AccuKnox
Secure Code to Cognition™

AccuKnox Control Plane Architecture Technical Review

Prepared by
AccuKnox
support@accuknox.com

CERTIFIED & ACCREDITED BY

AS FEATURED IN

AVAILABLE ON

Table of Contents

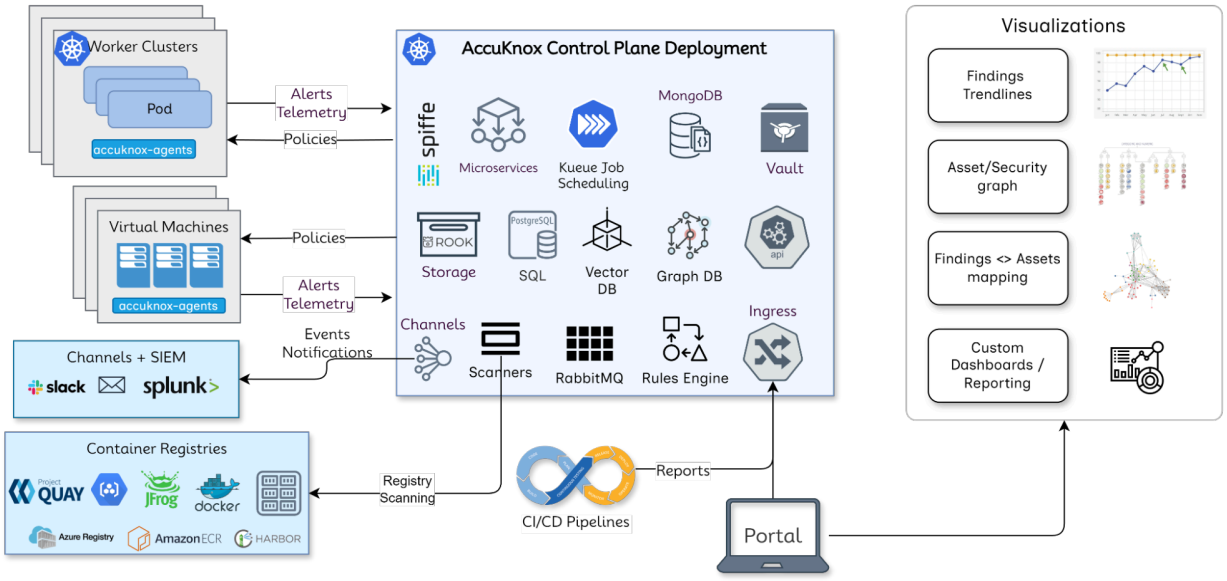
Introduction

This technical review presents the AccuKnox Control Plane Architecture, focusing on its non-functional characteristics, namely, resiliency, stability, and availability, that underpin secure, large-scale enterprise deployments. The control plane is designed as a kubernetes-native, distributed system that remains operational under failure conditions, scales predictably with workload growth, and maintains a consistent security posture enforcement across hybrid, multi-cloud, and edge environments. By leveraging fault-tolerant design principles, stateless services, horizontal scalability, and strong isolation between control and data paths, the AccuKnox control plane ensures high availability and graceful degradation without compromising scalability. This review highlights how these architectural decisions collectively deliver a stable, resilient, and always-on security platform suitable for mission-critical production environments.

High-Level AccuKnox Architecture

AccuKnox Control Plane is fully deployed on k8s. AccuKnox uses a k8s-native model for deployment/development, such as:

