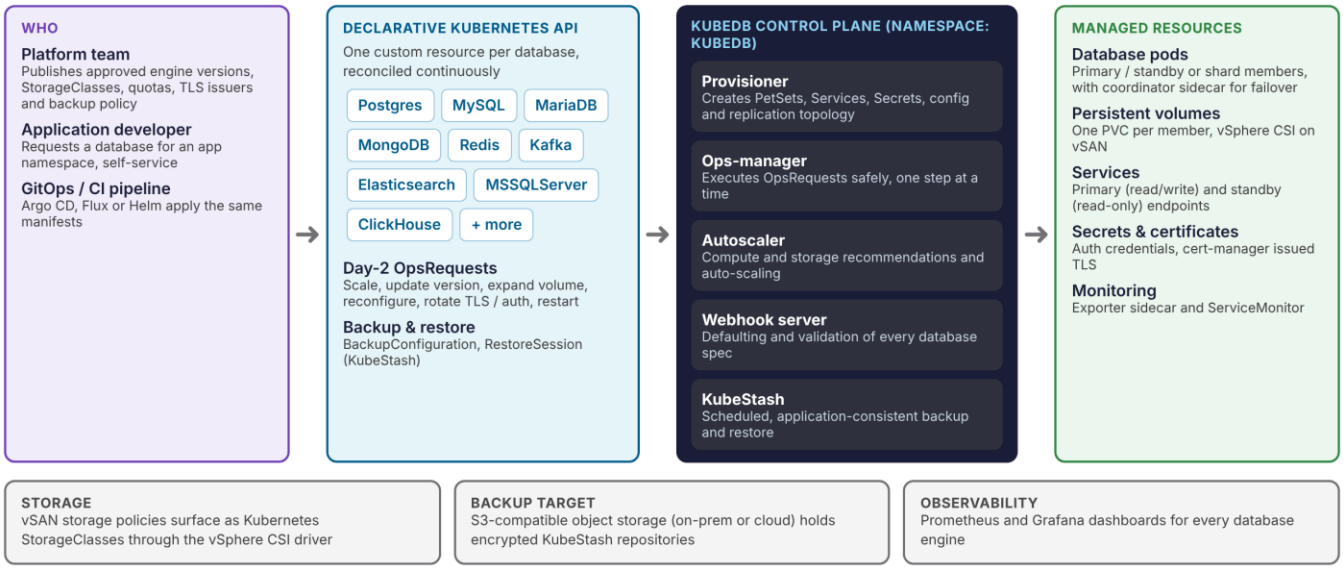


Figure 2: KubeDB Database-as-a-Service Workflow

Figure 2 shows the DBaaS workflow. A platform team publishes the approved engine versions, StorageClasses, certificate issuers, and backup policies. A developer or pipeline then declares the database it needs as a Kubernetes resource. The KubeDB provisioner creates and continuously reconciles the database pods, persistent volumes, Services, Secrets, and certificates. Day-2 changes are submitted as OpsRequests that the ops-manager executes one safe step at a time, and KubeStash runs the scheduled backups.

Solution Architecture

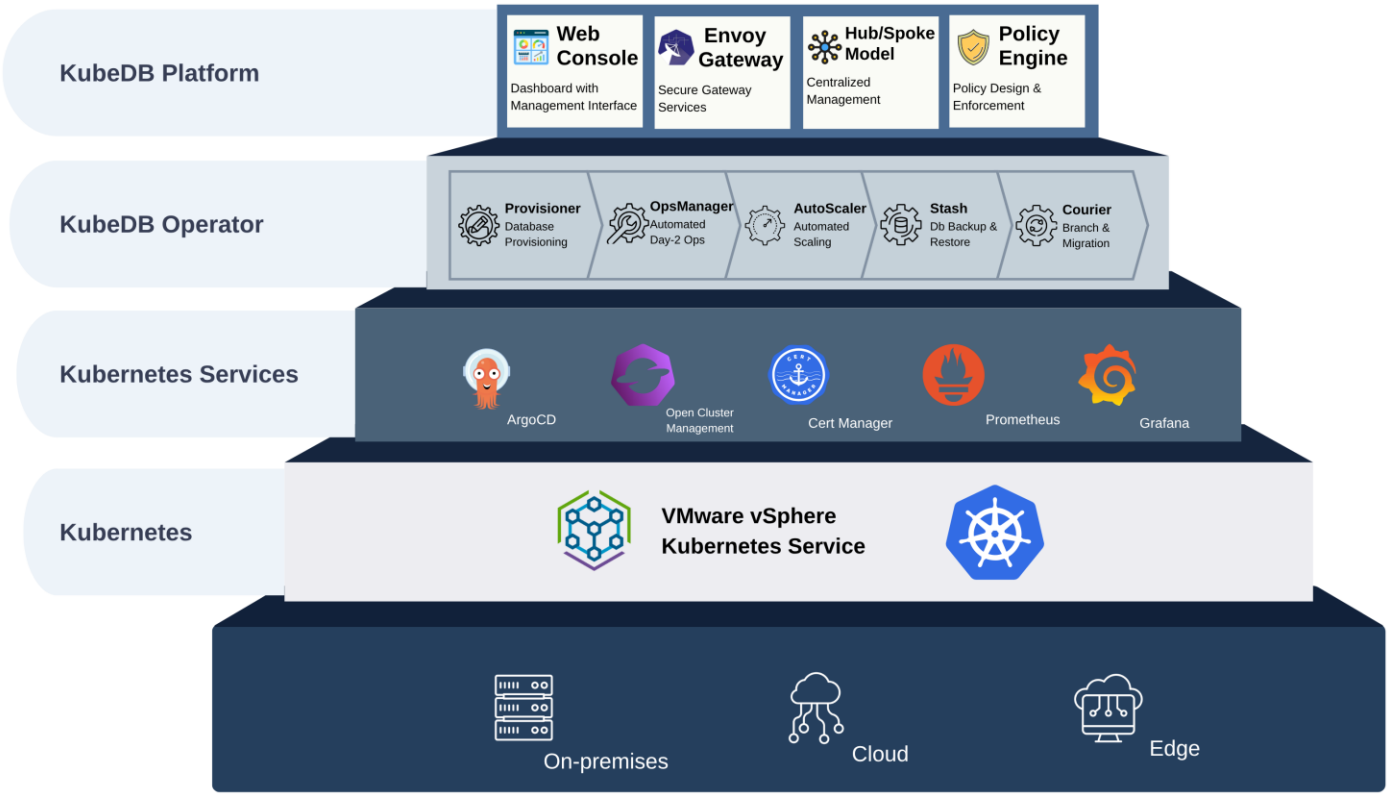


Figure 3: KubeDB Platform Architecture on VKS

Figure 3 shows the KubeDB platform as a stack of layers. At the base, the same platform runs on-premises, in the cloud, and at the edge. The Kubernetes layer is VMware vSphere Kubernetes Service, which provides the CNCF-conformant Kubernetes clusters that KubeDB manages through standard Kubernetes APIs.

