Offline but Not Blind : Observability in Air-Gapped Kubernetes Environments

Build your own Observability stack as a First-Class Workload in Isolated Clusters

Manoj Kumar Sardana

Agenda

The Air-Gap Paradox - High-velocity applications in low-connectivity zones.

The Self Hosted Blueprint - OpenTelemetry + LGTM Distributed Mode

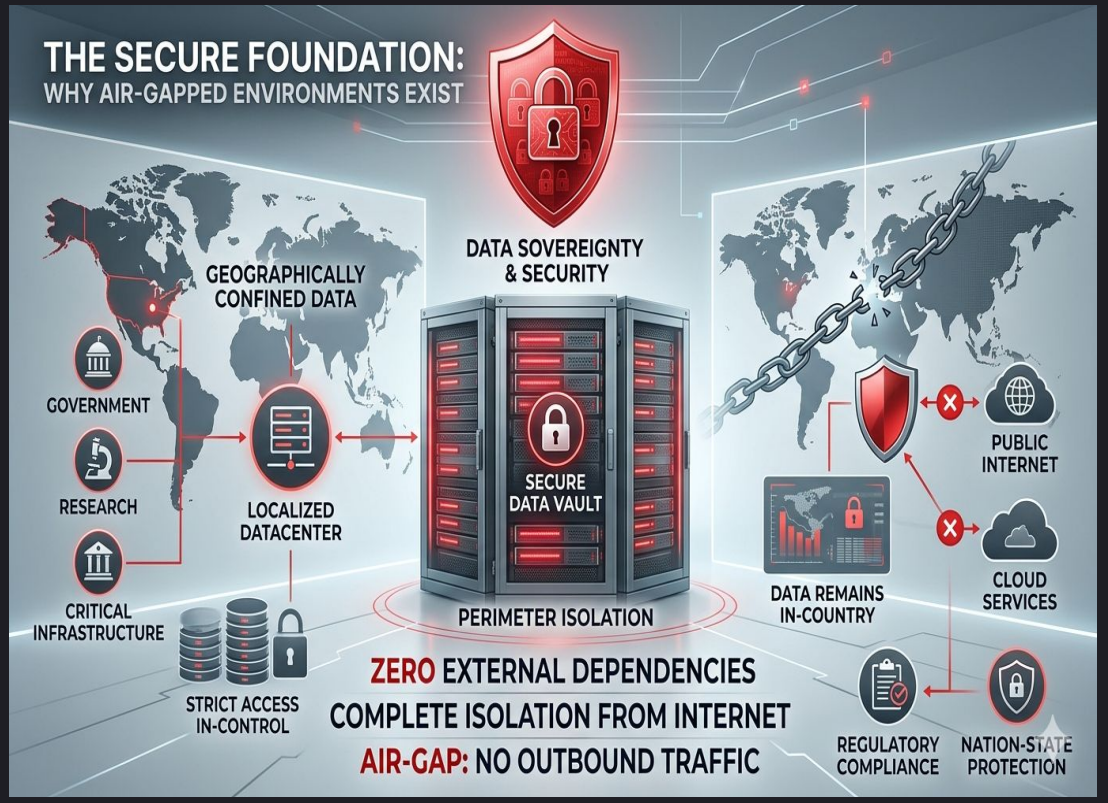
The Local Ecosystem - Managing images, storage, and configurations

Day-2 Governance - Observing the Observer, Compaction

Why Air-Gapped Environments Exist

Strict Data Sovereignty

- **Complete Network Isolation:** No internet access, no external DNS resolution, no public inbound/outbound traffic.
- **Target Industries:** National Defense, Aerospace/Research, Critical Infrastructure, Financial Core Engines.
- **The Core Tradeoff:** Hardening the perimeter reduces the external attack surface, but eliminates the convenience of cloud services.



The Platform: Self-Hosted K8s as the Norm

The Myth: Isolated systems are slow, monolithic, and rarely change.

The Reality: Air-gapped applications still require **modern cloud-native capabilities**:

- Microservices scaling and dynamic container elasticity.
- Strict High Availability (**99.99% uptime targets**).
- Rapid, automated incident response.