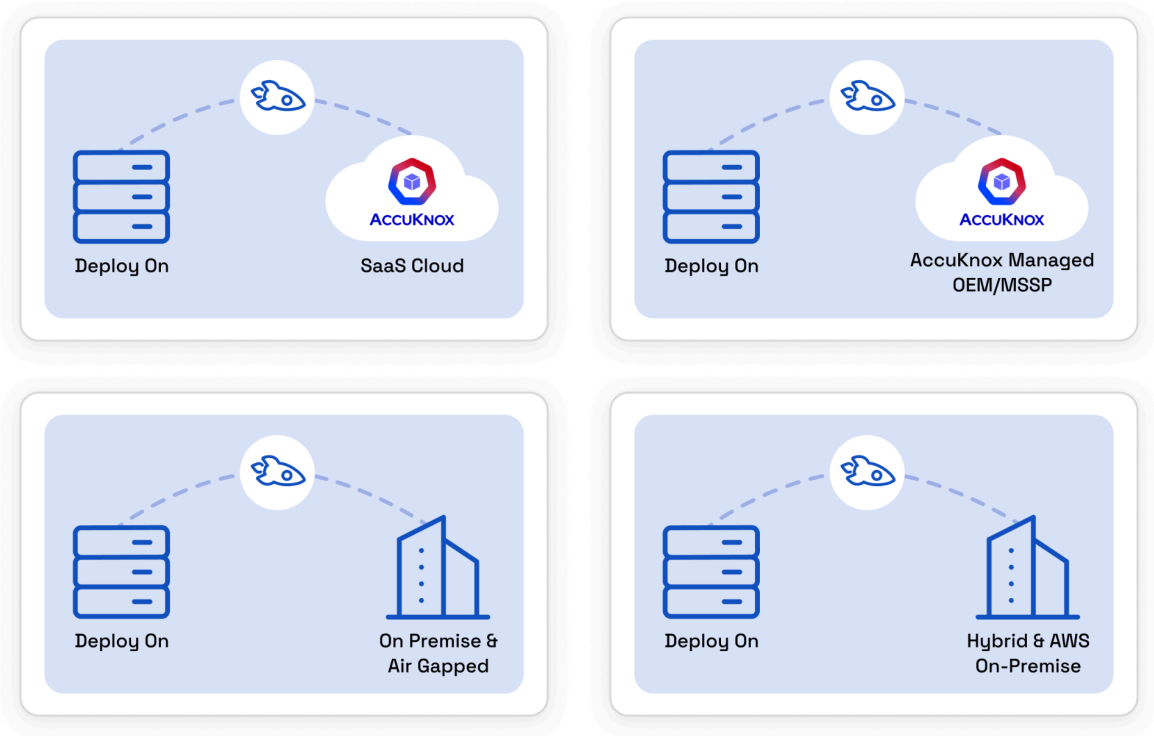
- Use of k8s config maps to keep the configuration
- Use of k8s horizontal scaling for workloads/microservices
- Using Vault and injecting secrets into the manifests dynamically
- As far as possible, AccuKnox uses stateless microservices. If the state has to be maintained, it is maintained in storage/databases that are common across all services.
- AccuKnox isolates tenants' playbooks in the control plane using separate k8s namespaces and applies AccuKnox namespace isolation techniques to prevent lateral movement.
- Use k8s rollout deployments for smooth or canary-based upgrades of the services.
- AccuKnox architecture assumes that a service may be killed at any processing boundary, and that it should be possible to restart gracefully.
- AccuKnox leverages strong onboarding mechanisms such as SPIFFE for onboarding end-user worker clusters (k8s or virtual machines).



Deployment Models

AccuKnox supports multiple deployment models to match operational, regulatory, and infrastructure needs, while *core platform capabilities like scaling, high availability, and self-healing remain consistent across SaaS and on-prem deployments.*

1. **SaaS (Fully Managed by AccuKnox):** Fastest to deploy and update. AccuKnox handles upgrades, maintenance, and data retention. Not air-gapped. Multi-tenant infrastructure. All features supported.
2. **Managed OEM / MSSP:** Designed for partners delivering managed services. AccuKnox operates and maintains the platform while the partner stays informed and customer-facing.
3. **AWS Hybrid (Cloud + On-Prem):** Control plane uses AWS managed services like S3 and RDS while workloads can remain on-prem. Balances scalability with partial infrastructure control.
4. **Full On-Prem / Air-Gapped:** Deployed entirely in the customer environment with no shared infrastructure. Customer manages upgrades and data retention with AccuKnox support. Highest isolation. AI CoPilot not included.