Above Kubernetes, a set of common services supports the platform: **Argo CD** for GitOps delivery of database manifests, **Open Cluster Management** for registering and governing many clusters, **cert-manager** for TLS certificates, and **Prometheus** and **Grafana** for metrics and dashboards. The **KubeDB Operator** layer automates the database lifecycle with five components: the **Provisioner** creates databases, the **OpsManager** runs automated day-2 operations, the **AutoScaler** scales compute and storage, **Stash** (KubeStash) performs backup and restore, and **Courier** migrates live databases into KubeDB-managed instances with minimal downtime.

At the top, the **KubeDB Platform** layer delivers the self-service experience: a **Web Console** with dashboards and a management interface, an **Envoy Gateway** that provides secure gateway services for database access, a **Hub/Spoke model** that centralizes management of databases across many VKS and other Kubernetes clusters, and a **Policy Engine**, an optional feature built on Gatekeeper or Kyverno that platform teams can enable to enforce rules such as allowed versions, sizes, and storage classes.

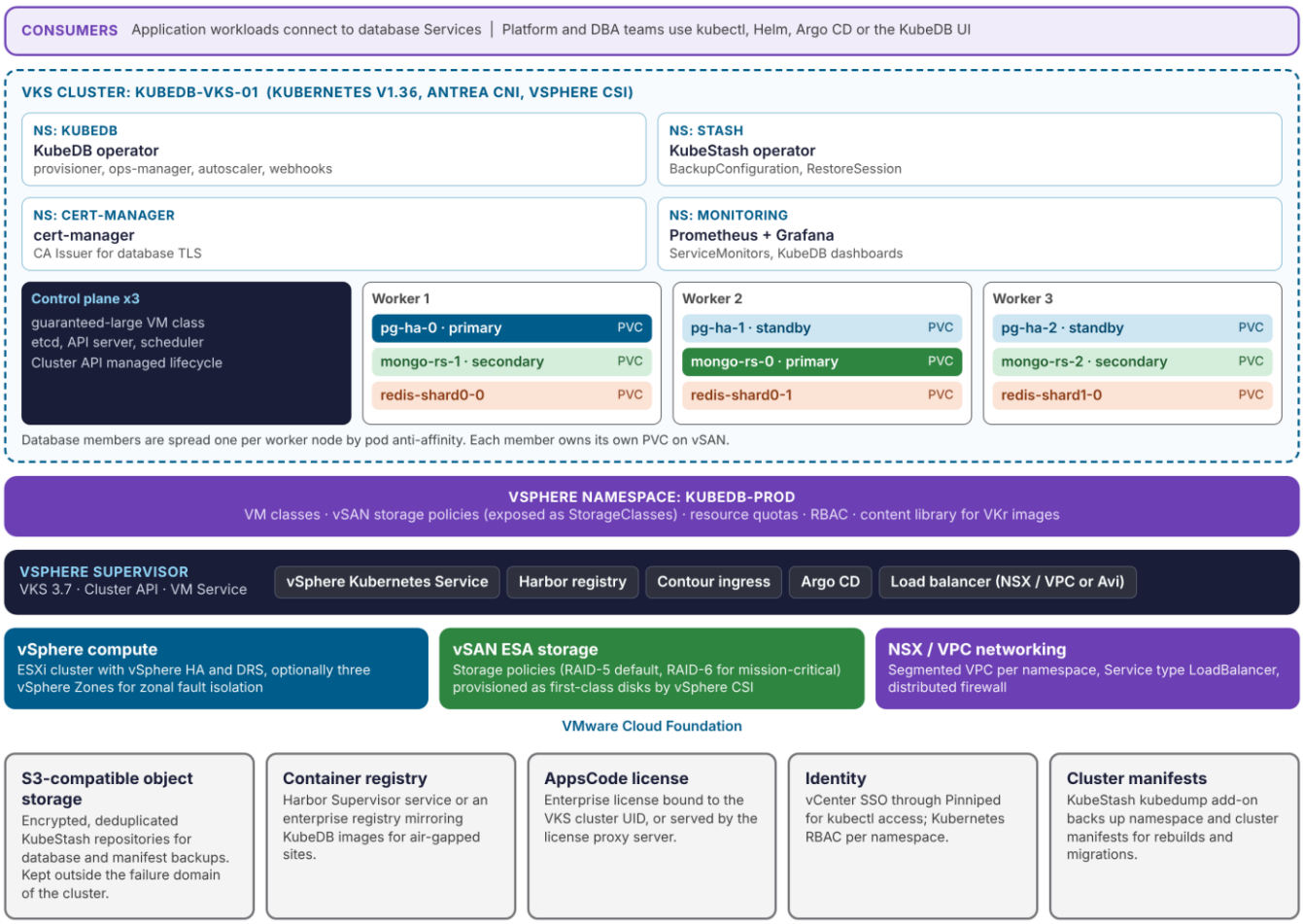


Figure 4: KubeDB Deployment Topology on VKS

Figure 4 shows how the solution is deployed on VMware Cloud Foundation. VMware Cloud Foundation provides compute (vSphere with HA and DRS), storage (vSAN Express Storage Architecture), and networking (NSX with VPCs). The vSphere Supervisor runs on the vSphere cluster and exposes a vSphere Namespace, **kubedb-prod**, which carries the VM classes, storage policies, quotas, and permissions for the database platform. A VKS cluster, **kubedb-vks-01**, is created in that namespace. KubeDB, KubeStash, cert-manager, and the monitoring stack run in their own namespaces inside the VKS cluster, and each database runs in an application namespace with one member per worker node and one vSAN -backed persistent volume per member.