Why Observability is Non-Negotiable

Microservices are Black Boxes

No External Lifeline

Internal Visibility is Everything

Shift in Paradigm: Saas vs Self-Hosted

The Architectural Divide

Feature	SaaS Observability	Self-Hosted Air-Gap
Data Boundary	Shipped outside the network perimeter	Strictly confined to local cluster storage
Resource Overhead	Minimal local footprint (only agents)	Consumes internal CPU, RAM, and PVs
Operational Effort	Outsourced to third-party vendor	Handled natively by local cluster admins
Scale Adjustments	Dynamic billing adjustment	Hard-capped by physical node limitations

The Operational Hurdles of the Air-Gap

Self hosted Observability

Storage Constraints

Ecosystem Gap

Maintenance

Reliability

The Blueprint: LGTM Stack + OpenTelemetry

The Open Source Pillars - The Stack and the GLUE

L

Loki

LOGS

Indexes labels, not content. LogQL queries. Object storage economics with label-first queries.

G

Grafana

THE PANE

One UI over all three. Correlates signals via exemplars, trace-to-logs, data links.

T

Tempo

TRACES

Trace storage on object store alone. TraceQL search. Built for 100% error retention.

M

Mimir

METRICS

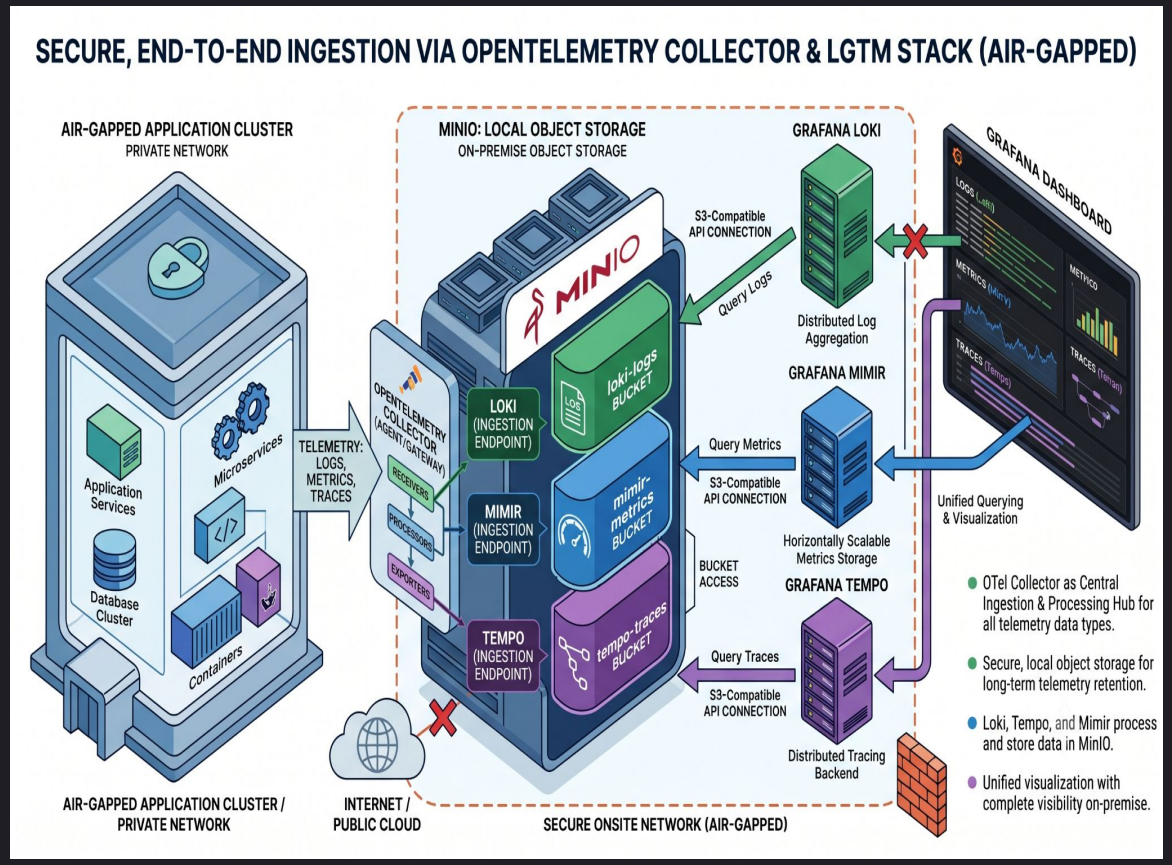
Horizontally scalable Prometheus. PromQL compatible. Billions of active series.

Built from the ground up to operate as native, containerized Kubernetes microservices.