Handling Scalability

How does AccuKnox scale resources when new tenants are onboarded or if existing tenants expand their asset base?

AccuKnox leverages the k8s control plane for scaling the resources horizontally. Most services do not need vertical scaling. New nodes are added dynamically based on the resource usage, and the services scaling happens using k8s native constructs. It's mandatory for all AccuKnox control plane entities to specify resource limits for both CPU and memory. AccuKnox leverages the following metrics for horizontal scaling:

- CPU-based thresholds
- Memory-based thresholds
- Queue-based thresholds (RMQ, Celery)
- Use of Kueue for rationing access to the resources.

How does AccuKnox handle the Noisy-Neighbor Problem?

AccuKnox runs a mix of security workloads that vary widely in CPU/memory/IO intensity and latency tolerance; for example, parsing telemetry, enrichment, policy generation, scanning, correlation, and on-demand "playbooks" (investigation workflows) or batch jobs. In a shared Kubernetes cluster, this mix can lead to the classic noisy neighbor problem: a tenant or heavy workload consumes disproportionate resources, causing tail latency spikes, throttling, or failures for other tenants.
A particular tenant could overwhelm the resource allocation, resulting in other tenants' executions getting starved. The noisy neighbor problem could occur in the following cases:

- Too many playbooks are invoked by a single tenant, causing it to hog all the playbook execution resources
- Too many telemetry events are causing the messaging queues to get overwhelmed while processing single-tenant data

To solve this, AccuKnox adopts a Kubernetes-native architecture for resource governance + scheduling fairness by combining:

- Isolation primitives (Namespaces + per-tenant / per-workload isolation)
- Workload shaping (requests/limits, priority, and topology rules)
- Cluster-wide admission + queuing with Kueue (fair sharing across teams/tenants/workload types)

- Dedicated parser worker pools (isolated, horizontally scalable ReplicaSets/Deployments)
- Policy-based quotas and borrowing (guarantees + controlled burst)

This yields predictable performance, efficient cluster utilization, and enforceable fairness without introducing a custom scheduler.

Why Kueue fits AccuKnox scheduling needs?

AccuKnox has two broad workload classes:

A. Always-on / steady-state services: Examples: API, ingestion gateways, control plane services, “hot path” real-time components. These should be protected first: stable SLOs, predictable resource reservations.
B. Elastic/bursty/batch workloads: Examples: playbooks, analysis jobs, batch correlation, large parsing/enrichment, scanning bursts, retroactive investigations. These are perfect candidates for queue-based admission control:
    - They can wait briefly if the cluster is constrained.
    - They should run with guaranteed fairness across tenants/teams.
    - They should be able to burst when capacity is available.

Kueue is designed for exactly this: it sits in front of batch-like workloads and decides when and where they are admitted, based on cluster capacity and fairness rules.

Handling Reliability

The AccuKnox Control Plane is designed with reliability as a first-class non-functional requirement. Given its role in managing security policies, telemetry ingestion, analytics, and orchestration across distributed cloud-native environments, the control plane must remain highly available, resilient to failures, and operationally predictable even under burst load or partial infrastructure failures.
AccuKnox achieves this by leveraging Kubernetes-native deployment patterns, stateless microservices, and managed cloud infrastructure, combined with robust data durability and recovery mechanisms.

Kubernetes-based Control Plane Deployment

AccuKnox deploys its control plane entirely on Kubernetes, using it as the foundational orchestration and reliability layer. Advantages of using Kubernetes for the control plane

1. Built-in high availability: Control plane microservices are deployed as Deployments / StatefulSets with multiple replicas. Kubernetes automatically reschedules pods on healthy nodes in case of:
    a. Node failures
    b. Kernel panics
    c. Cloud infrastructure disruptions
    This ensures self-healing behavior without manual intervention.
2. Declarative desired state: Kubernetes continuously reconciles the actual state of the system with the declared desired state. If a pod crashes, is evicted, or is manually deleted, it is automatically recreated. This guarantees service continuity and prevents configuration drift.
3. Rolling updates and zero-downtime upgrades: Kubernetes supports rolling deployments with:
    a. Readiness and liveness probes
    b. Controlled surge and unavailable limits
    c. AccuKnox can roll out security updates, bug fixes, or feature enhancements without service disruption.
4. Horizontal scalability: Kubernetes Horizontal Pod Autoscaler (HPA) enables control plane components to scale based on:
    a. CPU / memory utilization
    b. Custom metrics (request rate, queue depth)
    This allows the control plane to handle sudden spikes in customer activity or telemetry ingestion.