The Supervisor can also run optional services, such as the Harbor container registry, the Contour ingress controller, and the Argo CD GitOps utility. For more information on vSphere Supervisor services and installation, visit: https://techdocs.broadcom.com/us/en/vmware-cis/vcf/vsphere-supervisor-services-and-standalone-components/latest.html

This control plane is responsible for the automated deployment and lifecycle management of CNCF -compliant Kubernetes clusters via the vSphere Kubernetes Service (VKS) via Cluster API. The specific version of VKS used in this white paper is 3.7, running on vCenter Server 9.0.2 in VMware Cloud Foundation 9.0.

Table 2: Installed Supervisor Components

Component	Version	Notes
vSphere Kubernetes Service	3.7	Allows for CNCF K8s clusters

vCenter Server	9.0.2 (build 25148086)	VMware Cloud Foundation 9.0

Optional Supervisor services complement the design: **Harbor** can host a private mirror of the KubeDB images for disconnected sites, **Argo CD** can deliver database manifests through GitOps, and **Contour** can publish the KubeDB web console. Backups are written to an S3-compatible object store that sits outside the failure domain of the VKS cluster.

The example manifests, Helm values, and supporting configuration referenced throughout this paper are available in the accompanying repository:

https://github.com/vmware/vks-validated-solutions/tree/main/KubeDB

VKS Cluster Configuration

The recommended production cluster consists of three control plane nodes and three worker nodes. Storage is provided by vSAN Express Storage Architecture (ESA), using the default RAID 5 storage policy. The vSphere CSI driver exposes this policy to an adjacent Kubernetes storage class, **vsan-esa-default-policy-raid5**.

Worker nodes use the *best-effort-2xlarge* VM class and control plane nodes use the guaranteed-large VM class.

The validation in this paper ran on a smaller cluster with the same Kubernetes release, ClusterClass, and storage policy: one control plane node (best-effort-medium, 2 vCPU / 8 GB) and three worker nodes (vks-8-32-class, a custom 8 vCPU / 32 GB VM class, each with a 100 GiB containerd volume), in a single vSphere Zone on NSX VPC networking. With the full KubeDB Platform stack installed, the single best-effort-medium control plane node ran at 95% memory or more, so use the recommended guaranteed-large class with three replicas in production.

Table 3: Kubernetes Cluster Configuration Summary

Component	Configuration	Notes
Kubernetes Control Plane	3 Replicas — 4x vCPU 16GB RAM vSAN RAID 5	VM Class: “guaranteed-large” Storage Class: “vsan-esa-default-policy-raid5” Validated with 1 replica, VM Class “best-effort-medium”
Kubernetes Worker Nodes	3 Replicas — 8x vCPU 64GB RAM vSAN RAID 5	VM Class: “best-effort-2xlarge” Storage Class: “vsan-esa-default-policy-raid5” Validated with VM Class “vks-8-32-class” (8 vCPU / 32 GB)
Kubernetes Release	v1.36.2+vmware.2-vkr.3	
OS Image	Ubuntu 24.04.4 LTS	Kernel version 6.17.0-35-generic containerd 2.3.2+vmware.3-fips

The VKS cluster configuration is defined declaratively in the YAML file below. As previously outlined, applying this manifest to the Supervisor triggers the creation of the cluster, which is then managed and reconciled by the Cluster API.

```
# vks-cluster.yaml

apiVersion: cluster.x-k8s.io/v1beta2
kind: Cluster
metadata:
```

```
  name: kubedb-vks-01
  namespace: kubedb-prod
spec:
  clusterNetwork:
    pods:
      cidrBlocks: ["192.168.0.0/16"]
    services:
      cidrBlocks: ["10.96.0.0/12"]
    serviceDomain: cluster.local
  topology:
    classRef:
      name: builtin-generic-v3.7.0
    version: v1.36.2---vmware.2-vkr.3
    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: kubedb-vks-01-np-1
          replicas: 3
          metadata:
            annotations:
              run.tanzu.vmware.com/resolve-os-image: os-name=ubuntu,os-version=24.04
          variables:
            overrides:
              - name: vmClass
                value: best-effort-2xlarge
              - name: volumes
                value:
                  - name: containerd
                    mountPath: /var/lib/containerd
                    storageClass: vsan-esa-default-policy-raid5
                    capacity: 100Gi
    variables:
      - name: vmClass
        value: guaranteed-large
      - name: storageClass
        value: vsan-esa-default-policy-raid5
      - name: vsphereOptions
        value:
          persistentVolumes:
            defaultStorageClass: vsan-esa-default-policy-raid5
      - name: kubernetes
        value:
          certificateRotation:
            enabled: true
            renewalDaysBeforeExpiry: 90
          security:
            podSecurityStandard:
              enforce: baseline
```

For production database clusters, size worker nodes so that each node can host one member of every highly available database with headroom for a failover. Use **guaranteed** VM classes for workers that host databases with strict latency targets, so that vCPU and memory are fully reserved, and keep at least as many worker nodes as the largest database replica count so pod anti-affinity can place every member on a different node. Separate node pools, selected with node labels and tolerations in the KubeDB `podTemplate`, can isolate database workloads from general application workloads.

Storage Design

Persistent storage is provided by vSAN through the vSphere Container Storage Interface (CSI) driver. Each vSAN storage policy assigned to the vSphere Namespace appears in the VKS cluster as a Kubernetes StorageClass. When KubeDB creates a database, it creates one PersistentVolumeClaim per member from the StorageClass named in `spec.storage.storageClassName`, and vSphere CSI provisions each claim as a Cloud Native Storage (CNS) volume backed by a vSAN object. Platform administrators can see every database volume, its policy compliance, and the Kubernetes objects that use it in vCenter under **Cloud Native Storage > Container Volumes**.