Storage Solution: S3-Compatible Local Storage

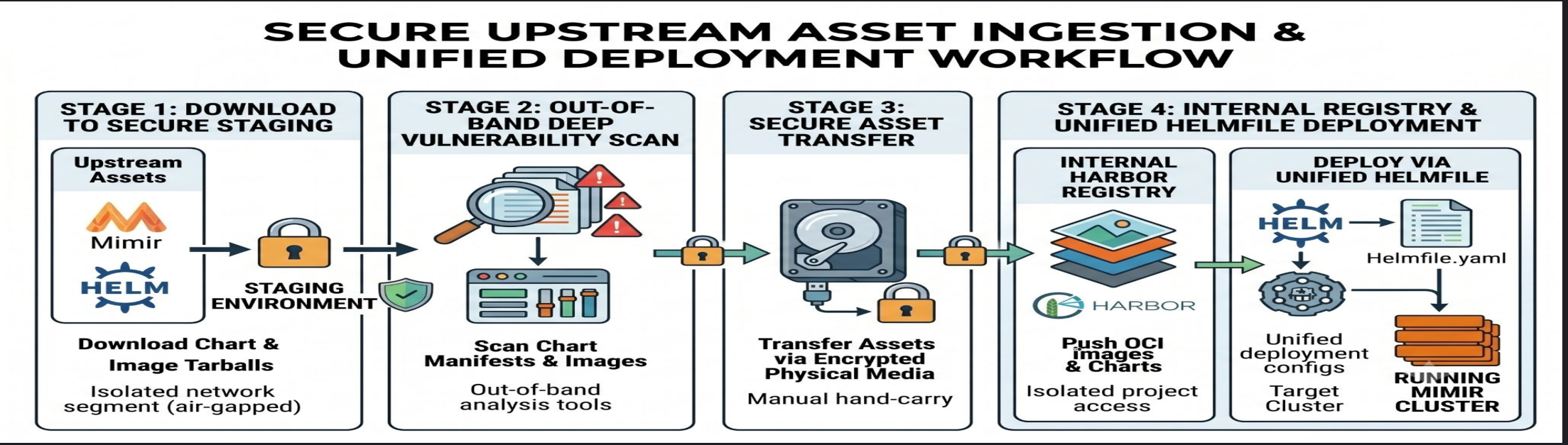
Mimicking Cloud Backends Locally

- **The Dependency:** Loki, Tempo, and Mimir require an object store to archive data blocks.
- **The Air-Gapped Solution:** Self-hosted S3 API abstractions.
- **Storage Contenders:**
  - **MinIO:** High brand enterprise recognition, strictly compatible S3 API layer.
  - **Ceph**: On-premises object store for the LGTM stack. Ceph provides a native, highly scalable S3-compatible API



Registry & Repository: Container Images & Helm Charts

- **Image Whitelisting:** Every single container image must be pre-scanned and verified.
- **Local Registries (Harbor):** Acts as the internal repository for container images and Helm charts (OCI).



- **The Air-Gapped Air-Lock Workflow:**

Registry & Repository: Container Images & Helm Charts

```
docker pull <image_name>:<tag>
docker save -o my_image.tar
  <image_name>:<tag>
```

```
helm pull grafana/mimir-distributed
 git  clone grafana/helm-charts
 helm  package ./grafana
```

```
docker load -i my_image.tar
docker tag <image_name>:<tag>
<local_registry_address>/<image_name>:<tag>
docker push
<local_registry_address>/<image_name>:<tag>
```