5. Isolation and blast-radius reduction: Namespaces, network policies, and resource quotas isolate control plane components. Failures are contained to specific microservices and do not cascade system-wide.

Where can I check the uptime of the AccuKnox Production environments?

status.accuknox.com

SaaS Production Deployment on Cloud-hosted Environments

AccuKnox operates its SaaS production control plane on cloud-hosted infrastructure, leveraging the reliability guarantees of hyperscale cloud providers. Key characteristics of the SaaS environment

1. Multi-zone deployment: Kubernetes clusters span multiple availability zones. Control plane workloads are distributed across zones using:
    a. Pod anti-affinity rules
    b. Topology spread constraints
    This protects the platform from:
        i. Single-node failures
        ii. Zone-level outages
2. Managed cloud services: Wherever possible, AccuKnox uses managed cloud services for:
    a. Databases
    b. Object storage
    c. Load balancers
    Managed services provide built-in replication, automated failover, and operational SLAs.
3. Secure and isolated runtime: SaaS environments are isolated from customer environments. Strong identity, access control, and network segmentation protect the control plane from lateral movement or accidental exposure.
4. Elastic infrastructure: Cloud infrastructure allows:
    a. Rapid scale-out during peak demand
    b. Scale-in during low utilization
    This ensures consistent performance without over-provisioning.

Stateless Microservices Architecture

A key reliability design principle of the AccuKnox control plane is that most microservices are stateless.

1. Fast recovery: Stateless services do not store local session or application state. If a pod crashes or is restarted:
    a. A new instance can be started immediately
    b. No data reconstruction or replay is required
    c. This minimizes mean time to recovery (MTTR).
2. Easy horizontal scaling
    a. Any instance can serve any request.
    b. Kubernetes can freely add or remove replicas based on load.
    c. Load balancers distribute traffic across healthy pods.
3. Resilience to restarts and upgrades: Routine events such as:
    a. Node reboots
    b. Image upgrades
    c. Security patching
    do not impact correctness or continuity. Services resume processing automatically after restart.
4. Simplified failure domains: Failures are limited to individual pod instances.
    a. No tight coupling exists between service instances.
5. Clear separation of concerns:
    a. Application logic is separated from persistence.
    b. All durable state is externalized to shared data services.

This design ensures that service restarts are treated as normal operational events, not exceptional failures.

Centralized data layer

1. Common databases
    a. Metadata
    b. Configurations
    c. policy definitions, and
    d. operational states are stored in shared databases. Databases are deployed in:
    e. Highly available configurations
    f. Multi-replica or managed cloud database services
    This ensures data consistency across all control plane components.
2. Shared object storage.
    a. Large artifacts such as:

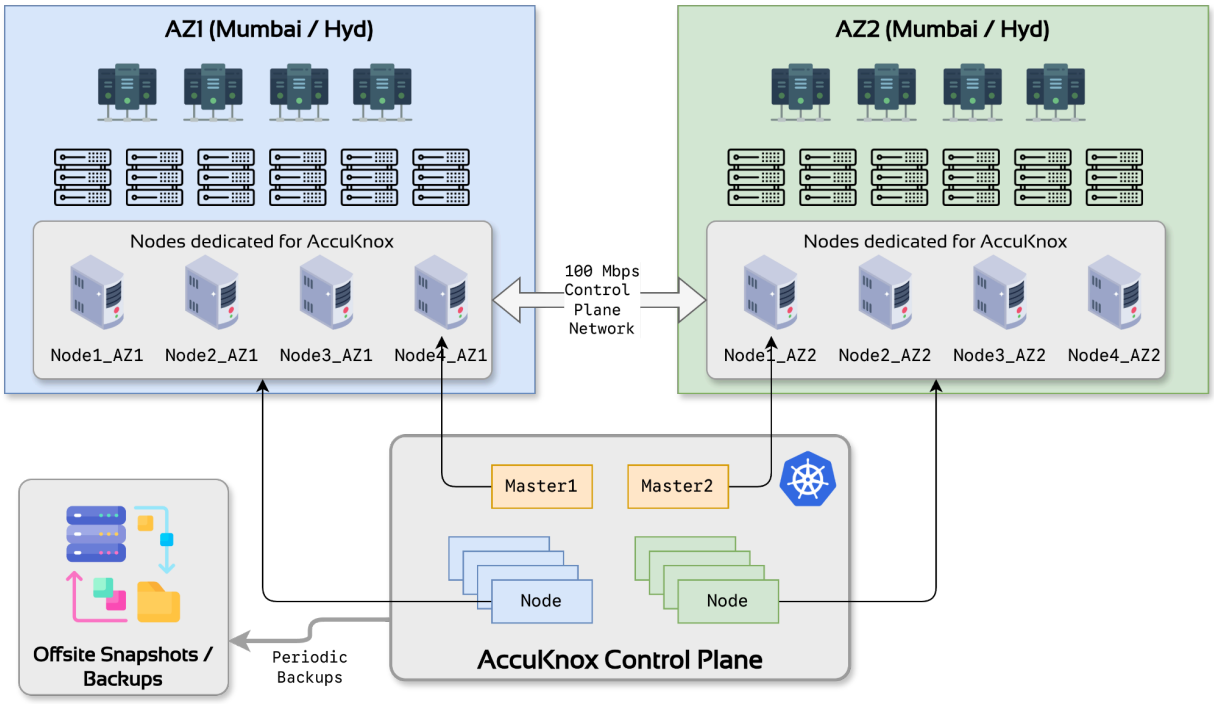
   i. Reports
   ii. Logs
   iii. Telemetry archives
   iv. Exported compliance data are stored in durable object storage.
b. Object storage provides:
   i. High durability
   ii. Built-in replication
   iii. Cost-efficient scaling

Backup and restore strategy

1. Automated backups
   a. Databases are backed up on a scheduled basis.
   b. Backups are:
      i. Encrypted
      ii. Stored in isolated backup locations
      iii. Retained based on defined retention policies
2. Restore validation
   a. Backup restore procedures are periodically tested.
   b. Ensures that backups are usable and recovery objectives are met.
3. Disaster recovery readiness. In case of a major regional outage:
   a. Control plane services can be redeployed
   b. Data can be restored from backups
   c. This enables controlled recovery with minimal data loss.

Multi AZ, Site, Region Deployment

Overview



AccuKnox supports high availability and disaster resilience by running in a Kubernetes multi-site / multi-AZ deployment model. The core idea is:

- Kubernetes control plane nodes (masters) and worker nodes are placed across multiple Availability Zones (AZs) and, where required, across multiple regions or sites (e.g., Region-A and Region-B, or DC-1 and DC-2).
- Stateful layers (database, object storage, message bus, caching) are deployed with multi-AZ replication and (optionally) cross-region replication, so the platform tolerates AZ failures and can recover from site/region failures with minimal operational disruption.
- Traffic routing (north-south and east-west) uses zone-aware load balancing, health checks, and failover policies, ensuring requests land on healthy instances as the topology changes.

This model enables:

- AZ failure tolerance (typical HA requirement)
- Reduced blast radius (isolation at zone/site level)
- Improved continuity for control plane + data plane services
- Operational flexibility for maintenance, upgrades, and incident response

Target Topology (Multi-AZ + Optional Multi-Region)

Multi-AZ (single region, multiple zones)

A common production topology is a single Kubernetes cluster spanning 3 AZs:

- Control plane: 3 or 5 control plane nodes spread across AZs (odd number for quorum).
- Workers: node groups in each AZ (balanced capacity).
- Platform services: stateless services spread across AZs; stateful services deployed with multi-AZ replication.

