VOLT ACTIVE DATA

Volt on Kubernetes

A stateful, strongly consistent data platform that behaves like a Kubernetes-native workload

Summary

Kubernetes was built for stateless services, and most stateful systems have never fully made the leap. They either stay outside the cluster on dedicated infrastructure, or they run inside it as a fixed-size monolith that Kubernetes schedules but cannot really manage. Volt takes a different route. The Volt Operator makes a strongly consistent, in-memory data platform a first-class Kubernetes workload: deployed with Helm, described as a custom resource, scaled up and down while it serves traffic, autoscaled on the metrics that matter to a transactional system, upgraded in service, and replicated between clusters, regions and clouds. Tier-1 communications service providers run real-time charging, policy and mediation on it today, on Red Hat OpenShift, Amazon EKS, Azure AKS and vendor-supplied Kubernetes distributions.

Stateful systems have a Kubernetes problem

Platform teams move to Kubernetes to get one operating model: one way to deploy, one way to scale, one way to patch, one set of observability tools and one place to look when something goes wrong. Stateless services deliver that almost for free. The systems that hold state, and especially the systems of record behind real-time charging, policy control, fraud detection and mediation, usually end up in one of two compromises.

The data platform stays outside. It runs on virtual machines or bare metal, managed by database administrators with their own tooling, and the containerized application tier reaches it over the network. The organization now runs two platforms, two upgrade cadences, two on-call rotations and two sets of automation. Every capacity change or operating-system patch on the data tier is a project with its own change window.

The data platform runs as a monolith inside. It is wrapped in a container image and deployed as a plain StatefulSet. It is “on Kubernetes” in the sense that the scheduler placed its pods, but little else has changed. The replica count is fixed and edited by hand. Adding capacity means a maintenance window and a runbook. Kubernetes has no idea that the pods form a coordinated cluster, so a routine node drain or a rolling upgrade can restart the wrong pods in the wrong order. The result is a monolith that happens to live in a container.

Both compromises exist for the same reason: the primitives built into Kubernetes do not understand what a strongly consistent, partitioned cluster needs.

Why the built-in tools can't fix it

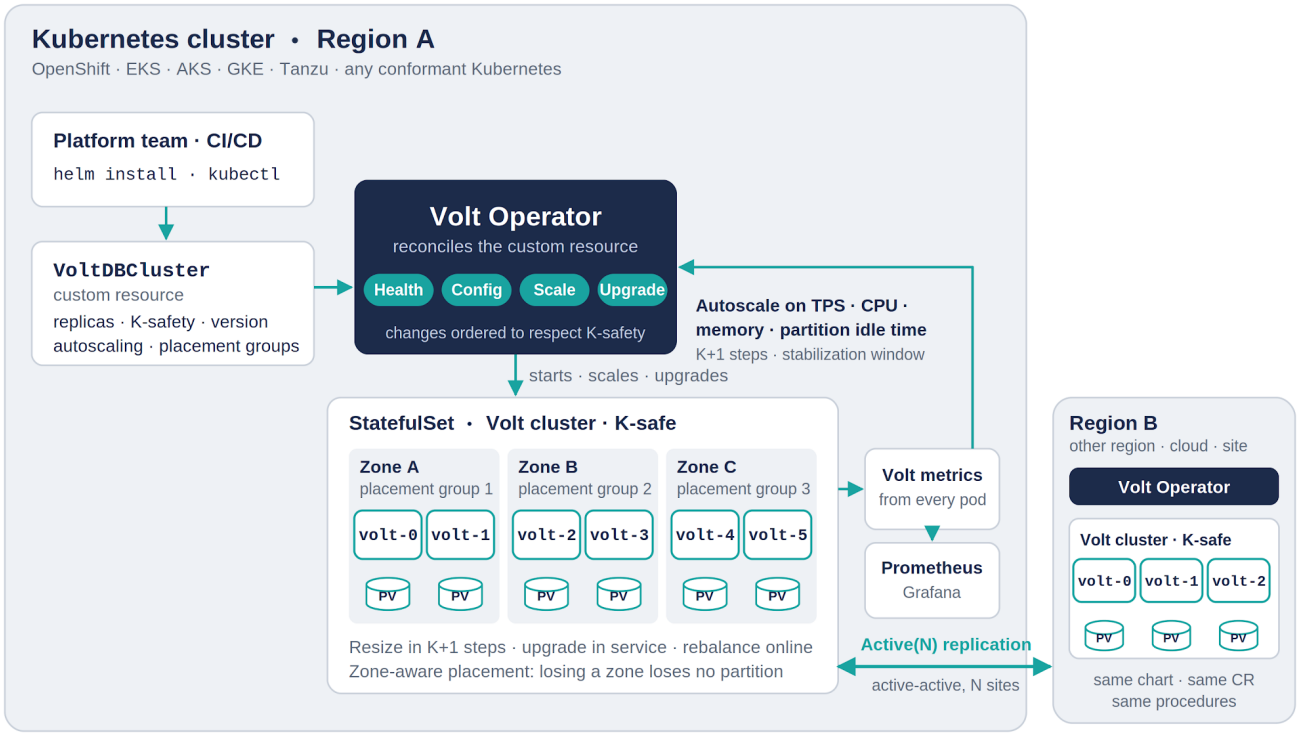
The Horizontal Pod Autoscaler watches pod CPU and memory, or a custom metric, and changes a replica count. For a web tier, that is exactly right. For a partitioned, replicated data platform, it is wrong in four ways.