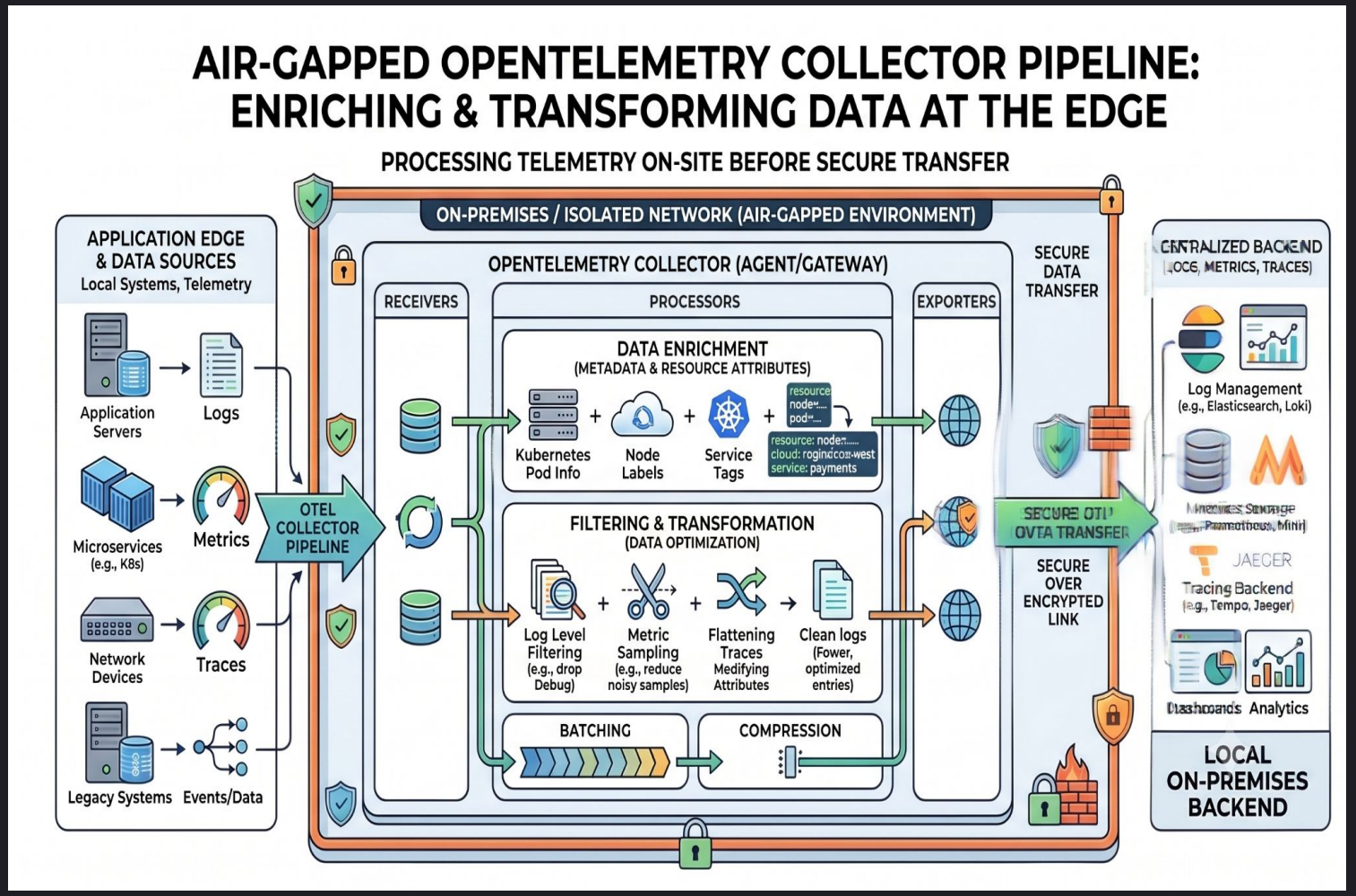
```
helm registry login <local registry URL>
--username <username> --password
<password>
helm push grafana-12.4.5.tgz <local registry
URL>/<projectName>
```

Standardizing Ingestion with OpenTelemetry (OTel)