This provides high availability for most failures (one AZ down) without the complexity of cross-region networking.

Multi-site / Multi-region (two regions/sites)

For stricter continuity requirements, AccuKnox can be deployed across two regions (or sites) in one of two patterns:

Pattern A: Active-Active (multi-site serving traffic)

- Both regions accept traffic.
- Data layer uses synchronous replication only within a region (to avoid latency penalties) and asynchronous replication across regions.
- Requires more careful design for consistency and conflict handling.

Pattern B: Active-Passive (primary + DR)

- Region-A serves traffic; Region-B is a warm standby.
- Cross-region replication keeps Region-B current.
- Failover is simpler and typically preferred when strong consistency is required.

Control Plane Distribution and Quorum

Why master separation matters

The Kubernetes control plane includes etcd, API server, scheduler, and controller managers. In a multi-AZ model:

- Control plane nodes are placed across AZs so the cluster retains quorum and continues scheduling even if one AZ fails.
- etcd runs as an odd-sized cluster (3 or 5 members). Losing one AZ should not lose quorum.

Practical placement guidance

- 3 control plane nodes: one per AZ (common).
- 5 control plane nodes: used for larger clusters or when extra fault tolerance is desired.
- Ensure anti-affinity rules and topology spread constraints prevent control plane pods from co-locating in a single AZ.

Multi-region caution

A single Kubernetes cluster stretched across regions can introduce:

- high latency to etcd
- increased leader election churn
- degraded API responsiveness during network partitions

For most enterprises, the recommended multi-region approach is two clusters (one per region) with federated or replicated platform components, rather than one stretched cluster; unless the networking and latency requirements are well controlled.

Worker Node Distribution and Service Placement

Zone-balanced compute pools

Worker nodes are provisioned in each AZ via:

- managed node groups / autoscaling groups per AZ
- node labels and taints to separate “platform” vs “workload” capacity

Scheduling and resilience controls

To make multi-AZ effective, AccuKnox workloads use:

- Pod anti-affinity for replicas (do not place same replica in same AZ/node)
- TopologySpreadConstraints across topology.kubernetes.io/zone
- PodDisruptionBudgets (PDBs) so upgrades/maintenance do not take down all replicas
- Priority classes for critical control-plane services to preempt less critical pods when capacity is constrained

Failure Scenarios and Behavior

AZ failure (single region)

Expected behavior:

- ingress shifts traffic away from the failed AZ
- deployments maintain desired replica count by rescheduling pods in remaining AZs
- database fails over if primary was in affected AZ
- system remains available with reduced capacity

Control-plane node loss

Expected behavior:

- remaining control plane nodes keep quorum
- API remains available; scheduling continues
- autoscaling may reprovision nodes

Region/site failure (multi-region)

Depending on chosen mode:

- Active-Passive: promote standby DB (or switch DB endpoints), redirect DNS/traffic to DR, scale up DR workers
- Active-Active: route traffic to surviving region; ensure data replication and reconciliation guarantees are met

Example "Multi-AZ Readiness" Checklist

- ☐ Control plane nodes distributed across 3 AZs (odd quorum)
- ☐ Workers balanced across AZs; autoscaling enabled
- ☐ Topology spread constraints + anti-affinity configured for all critical services
- ☐ DB multi-AZ replication + tested failover
- ☐ Message bus replicated across AZs (RF≥3), min ISR configured
- ☐ Ingress replicated across AZs; LB cross-zone enabled
- ☐ Backups + restore tested; DR runbook validated
- ☐ Monitoring per AZ; alerts for imbalance and failover events