- **Granularity.** A K-safe cluster adds or removes K+1 nodes at a time so that every partition keeps K replicas. The HPA adds one pod.
- **Data placement.** When the HPA scales a StatefulSet down, it deletes the highest-numbered pod, whatever data that pod holds. A stateful system has to move partitions off a node before the node leaves.
- **Triggers.** Pod CPU says little about a transactional system's headroom. The signals that matter are throughput, partition idle time and memory footprint, and those live inside the cluster, not in the kubelet.
- **Rebalancing.** Resizing a stateful cluster is a multi-minute operation that redistributes data. The HPA's model of "add a pod and the service mesh does the rest" does not apply, and without a stabilization window it will flap between scaling up and scaling down.

Rolling restarts, node drains and upgrades have the same gap: Kubernetes performs them in pod order, not in an order that keeps every partition available. The Vertical Pod Autoscaler restarts pods to resize them, which is the opposite of what a system of record wants.

The Kubernetes answer to this class of problem is an Operator: a controller that encodes the domain knowledge Kubernetes itself lacks. Volt ships one, and it is the heart of Volt on Kubernetes.

How Volt does it



One control plane

A Volt cluster on Kubernetes is described by a single custom resource (`VoltDBCluster`) and installed with one Helm chart. The Volt Operator reconciles that resource: it initializes and starts the Volt nodes as a StatefulSet with persistent volumes for command logs and snapshots, monitors their health, applies configuration changes, and coordinates every structural change in an order that respects K-safety. One Operator and one chart manage every supported Volt version; choosing a release is a single property, with no need to match Operator, chart and image versions by hand.

Elastic scaling, online

Change the replica count in the Helm release and the Operator expands or shrinks the cluster while it continues to serve traffic, in steps of K+1 nodes, redistributing partitions as it goes. Shrinking is as supported as growing: the Operator moves data off the departing nodes before they leave, so the cluster never loses a replica it did not intend to.

Autoscaling driven by the data platform, not the pod

With autoscaling enabled, the Operator watches four metrics from inside Volt against scale-up and scale-down thresholds you set: transactions per second, CPU, memory footprint (as resident set size or percentage) and partition idle time. When a threshold is crossed and stays crossed for a stabilization window (ten minutes by default), the Operator starts an elastic resize of K+1 nodes, within the minimum and maximum cluster size you allow. Progress is reported in the cluster status and as Kubernetes events, so `kubectl describe` shows exactly which metric triggered the change and how far it has progressed. This is autoscaling designed for a stateful system: triggered by the right signals, sized in the right increments, and damped against flapping.

Zero-downtime change

Volt software upgrades run as in-service software upgrades (ISSU), driven by the Operator one node at a time while the cluster keeps processing transactions. Many database properties change on the running cluster; those that need a restart get a coordinated rolling restart rather than a stop-start. Schema changes are made online with the standard Volt tools. Kubernetes maintenance becomes routine rather than exceptional: K-safety keeps every partition available while a node's pods are rescheduled, and command logs and snapshots on persistent volumes survive pod restarts intact.

Failure domains you control

Placement groups map Volt's replica placement onto Kubernetes topology (availability zones, racks or node pools), so the loss of an entire zone still leaves every partition with a live copy. Combined with standard Kubernetes affinity rules, placement groups let the Operator keep replicas apart where it matters and together where it is efficient.

Multi-cluster, multi-region, multi-cloud

Active(N) cross-data-center replication runs between Volt clusters in separate Kubernetes clusters, with a networking option for every topology: clusters in the same Kubernetes cluster, per-pod load balancers, a single load balancer with network peering, node ports, or an external network service. Each site is active, so traffic can be served from any region, and rejoining a site after an outage is an Operator-managed procedure. This is how Tier-1 operators run active-active across availability zones and regions today, including three-site active-active-active deployments in which every site serves live traffic.

Security and observability the Kubernetes way

Users and roles are defined in Kubernetes Secrets. TLS material comes from Secrets or cert-manager, with certificate revocation lists and rotation, and the Operator and the Volt Management Center are secured the same way. Every pod exposes Volt metrics to Prometheus for your existing Grafana dashboards and alerting, and the Operator publishes cluster state as Kubernetes events. A diagnostics pod collects logs and snapshots for support without requiring shell access to the data nodes.

Runs where your platform team already is

Volt runs on any recent, conformant Kubernetes distribution and is validated on AWS, Azure, Google Cloud, Red Hat OpenShift and VMware Tanzu. On OpenShift, it installs from OperatorHub through the Operator Lifecycle Manager. Container images are built on Red Hat Universal Base Image, and every release ships multi-architecture images for x86-64 and 64-bit ARM from the same release train. Helm is the only client tooling required.