Recommendations:

- Use the **late-binding** (`WaitForFirstConsumer`) variant of each StorageClass when the Supervisor spans multiple vSphere Zones, so that each volume is created in the zone where its pod is scheduled. VKS creates this variant automatically, for example vsan-esa-default-policy-raid5-latebinding.
- Keep `allowVolumeExpansion: true` on database StorageClasses. KubeDB volume expansion OpsRequests and storage autoscaling rely on online expansion through vSphere CSI.
- Database replication and vSAN protection are complementary: replication protects availability of the service, while the vSAN policy protects the durability of each replica and avoids a full database re-sync after a single disk or host failure.
- Install the CSI snapshot controller (part of VKS) and a VolumeSnapshotClass when volume snapshots are used for fast local rollback.

Table 4: Storage Policy Guidance for Database Workloads

Workload	vSAN policy	StorageClass / KubeDB setting	Rationale
General-purpose databases (PostgreSQL, MySQL, MongoDB replica sets)	vSAN ESA RAID-5, FTT=1 (default)	vsan-esa-default-policy-raid5, storageType: Durable	Space efficient. On vSAN ESA, RAID-5 delivers performance comparable to RAID-1.
Mission-critical or regulated databases	vSAN ESA RAID-6, FTT=2 (six or more hosts)	Dedicated StorageClass per policy	Survives two simultaneous host or device failures without a database rebuild.
Self-replicating streaming and search engines (Kafka, Elasticsearch, Redis Cluster)	vSAN ESA RAID-5, FTT=1	Same as above, with application replication factor of 3	Application replicas provide availability; vSAN keeps each replica durable.
Pure caches with no durability requirement	None	storageType: Ephemeral	No persistent volume is created; data is rebuilt from the source of truth.

Networking Design

Pod networking inside the VKS cluster is provided by Antrea, a VKS core package. Each database is exposed inside the cluster by KubeDB-managed Services: a primary Service for read/write traffic and, for replicated engines, a standby Service for read-only traffic. Applications in the same VKS cluster connect to these ClusterIP Services by DNS name, which keeps database traffic off external networks.

When clients outside the cluster need access, set the Service type to **LoadBalancer** through the KubeDB `serviceTemplates` field. The request is fulfilled by the Supervisor load balancer, which is NSX with VPC networking in VCF 9 or the VMware Avi Load Balancer, and the address is allocated from the namespace or VPC IP pool. With NSX VPCs, each vSphere Namespace receives its own isolated network, and the distributed firewall can restrict which networks may reach the database load balancer addresses. In the validation environment, the KubeDB Platform gateway received its LoadBalancer address from the NSX VPC IP pool of the vSphere Namespace.

Inside the cluster, enable KubeDB network policies (`global.networkPolicy.enabled=true` at install time) or apply Kubernetes NetworkPolicies so that only the intended application namespaces can reach each database. Antrea enforces these policies natively.

High Availability and Placement

Availability is designed in layers so that no single failure takes a database offline:

- **Database layer**: KubeDB runs replicated topologies (for example a PostgreSQL primary with two hot standbys, or a three-member MongoDB replica set) and performs automatic leader election and failover through a coordinator sidecar. When a cluster has an even number of members, KubeDB adds a small arbiter pod that runs only the coordinator, so leader election always has a quorum. The primary Service always follows the current primary, so applications reconnect without configuration changes.
- **Kubernetes layer**: KubeDB applies preferred pod anti-affinity so that members of one database land on different worker nodes when enough nodes are available. Add `topologySpreadConstraints` on `topology.kubernetes.io/zone` when the cluster spans vSphere Zones, and use PodDisruptionBudgets, which KubeDB creates, to keep quorum during node drains and VKS rolling upgrades.
- **VKS layer**: three control plane nodes protect the Kubernetes API, and Cluster API replaces failed worker nodes automatically. Multi-zone Supervisors place each node pool in a different vSphere Zone for rack or site-level isolation.
- **vSphere layer**: vSphere HA restarts VKS node VMs after a host failure, DRS balances load, and VM anti-affinity keeps nodes of the same pool on separate hosts.

Security and Multi-Tenancy

- **Isolation boundaries**: a vSphere Namespace per environment or business unit sets resource quotas, VM classes, and storage policies. Inside the VKS cluster, each team receives a Kubernetes namespace, and KubeDB databases are scoped to that namespace.
- **Identity and access**: users authenticate to VKS with vCenter Single Sign-On through Pinniped. Kubernetes RBAC grants developers rights on KubeDB resources such as `postgreses.kubedb.com` and `postgresopsrequests.ops.kubedb.com` only in their own namespaces, while platform administrators own the operator namespaces and cluster-scoped catalog objects.
- **Encryption in transit**: cert-manager, installed by the AppsCode installer or available as a VKS add-on, issues certificates for every database through a namespace Issuer or ClusterIssuer referenced in `spec.tls.issuerRef`. KubeDB rotates certificates through ReconfigureTLS OpsRequests.
- **Encryption at rest**: enable vSAN data-at-rest encryption on the cluster, and use the encryption secret of each KubeStash repository so that backup data is encrypted before it leaves the cluster.

- **Credentials**: KubeDB generates database credentials as Kubernetes Secrets, supports user-supplied secrets, and rotates credentials through RotateAuth OpsRequests. KubeVault, also from AppsCode, can issue short-lived, dynamic database credentials from HashiCorp Vault for teams that need them.
- **Supply chain**: mirror KubeDB images to Harbor or another trusted registry for disconnected or regulated sites, and point the Helm chart at that registry.
- **Pod Security**: the VKS cluster manifest in this paper enforces the Pod Security Standard baseline (podSecurityStandard.enforce: baseline). KubeDB, KubeStash, KubeDB Platform, and the database pods run under baseline without changes. The AppsCode installer labels only the monitoring and cert-manager namespaces privileged, because node-exporter and the cert-manager CSI driver need host access.

Data Protection and Disaster Recovery

Data protection combines database-aware backups with platform-level backups. KubeStash, from AppsCode, understands each database engine and takes consistent logical or physical backups on a schedule, applies retention policies, and encrypts and deduplicates the data before writing it to S3-compatible object storage. For engines such as PostgreSQL, MySQL, MariaDB, and MongoDB, the KubeDB archiver adds continuous log archiving and point-in-time recovery. KubeStash also protects everything around the databases: its kubedump add-on backs up the Kubernetes manifests of selected namespaces or the whole cluster, including KubeDB custom resources, Secrets, and ConfigMaps, so a single tool covers both database data and cluster state.