Interaction between Customer Plane and AccuKnox Control Plane

AccuKnox needs visibility into the customer's assets metadata and, in some cases, needs the ability to probe the workloads. AccuKnox also provides an optional runtime security engine that operates in the customer data plane and ensures that runtime attacks such as privilege escalation, data exfiltration, and tampering/poisoning of data/configs are prevented.
Overall, the asset protection is done using the following models:

Category	Feature	Deployment Model	Onboarding	Permissions	Where is the Playbook hosted?
Cloud Security	CSPM - OrgUnits	Agentless	Cross-Tenant roles	Read + Security Audit	AccuKnox Control Plane hosted playbooks
	CSPM - Individual Cloud Accounts	Agentless	Shared key/id	Read + Security Audit	AccuKnox Control Plane hosted playbooks
	CDR	Agentless	Cloud Formation or Terraform script to be executed from the customer user	Read for logs + Create serverless function + S3 bucket one time for serving logs locally	Customer Cloud env
Workload Protection	KSPM scans	Agentless	User deploys a job using helm	Read access across namespaces	Kubernetes Jobs
	Runtime Security	Agent based (KubeArmor)	User deploys a daemonset using helm	Ability to use eBPF on the k8s worker nodes	K8s Daemonset, services
Nodes/Hosts security	Vulnerability Scanning on Nodes	Agentless	Using terraform script	Create VM snapshot and scan	Customer hosted VM

	Vulnerability Scanning on Nodes	Agent based	Deploy local agent using Knoxctl cli tool	Scan the files / folders	knoxctl cli tool
ASPM	SAST, Secrets, IaC scan	Agentless	Collectors based	Read access to code repos	AccuKnox hosted collector or customer hosted collectors
	SAST, Secrets, IaC scan	CI/CD Pipeline (DevOps)	Users create CI/CD workflows	Read access to code repos	Customer CI/CD hosted
	DAST	Collectors model	For authenticated scans, the user has to configure creds.	Read/Probe of the endpoints. It is sometimes required that the customer whitelists AccuKnox NAT Egress Gateway IP addresses in their SaaS hosted endpoints (such as Netlify, Cloudflare).	AccuKnox Hosted or Customer Hosted
AI	Cloud Hosted Model Red Teaming	Agentless	Onboarding same as cloud accounts/orgs	Read and security audit	AccuKnox host playbooks
	Onprem Hosted Model Red Teaming	Collectors model	Provide API endpoint + access key to read models metadata	Read models metadata.	AccuKnox Hosted or Customer Hosted
	Prompt Firewall	SDK Model	App dev team has to use the AccuKnox SDK to sanitize prompts/responses.	NA	Hosted in customer code based
		AI Gateway based	LiteLLM or BiFrost	Prompts/Responses data	AccuKnox or Customer hosted.

			configuration change		AccuKnox provides an AI gateway by itself.
		Cloud hosted	Azure CP Studio, or APIM based integration	Prompts/Responses data	Customer hosted.

AccuKnox SLAs

Recovery Time Objective (RTO)

AccuKnox is designed with resiliency and operational continuity as core non-functional requirements. In the event of a major service disruption—such as infrastructure failure, regional outage, or critical platform incident—AccuKnox maintains a Recovery Time Objective (RTO) of **6 hours**. This defines the maximum acceptable duration within which the AccuKnox platform is restored to an operational state.

What the 6-Hour RTO Means

- AccuKnox guarantees restoration of core control plane services within 6 hours of incident declaration.
- The RTO applies to platform availability, including policy management, telemetry ingestion, dashboards, and enforcement orchestration.
- Runtime enforcement on customer workloads continues wherever agents are already deployed, ensuring fail-secure behavior even during control plane recovery.

Recovery Point Objective(RPO)

Recovery Point Objective (RPO) defines the maximum acceptable amount of data loss measured in time, in the event of a system failure, service disruption, or disaster. It represents the point in time to which data must be restored to resume normal operations after an incident.

RPO is determined by an organization's data-loss tolerance, which varies based on business criticality, regulatory requirements, and operational impact. A lower RPO implies more frequent data protection mechanisms and higher operational complexity, while a higher RPO allows greater flexibility in backup frequency.