Three patterns compared

	Outside Kubernetes (VMs or bare metal)	Monolith inside Kubernetes (plain StatefulSet)	Volt with the Operator
Deployment	Separate tooling and runbooks from the application tier	Helm or YAML, but cluster bootstrap scripted by hand	One Helm chart; the Operator bootstraps and reconciles
Scaling up	Provision servers, join nodes manually	Replica count edited by hand; cluster membership managed manually	Change the replica count; the Operator adds K+1 nodes online and rebalances
Scaling down	Rare, manual, risky	StatefulSet removes the last pod regardless of the data it holds	The Operator moves partitions off departing nodes, then removes K+1 nodes online
Autoscaling	None	HPA on pod CPU or memory: wrong increment, wrong triggers	Operator autoscaling on TPS, CPU, memory and partition idle time, with stabilization windows and size bounds
Software upgrade	Maintenance window or bespoke procedure	Rolling restart in pod order; availability not guaranteed	Operator-driven in-service software upgrade, one node at a time, cluster stays in service
Node maintenance	Platform team coordinates with DBAs	A drain can evict several replicas of one partition	K-safety and placement groups keep every partition available as pods move
Placement across zones	Manual host selection	Anti-affinity only	Placement groups map replicas to zones and racks
Multi-region	Replication configured host by host	Do-it-yourself networking between clusters	Active(N) between Kubernetes clusters, with five networking options
Observability	Separate monitoring stack	Pod metrics only	Volt pods autodiscovered and scraped by Prometheus; cluster state as Kubernetes events
Where the operational knowledge lives	DBAs and runbooks	DBAs and runbooks	The Operator

What this means in practice

One operating model for the whole stack. The data tier is deployed, scaled, patched, secured and observed with the same tools and the same pipelines as everything else on the cluster. There is no second platform to own.

Capacity that follows the workload. Rush-hour traffic, campaign spikes and end-of-month cycles are absorbed by the Operator adding nodes when throughput or memory demands it, and releasing them when it does not. Headroom is bought by the hour rather than provisioned for the peak.

Change without ceremony. Version upgrades, configuration changes and node maintenance happen while the cluster processes transactions. The maintenance window stops being the unit of operational planning.

Portability as a decision, not a project. The same chart, the same custom resource and the same procedures apply on OpenShift in a private data center, on EKS or AKS in a public cloud, and on a network vendor's own Kubernetes distribution. Moving a workload between them is a configuration change at deployment time.

Proven in production

The Volt Operator first shipped in 2020, and Tier-1 communications service providers have run production charging, policy and mediation workloads on it for several years, across Europe, North America, the Middle East and Africa, on private cloud and public cloud alike:

- Real-time charging and policy control for Tier-1 CSPs on Red Hat OpenShift on operators' private clouds and dedicated hardware, including multi-site deployments with Active(N) replication between OpenShift clusters
- Red Hat OpenShift on AWS
- Amazon EKS, including a 20-node production cluster, with Active(N) sites in separate availability zones
- Azure AKS for a Tier-1 operator: policy control at K=2 across three active-active-active sites, charging in two-site Active(N) pairs
- VMware Tanzu Kubernetes Grid on an operator's telco cloud platform
- A network equipment vendor's own Kubernetes distribution, inside a Tier-1 European operator's mediation stack

Architectural properties

- Volt Operator, Helm chart and container images ship with every Volt release; one Operator manages all supported Volt versions
- Validated on AWS, Azure, Google Cloud, Red Hat OpenShift and VMware Tanzu; OpenShift installation via OperatorHub and the Operator Lifecycle Manager
- Elastic scale-up and scale-down online in K+1 steps; Operator autoscaling on TPS, CPU, memory and partition idle-time thresholds with stabilization windows and minimum/maximum bounds
- In-service software upgrade (ISSU) on Kubernetes; rolling configuration changes; online schema changes
- K-safety with placement groups for zone- and rack-aware replica placement
- Active(N) cross-data-center replication between Kubernetes clusters: same-cluster, per-pod load balancer, single load balancer with peering, node port, or external network service
- Kubernetes Secrets and cert-manager for credentials and TLS; certificate revocation list support; TLS for the Operator and Volt Management Center
- Prometheus metrics from every pod; cluster status and events via kubectl; diagnostics pod for support data collection
- Red Hat Universal Base Image; multi-architecture images for x86-64 and 64-bit ARM
- Full enterprise feature set on Kubernetes: command logging, snapshots, export and import, Volt Management Center

Key outcomes

- Run the data tier with the same tools, pipelines and people as the rest of the Kubernetes estate
- Scale up and down with the workload, in the right increments, without a maintenance window
- Upgrade, reconfigure and patch while the cluster keeps processing transactions
- Choose cloud, Kubernetes distribution and hardware on price and policy, not on what the data platform supports

AVAILABILITY

Volt on Kubernetes (the Volt Operator, Helm charts and container images) is part of Volt Active Data Enterprise and ships with every current release.

Book a technical call voltactivedata.com/contact

Volt Active Data is the real-time decision layer for mission-critical systems.