Table 5: Data Protection Layers

Layer	Tool	What it protects	Typical use
Database-aware backup	KubeStash with KubeDB add-ons	Database contents, consistent per engine	Scheduled backups, restores into new or existing databases, point-in-time recovery
Volume snapshots	vSphere CSI VolumeSnapshot	Crash-consistent copy of a persistent volume	Fast local rollback before planned changes
Cluster objects	KubeStash kubedump add-on (manifest-backup)	Kubernetes resources including KubeDB custom resources and Secrets	Cluster rebuild, namespace migration between VKS clusters
Platform	VCF and vSAN tools	vCenter, Supervisor, and infrastructure configuration	Recovery of the underlying private cloud

For disaster recovery, write KubeStash repositories to an object store in a second site. A standby VKS cluster in that site can restore any database from the latest backup or from a chosen point in time, and a KubeStash manifest restore recreates the surrounding namespace objects. Where a lower recovery point is required, engines such as PostgreSQL and MySQL can also run remote replicas in the second site.

Monitoring and Day-2 Operations

Every KubeDB database can expose metrics through a built-in exporter sidecar. With spec.monitor.agent: prometheus.io/operator, KubeDB creates a ServiceMonitor that the Prometheus Operator discovers automatically when it carries the label the Prometheus instance selects on (release: kube-prometheus-stack in this architecture), and AppsCode publishes Grafana dashboards for each engine. VKS also offers Prometheus, Grafana, and Fluent Bit as standard packages for cluster-level telemetry.

Day-2 changes are performed with OpsRequests rather than direct edits. Each OpsRequest is validated by the KubeDB webhook, executed by the ops-manager step by step (for example, updating standbys before the primary and switching

over at the end), and recorded with its status and conditions, which gives operations teams an audit trail for every change. Because VKS upgrades nodes with rolling replacements, databases remain available during Kubernetes upgrades as long as PodDisruptionBudgets and anti-affinity are in place.

Onboarding Existing Databases

VMware Cloud Foundation runs databases on virtual machines and on VKS side by side, on the same vSphere, vSAN, and NSX foundation. KubeDB complements this by giving Kubernetes-native applications a self-service DBaaS on VKS. When a team wants an existing database to be managed by KubeDB, Courier, a KubeDB capability, connects to the existing source database, replicates its data into a KubeDB-managed instance on VKS, keeps the two in sync, and cuts over with minimal downtime and no custom scripts. For MySQL, MariaDB, and MongoDB, the migration runs through schema, snapshot, and streaming (change data capture or oplog) phases. Once migrated, the database is governed by the same policies, backups, and monitoring as every other KubeDB database on the platform.

AppsCode recommends a focused proof-of-value approach for adoption: assess the current cost and workflow, select one contained candidate workload, run it on KubeDB on VKS, validate backup and disaster recovery, and then build the production adoption roadmap. Measuring provisioning time, restore time and success rate, and operational effort on that first workload gives a clear baseline before the same operating model is scaled across the estate.

Kubernetes Packages

VKS clusters (via the VKr release) include a default set of core packages, such as Antrea for networking and Pinniped for authentication.

Table 6: Kubernetes Cluster Packages

Package	Version	Notes
Antrea	2.6.2	VKS core package
Gateway API	1.5.1	VKS core package; CRDs also used by the KubeDB Platform gateway
Guest-cluster auth-service	1.4.8	VKS core package
Metrics Server	0.8.1	VKS core package
Pinniped	0.46.0	VKS core package
Secretgen Controller	0.21.1	VKS core package
kapp-controller	0.60.4	VKS core package
vSphere CPI	1.36.0	VKS core package
vSphere CSI	3.8.0	VKS core package
Cert-Manager	v1.19.3	Installed by the AppsCode installer, or the VKS cert-manager add-on
KubeDB	v2026.10.12	AppsCode DBaaS operators: provisioner, ops-manager, autoscaler, webhook server
KubeStash	v2026.10.12	Database backup, restore and PITR
kube-prometheus-stack	69.2.2	Prometheus Operator v0.80.0, Prometheus v3.1.0, and Alertmanager v0.28.0
KubeDB Platform (ACE)	v2026.10.12	Web console, Hub/Spoke management, and Envoy Gateway (option A only)
Envoy Gateway	v1.8.2	Gateway for the KubeDB Platform console and database access (option A only)
Flux helm-controller	1.5.5	VKS add-on (Flux helm-controller and source-controller); delivers the AppsCode HelmReleases (option A only)
Open Cluster Management	v1.2.0	Cluster manager hub for multi-cluster management (option A only)

Solution Validation

This section walks through installing KubeDB on the VKS cluster described above, deploying a highly available PostgreSQL database, and validating availability, day-2 operations, and backup and restore. The same flow applies to the other KubeDB engines. Command output is taken from the validation run and will differ in names, ages, and identifiers.

Install KubeDB on VKS

Step 1: Connect to the VKS cluster. Log in to the Supervisor with the VCF CLI and vCenter Single Sign-On, and switch to the VKS cluster context. Use kubectl v1.35 or later, within the supported version skew of the v1.36 cluster.

```
vcf context create kubedb-vks-01 \
  --endpoint <SUPERVISOR-IP> \
  --username <SSO-USER> \
  --workload-cluster-namespace kubedb-prod \
  --workload-cluster-name kubedb-vks-01
vcf context use kubedb-vks-01:kubedb-vks-01
kubectl get nodes
```

Step 2: Confirm the vSAN StorageClass and volume expansion.

```
kubectl get storageclass

# Example output
NAME                                  PROVISIONER              VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION
vsan-esa-default-policy-raid5 (default)   csi.vsphere.vmware.com   Immediate              true
vsan-esa-default-policy-raid5-latebinding   csi.vsphere.vmware.com   WaitForFirstConsumer   true
```

Step 3: Obtain a license. KubeDB and KubeStash licenses are issued for a specific cluster, identified by the UID of the `kube-system` namespace. Request a license from the AppsCode License Server (https://appscode.com/issue-license?p=kubedb) or from your AppsCode account team. For option B, save it as `license.txt`.

```
kubectl get ns kube-system -o jsonpath='{.metadata.uid}'
```