AccuKnox RPO Commitment

AccuKnox commits to an RPO of 24 hours under its standard Service Level Agreement (SLA). This means that, in the event of a catastrophic failure affecting the AccuKnox control plane or managed data stores, AccuKnox guarantees restoration of customer data to a state that is no more than 24 hours old from the time of the incident.

Scope of RPO Coverage

The 24-hour RPO applies to the following categories of data within the AccuKnox platform:

- Control plane configuration data
- Security policies, enforcement rules, and metadata
- Tenant and account configuration information
- Audit logs, alerts, and event metadata (subject to retention policy)
- Platform state required for service continuity
- Runtime enforcement on customer clusters continues to operate independently and is not directly impacted by control plane recovery operations.

Escalation Matrix

Severity Levels & Definitions

Severity Level	Definition	Examples
P1 (Critical)	AccuKnox is down and inaccessible. Severe service failure or degradation affecting multiple users.	• Users cannot access a business-critical application. • Consistent ”page not found” errors prevent login.
P2 (High)	Partial service failure or mild degradation. Some, but not all, business resources are accessible.	• Admin console write-access issue. • Users experience slow access, occasional ”page not found” errors. • Bug causing significant impact to service or integration.
P3 (Medium)	Minor service impact, affecting individual users or non-critical third-party applications.	• One user is unable to access an application. • Difficulty integrating new business applications.
P4 (Low)	Minor impact or feature enhancement request.	• How-to inquiries. • Feature enhancement requests.

Personnel Contacts for Support Levels

Severity Level	Personnel To Contact	Email	Contact Number
L1 (Low)	Udit, Aditya - Sr Solution Engineers @AccuKnox	udit@accuknox.com, adityaraj@accuknox.com	8755357024, 6207338299
L2 (Medium)	Gaurav - Product Manager @AccuKnox	gaurav.mishra@accuknox.com	9953525525
L3 (Critical)	Rahul Jadhav - CTO @AccuKnox	r@accuknox.com	9538045248

Support Hours

Support Level	Support Hours
Silver (Standard)	12x5 (6:00 AM – 6:00 PM PT, Mon-Fri, excluding US holidays)
Gold (Premium)	12x7 (Extended Phone Support)
Platinum (PremiumPlus)	24x7 (Full Round-the-Clock Support)

SLA Response Times Based on Support Packages

Severity Level	Silver (Standard)	Gold (Premium)	Platinum (PremiumPlus)
P1 (Critical)	**First Response:** 4 Hours **Updates:** 1 Business Day	**First Response:** 1 Hour **Updates:** 2 Hours	**First Response:** 1 Hour **Updates:** 2 Hours
P2 (High)	**First Response:** 1 Business Day **Updates:** 2 Business Days	**First Response:** 2 Hours **Updates:** 8 Hours	**First Response:** 2 Hours **Updates:** 8 Hours

P3 (Medium)	**First Response:** 2 Business Days **Updates:** 3 Business Days	**First Response:** 2 Hours **Updates:** 48 Hours	**First Response:** 2 Hours **Updates:** 48 Hours
P4 (Low)	**First Response:** 2 Business Days **Updates:** 3 Business Days	**First Response:** 8 Hours **Updates:** 48 Hours	**First Response:** 8 Hours **Updates:** 48 Hours

Periodic Updates to AccuKnox Control Plane

AccuKnox control plane update operations are designed around two independent cadences:

- Microservices/platform updates (feature releases, performance improvements, bug fixes): typically every 4–8 weeks
- Hot Patches if any might need to be updated more frequently. This is on case to case basis.
- Vulnerability intelligence updates (CVE feeds, EPSS/KEV, package metadata, exploit signals): typically weekly or more frequently

To avoid coupling operational risk across these two cadences, AccuKnox treats them as separate delivery streams with separate validation, rollout, and rollback paths.

References

- AccuKnox Global Production Regions Health Monitoring
- AccuKnox completes AWS Foundational Technical Review
- AccuKnox Help Documentation
- AccuKnox FAQs