No Vendor Lock-In

The OTel Collector Architecture

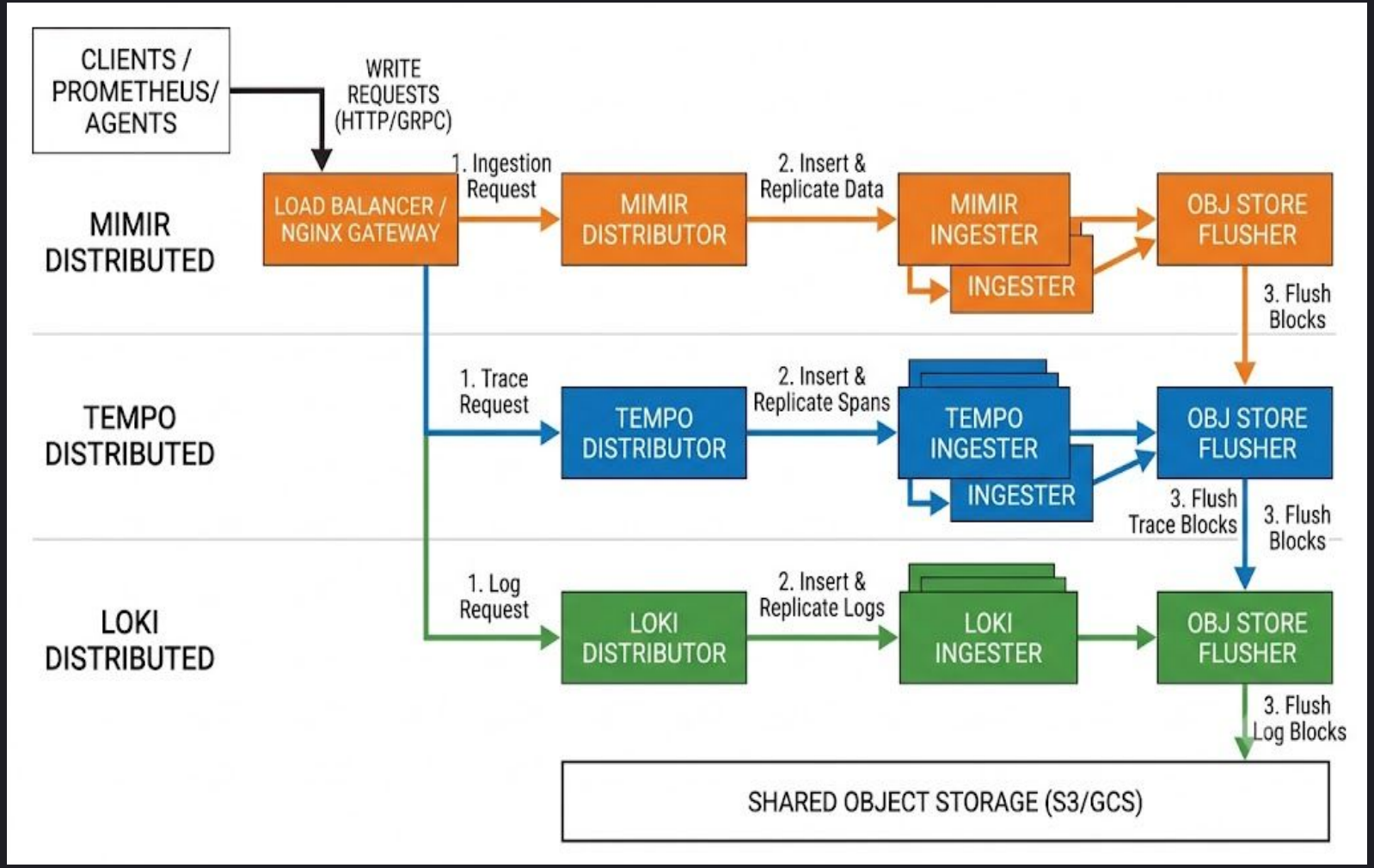
Air-Gapped Efficiency



The Universal Proxy

Architecture of Resilience: LGTM Distributed Mode

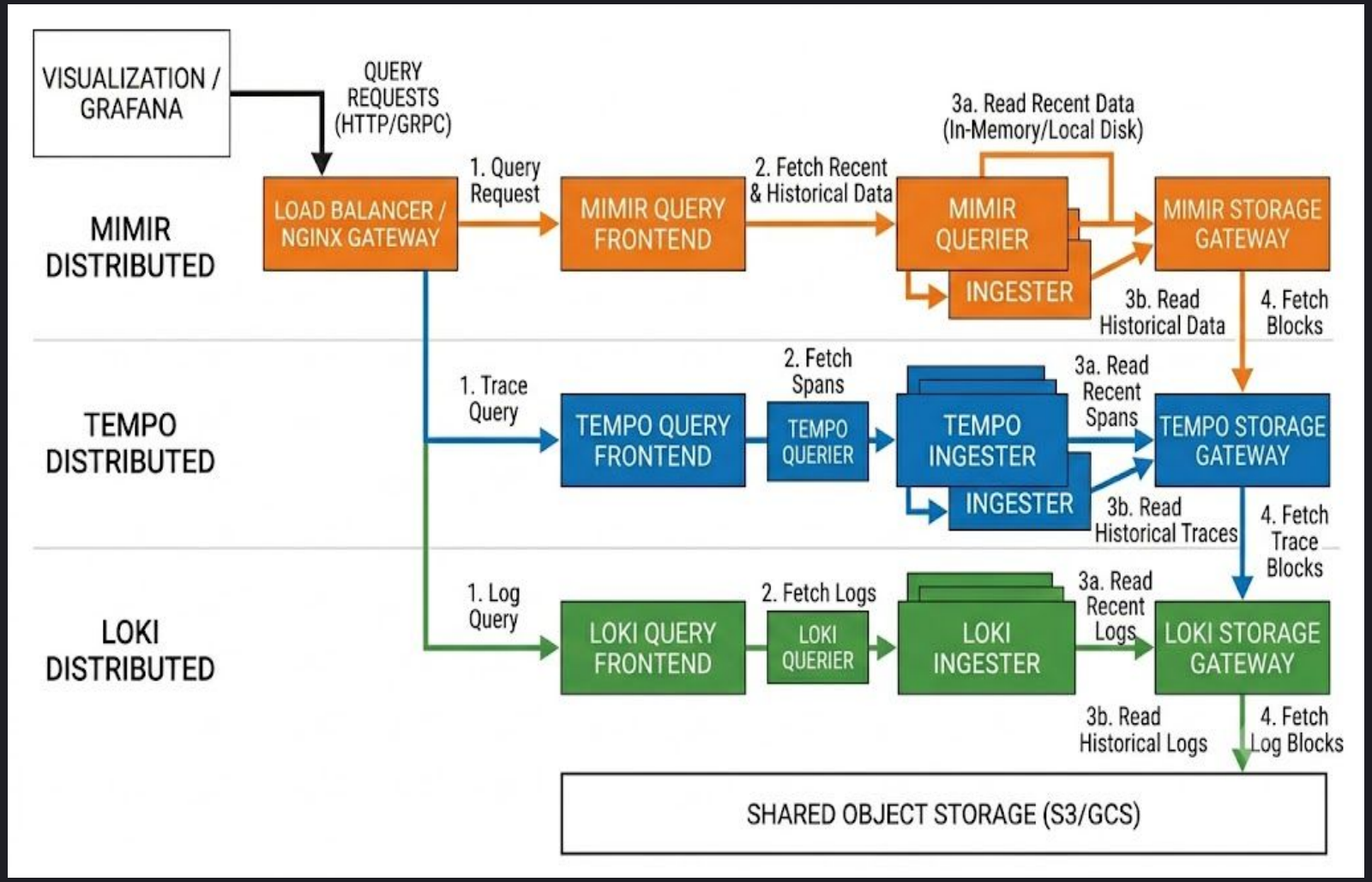
- Gateway
- Distributor (Producer)
- Ingesters (Consumers)
- Object Store Flusher