Step 4, option A: Install the KubeDB Platform with the AppsCode installer. This is the recommended path and installs the full stack in Table 6. Generate an installer for the cluster from your AppsCode account (https://appscode.com), download the generated archive, and run the deployment steps in its README. On VKS 3.7, enable the VKS helm-controller add-on first and generate the installer with the flux2 feature enabled but not managed. The installer then uses the Flux helm-controller and source-controller provided by VKS instead of installing its own, and installs KubeDB, KubeStash, cert-manager, kube-prometheus-stack, Open Cluster Management, and the KubeDB Platform console as Flux HelmReleases. If the VKS cert-manager add-on is already installed, disable the cert-manager feature in the installer. VKS 3.7 already provides the Gateway API CRDs (v1.5.1) that the KubeDB Platform gateway requires.

```
kubectl get helmreleases -A

# Every HelmRelease (34 in the validation run), including kubedb, kubestash, cert-manager
# and kube-prometheus-stack, reports READY True
```

The installer enables the Elasticsearch (including OpenSearch), Kafka, MariaDB, MongoDB, MySQL, PostgreSQL, Pgpool, and Redis (including Valkey) engines by default. Other engines are enabled by setting `global.featureGates.<Engine>=true` in the kubedb chart values. The validation in this paper covers PostgreSQL.

Step 4, option B: Install KubeDB and KubeStash only, from the AppsCode OCI Helm charts. Use this path when the KubeDB Platform console and multi-cluster management are not needed. It relies on the VKS cert-manager add-on and installs kube-prometheus-stack for the monitoring tests in Table 7. The validation run in this paper used option A; option B installs the same KubeDB and KubeStash charts at the same version.

```
helm install kubedb oci://ghcr.io/appscode-charts/kubedb \
  --version v2026.10.12 \
  --namespace kubedb --create-namespace \
  --set-file global.license=./license.txt \
  --set global.networkPolicy.enabled=true \
  --wait --burst-limit=10000 --debug

helm install kubestash oci://ghcr.io/appscode-charts/kubestash \
  --version v2026.10.12 \
  --namespace stash --create-namespace \
  --set-file global.license=./license.txt \
  --wait --burst-limit=10000 --debug

kubectl create namespace monitoring
kubectl label namespace monitoring pod-security.kubernetes.io/enforce=privileged
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install kube-prometheus-stack prometheus-community/kube-prometheus-stack \
  --version 69.2.2 \
  --namespace monitoring \
  --wait
```

The kubedb chart enables Elasticsearch, Kafka, MariaDB, MongoDB, MySQL, PostgreSQL, and Redis by default. Add `--set global.featureGates.<Engine>=true` for each additional engine, for example `global.featureGates.MSSQLServer=true`. The monitoring namespace is labeled privileged because node-exporter needs host access, which the baseline Pod Security Standard does not allow. Option B provides the Prometheus part of the monitoring test; the KubeDB Grafana dashboards are installed by option A, or can be imported as described in the KubeDB monitoring documentation.

For disconnected sites, mirror the AppsCode images to Harbor and add the registry overrides documented in the KubeDB air-gapped installation guide.

Step 5: Verify the operators.

```
kubectl get pods -n kubedb
kubectl get pods -n stash
kubectl get postgresversions

# All kubedb-* pods (provisioner, ops-manager, autoscaler, webhook-server, petset)
# and the kubestash operator pods report Running
```

Deploy a Highly Available PostgreSQL Database

Create a namespace and a CA Issuer for database TLS. In production, reference an enterprise ClusterIssuer instead of a self-signed CA.

```
kubectl create namespace demo
openssl req -x509 -nodes -newkey rsa:2048 -days 365 \
  -keyout ca.key -out ca.crt -subj "/CN=postgres/O=kubedb"
kubectl create secret tls pg-ca --cert=ca.crt --key=ca.key -n demo
```

```
cat <<EOF | kubectl apply -f -
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: pg-ca-issuer
  namespace: demo
spec:
  ca:
    secretName: pg-ca
EOF
```

Declare a three-member PostgreSQL cluster on the vSAN StorageClass, with TLS and Prometheus monitoring enabled.

```
# pg-ha.yaml
apiVersion: kubedb.com/v1
kind: Postgres
metadata:
  name: pg-ha
  namespace: demo
spec:
  version: "18.3"
  replicas: 3
  standbyMode: Hot
  storageType: Durable
  storage:
    storageClassName: vsan-esa-default-policy-raid5
    accessModes: ["ReadWriteOnce"]
    resources:
      requests:
        storage: 50Gi
  sslMode: require
  tls:
    issuerRef:
      apiGroup: cert-manager.io
      kind: Issuer
      name: pg-ca-issuer
  podTemplate:
    spec:
      containers:
        - name: postgres
          resources:
            requests: { cpu: "2", memory: 8Gi }
            limits: { memory: 8Gi }
  monitor:
    agent: prometheus.io/operator
    prometheus:
      serviceMonitor:
        labels:
          release: kube-prometheus-stack
        interval: 30s
  deletionPolicy: Delete
```

```
kubectl apply -f pg-ha.yaml
kubectl get postgres -n demo -w
```

Validate KubeDB on VKS

Database status and roles. The database reaches the Ready state, and each pod carries a kubedb.com/role label that identifies the primary and the standbys. Each pod runs three containers: postgres, the pg-coordinator sidecar that handles leader election, and the Prometheus exporter.

```
kubectl get postgres -n demo
NAME    VERSION   STATUS   AGE
pg-ha   18.3      Ready    2m

kubectl get pods -n demo -l app.kubernetes.io/instance=pg-ha  -L kubedb.com/role
NAME      READY   STATUS    RESTARTS   AGE   ROLE
pg-ha-0   3/3     Running   0          63s   primary
pg-ha-1   3/3     Running   0          47s   standby
pg-ha-2   3/3     Running   0          37s   standby
```