Ingestion Path

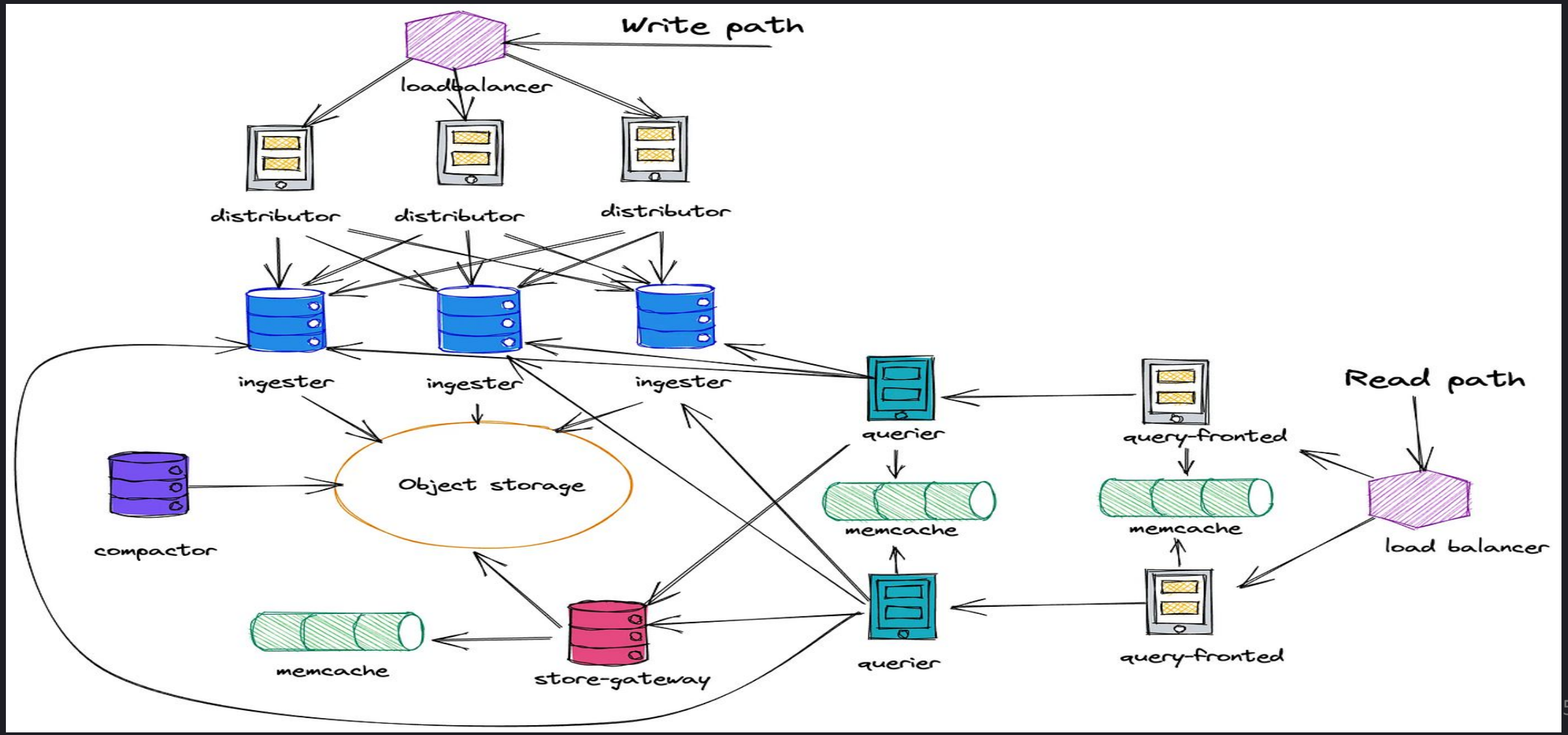
Architecture of Resilience: LGTM Distributed Mode