Persistent volumes on vSAN. Each member has its own claim bound to the vSAN StorageClass. The same volumes can also be viewed in vCenter under Cloud Native Storage > Container Volumes, together with their storage policy compliance.

```
kubectl get pvc -n demo
NAME          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
data-pg-ha-0  Bound    pvc-975af0b3-633a-4dd2-ae73-9b4bdd23c621   50Gi       RWO            vsan-esa-default-policy-raid5
data-pg-ha-1  Bound    pvc-ce4c444d-5ed7-4740-9ee4-529d8167ca47   50Gi       RWO            vsan-esa-default-policy-raid5
data-pg-ha-2  Bound    pvc-688e11a0-e965-49a7-9bfb-0ee8279e5ee8   50Gi       RWO            vsan-esa-default-policy-raid5
```

Connectivity and data. Read the generated credentials, write test data through the primary Service (pg-ha), and read it back through the standby Service (pg-ha-standby). A member pod serves only as the psql client.

```
PGPASSWORD=$(kubectl  get secret -n demo  pg-ha-auth  -o jsonpath='{.data.password}'  | base64  -d)
kubectl exec -n demo  pg-ha-0 -c postgres -- env PGPASSWORD="$PGPASSWORD"  \
  psql "host=pg-ha  user=postgres  sslmode=require"  \
  -c "CREATE  TABLE  ra(id int); INSERT  INTO ra VALUES  (1),(2),(3);"    # primary  Service
kubectl exec -n demo  pg-ha-0 -c postgres -- env PGPASSWORD="$PGPASSWORD"  \
  psql "host=pg-ha-standby  user=postgres sslmode=require"  \
  -c "SELECT  count(*) FROM ra;"    # standby  Service
```

Automatic failover. Delete the primary pod to simulate a failure. A standby is promoted, the primary Service follows it, and the old primary rejoins as a standby once Kubernetes recreates it. Writes through the primary Service succeed without any client change.

```
kubectl delete  pod -n demo  pg-ha-0
kubectl get pods  -n demo -l app.kubernetes.io/instance=pg-ha  -L kubedb.com/role  -w
NAME      READY   STATUS    RESTARTS   AGE    ROLE
pg-ha-0   3/3     Running   0          16s    standby
pg-ha-1   3/3     Running   0          119s   primary
pg-ha-2   3/3     Running   0          109s   standby

kubectl exec -n demo  pg-ha-2 -c postgres -- env PGPASSWORD="$PGPASSWORD"  \
  psql "host=pg-ha  user=postgres  sslmode=require"  \
  -c "INSERT  INTO ra VALUES  (4);"    # pg-ha Service now routes to pg-ha-1
```

Online volume expansion. Grow every member volume from 50Gi to 100Gi while the database stays online. vSphere CSI expands the CNS volumes and the file systems in place.

```
apiVersion: ops.kubedb.com/v1alpha1
kind: PostgresOpsRequest
metadata:
  name: pg-ha-expand
  namespace: demo
spec:
  type: VolumeExpansion
  databaseRef:
    name: pg-ha
  volumeExpansion:
    mode: Online
    postgres: 100Gi
---
kubectl get postgresopsrequest -n demo
NAME           TYPE              STATUS       AGE
pg-ha-expand   VolumeExpansion   Successful   3m
```

Scaling and TLS rotation. Horizontal scaling, vertical scaling, and certificate rotation use the same pattern; apply each OpsRequest after the previous one reports Successful. Scaling from three to four members also adds the arbiter pod described in High Availability and Placement, and scaling back to three removes it. Vertical scaling and TLS rotation restart the members one at a time, standbys first, and finish with a switchover. Version updates use type UpdateVersion.

```
apiVersion: ops.kubedb.com/v1alpha1
kind: PostgresOpsRequest
metadata: { name: pg-ha-scale-4, namespace: demo }
spec:
  type: HorizontalScaling
  databaseRef: { name: pg-ha }
  horizontalScaling: { replicas: 4 }    # pg-ha-scale-3 sets replicas: 3
---
apiVersion: ops.kubedb.com/v1alpha1
kind: PostgresOpsRequest
metadata: { name: pg-ha-vscale, namespace: demo }
spec:
  type: VerticalScaling
  databaseRef: { name: pg-ha }
  verticalScaling:
    postgres:
      resources:
        requests: { cpu: "3", memory: 10Gi }
        limits: { memory: 10Gi }
---
apiVersion: ops.kubedb.com/v1alpha1
kind: PostgresOpsRequest
metadata: { name: pg-ha-tls-rotate, namespace: demo }
spec:
  type: ReconfigureTLS
  databaseRef: { name: pg-ha }
  tls: { rotateCertificates: true }
---
kubectl get postgresopsrequest -n demo
NAME              TYPE              STATUS       AGE
pg-ha-expand      VolumeExpansion   Successful   16m
pg-ha-scale-4     HorizontalScaling Successful   14m
pg-ha-scale-3     HorizontalScaling Successful   13m
pg-ha-vscale      VerticalScaling   Successful   10m
```

```
pg-ha-tls-rotate   ReconfigureTLS      Successful   7m
```

Backup and restore with KubeStash. Define an S3-compatible backup target, a retention policy, and a backup schedule for the database, then restore the latest snapshot into a new database.

```
kubectl create secret generic -n demo s3-secret \
  --from-literal=AWS_ACCESS_KEY_ID=<KEY>  --from-literal=AWS_SECRET_ACCESS_KEY=<SECRET>
kubectl create secret generic -n demo encrypt-secret --from-literal=RESTIC_PASSWORD=<PASSWORD>

apiVersion: storage.kubestash.com/v1alpha1
kind: BackupStorage
metadata: { name: s3-storage, namespace: demo }
spec:
  storage:
    provider: s3
    s3:
      endpoint: https://s3.example.internal
      bucket: kubedb-backups
      region: us-east-1
      prefix: /vks/kubedb-vks-01
      secretName: s3-secret
  usagePolicy: { allowedNamespaces: { from: Same } }
  default: false
  deletionPolicy: Delete
---
apiVersion: storage.kubestash.com/v1alpha1
kind: RetentionPolicy
metadata: { name: keep-14d, namespace: demo }
spec:
  maxRetentionPeriod: 14d
  successfulSnapshots: { last: 14 }
  failedSnapshots: { last: 2 }
  usagePolicy: { allowedNamespaces: { from: Same } }
---
apiVersion: core.kubestash.com/v1alpha1
kind: BackupConfiguration
metadata: { name: pg-ha-backup, namespace: demo }
spec:
  target: { apiGroup: kubedb.com, kind: Postgres, namespace: demo, name: pg-ha }
  backends:
    - name: s3-backend
      storageRef: { namespace: demo, name: s3-storage }
      retentionPolicy: { namespace: demo, name: keep-14d }
  sessions:
    - name: nightly
      scheduler: { schedule: "0 1 * * *", jobTemplate: { backoffLimit: 1 } }
      repositories:
        - name: pg-ha-repo
          backend: s3-backend
          directory: /pg-ha
          encryptionSecret: { namespace: demo, name: encrypt-secret }
      addon:
        name: postgres-addon
        tasks: [ { name: logical-backup } ]
```

Set region to the region of the bucket; S3-compatible stores such as MinIO also need a value, for example us-east-1. If the endpoint uses a certificate from a private CA, add the CA certificate to s3-secret as CA_CERT_DATA. The BackupStorage is not marked as the cluster default because the AppsCode installer already creates one. KubeStash takes the first backup as soon as the BackupConfiguration is created and then follows the schedule. To take a backup on demand, use the kubectl-kubestash plugin: kubectl kubestash trigger pg-ha-backup -n demo --sessions nightly.

```
kubectl get backupconfiguration,backupsession  -n demo
kubectl get snapshots -n demo -l kubestash.com/repo-name=pg-ha-repo

# Create the target database. waitForInitialRestore keeps it from
# accepting connections until the RestoreSession completes.
apiVersion: kubedb.com/v1
kind: Postgres
metadata:
  name: pg-restored
  namespace: demo
spec:
  init:
    waitForInitialRestore: true
  version: "18.3"
  replicas: 3
  standbyMode: Hot
  storageType: Durable
  storage:
    storageClassName: vsan-esa-default-policy-raid5
    accessModes: ["ReadWriteOnce"]
    resources:
      requests:
        storage: 100Gi
  deletionPolicy: Delete
---
# Restore the latest snapshot of pg-ha into pg-restored
apiVersion: core.kubestash.com/v1alpha1
kind: RestoreSession
metadata: { name: pg-ha-restore, namespace: demo }
spec:
  target: { apiGroup: kubedb.com, kind: Postgres, namespace: demo, name: pg-restored }
  dataSource:
    repository: pg-ha-repo
    snapshot: latest
    encryptionSecret: { namespace: demo, name: encrypt-secret }
  addon:
    name: postgres-addon
    tasks: [ { name: logical-backup-restore } ]
```

Verify the restore. The restored database contains every row written to pg-ha, including rows written after the failover. Any member of pg-restored can become its primary, so do not assume pod 0 is the primary; reads work on every member.

```
kubectl get restoresession -n demo
kubectl exec -n demo pg-restored-0 -c postgres -- psql -U postgres -c "SELECT count(*) FROM ra;"
 count
-------
     4
(1 row)
```

Monitoring. KubeDB creates the ServiceMonitor pg-ha-stats with the label release: kube-prometheus-stack, and Prometheus scrapes two endpoints on every member: the PostgreSQL exporter (metrics) and the pg-coordinator (raft-metrics). In Grafana, the KubeDB / Postgres / Summary dashboard shows the database.

```
kubectl get servicemonitor -n demo pg-ha-stats

# Prometheus query: up{namespace="demo", job="pg-ha-stats"}
# Returns 1 for all six targets (three members x two endpoints)
```

Table 7 summarizes the tests and the results measured on the validation cluster described in VKS Cluster Configuration. All tests passed.

Table 7: Validation Test Matrix

Test	Method	Expected result	Result
Provisioning	Apply a Postgres resource with 3 replicas on the vSAN StorageClass	Status Ready; one primary and two standbys; three bound CNS volumes	Pass. Ready in 71 s
Failover	Delete the primary pod	A standby is promoted automatically; the primary Service follows it; the old primary rejoins as a standby	Pass. Standby promoted in 31 s; old primary rejoined in 47 s
Volume expansion	PostgresOpsRequest of type VolumeExpansion, mode Online, 50Gi to 100Gi	All PVCs grow to the new size without restart	Pass. 2 min 28 s; no pod restarts
Horizontal scaling	PostgresOpsRequest of type HorizontalScaling, 3 to 4 and back to 3 members	Replica count changes with the database available	Pass. 68 s up; 1 min 51 s down
Vertical scaling	PostgresOpsRequest of type VerticalScaling	Every member runs with the new CPU and memory after a rolling restart	Pass. 3 min 8 s
TLS rotation	PostgresOpsRequest of type ReconfigureTLS with rotateCertificates	New server certificates are issued; data is intact	Pass. 5 min 3 s
Backup and restore	KubeStash BackupConfiguration and RestoreSession to S3	Snapshot succeeds; restored database contains the test data	Pass. Backup in 9 s; restore in 37 s; all rows restored
Monitoring	Prometheus targets and KubeDB Grafana dashboard	Database metrics are scraped and displayed	Pass. All pg-ha targets up; dashboard populated

Conclusion

KubeDB on vSphere Kubernetes Service gives VMware Cloud Foundation customers a private-cloud Database-as-a-Service that matches the self-service experience of a public cloud while keeping data, cost, and governance under their own control. VKS supplies conformant Kubernetes clusters that are provisioned and upgraded with familiar vSphere tools, vSAN supplies policy-driven persistent storage through vSphere CSI, and NSX supplies isolated, load-balanced networking. KubeDB turns that foundation into a catalog of production-ready databases with automated provisioning, high availability, day-2 operations, monitoring, and backup and recovery.

The architecture in this paper, a Supervisor that can span multiple vSphere Zones, a dedicated VKS cluster for data services, vSAN storage policies matched to each database tier, and layered data protection with KubeStash and vSphere CSI snapshots, provides a repeatable starting point. Platform teams can begin with a single engine such as PostgreSQL or one migrated workload, validate it with the tests in Table 7, and then extend the same patterns to MySQL, MongoDB, Redis, Kafka, and the other engines KubeDB supports.

For more information, see the KubeDB documentation at https://kubedb.com/docs, the KubeStash documentation at https://kubestash.com/docs, and the vSphere Kubernetes Service documentation at https://techdocs.broadcom.com.

vmware by Broadcom

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