- Gateway Entry
- Query Frontend
- Querier
- Storage gateway



Read Path

Architecture of Resilience: LGTM Distributed Mode



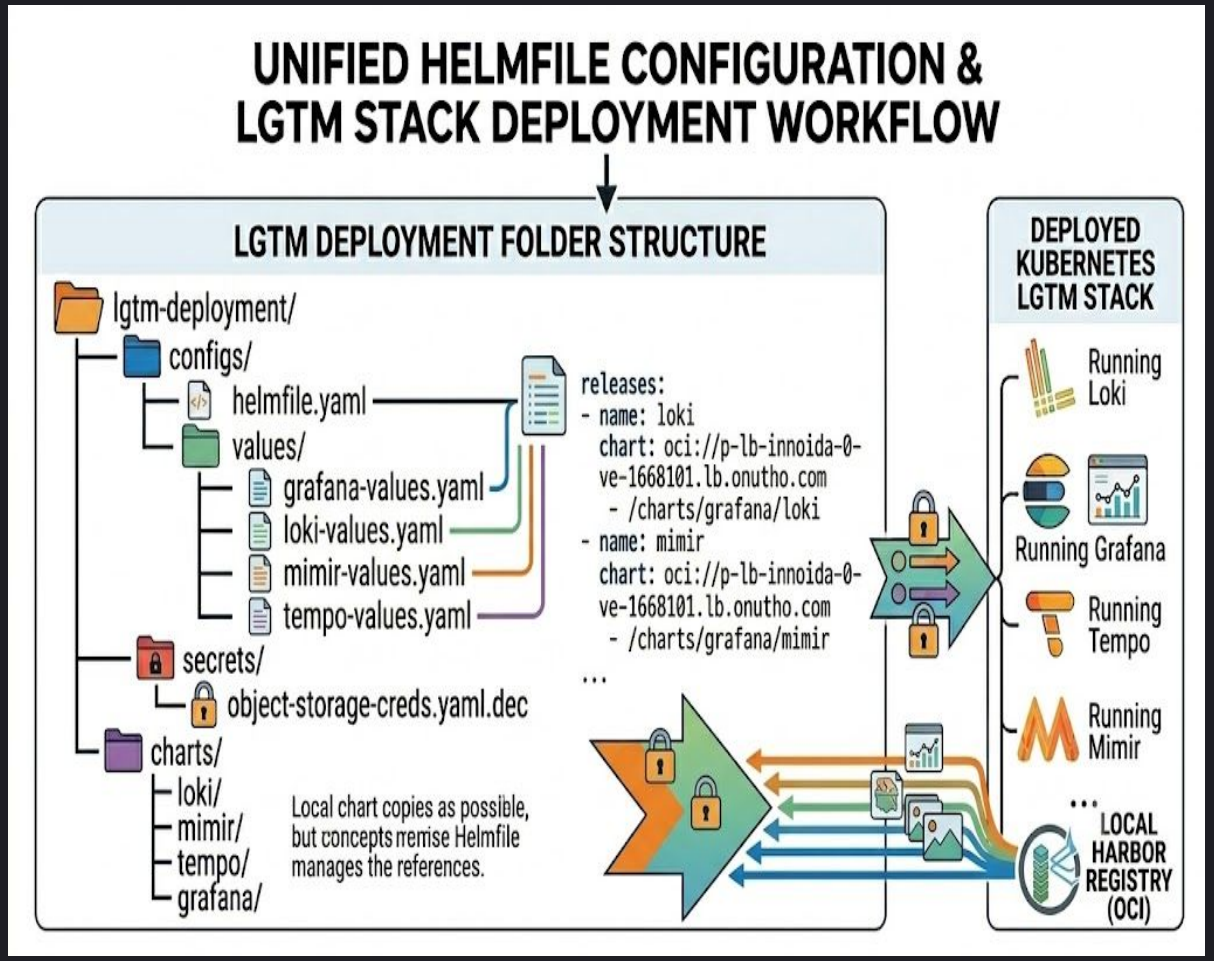
LGTM deployment : helm & helmfile

Declarative "Stack-as-Code"

- **helmfile based Deployment -** A declarative spec for deploying multiple Helm charts. It ensures your versions, values, and namespaces are version-controlled and reproducible. Instead of four separate helm install commands, you use one helmfile apply.

  Benefits:

  - **Dependency Management:**
  - **Environment Sync**
  - **Drift Detection**



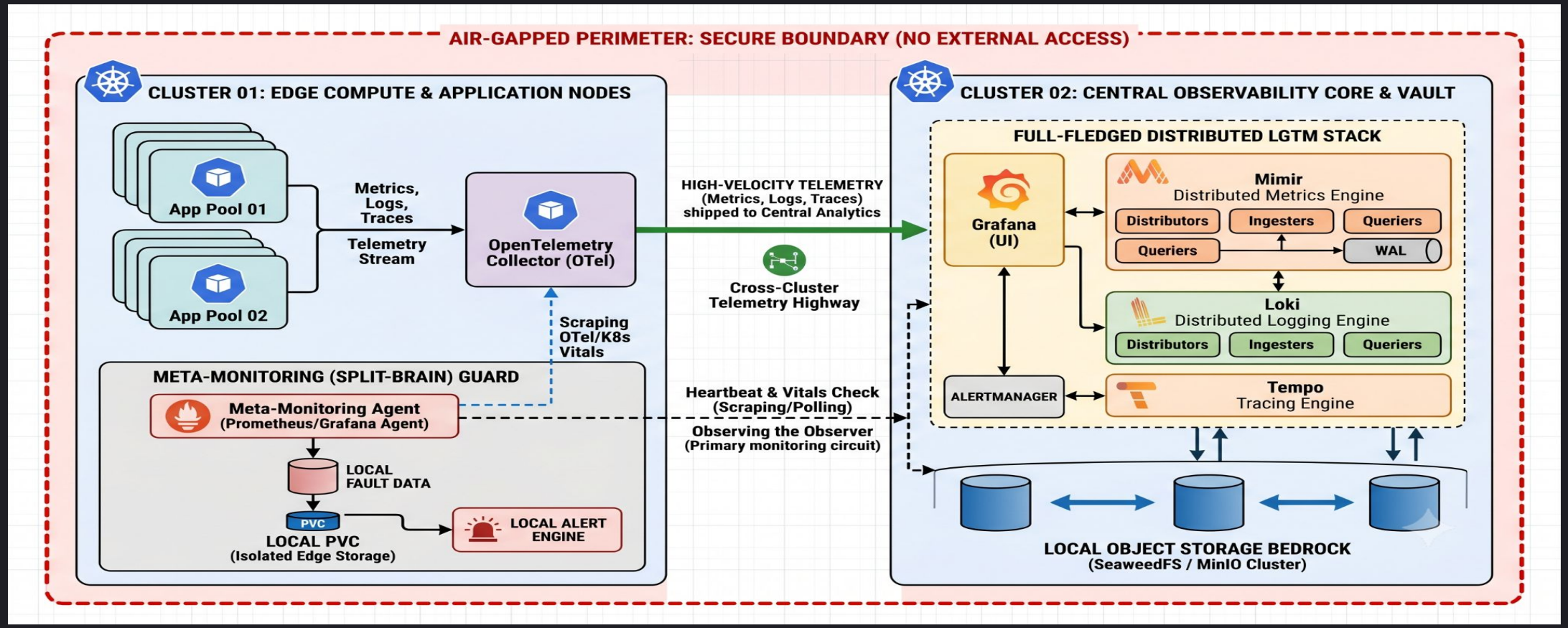
Day 2 Governance: Managing Retention & Compaction

Preventing Storage Exhaustion

- **The Strategy:** Local object storage is finite. You must configure active cleanup.
- **The Mimir/Loki Compactor:** The internal janitor that deduplicates raw 2-hour blocks into large 24-hour blocks.
- **Safety Configuration Overrides:**
  - Utilize `deletion_delay` (e.g., 12 hours) as a safety buffer before physical disk erasure.
  - Configure strict global ceilings (`compactor_blocks_retention_period: 90d`).
- **Performance Note:** Assign fast SSD storage to Compactor Persistent Volumes for intense merge operations.

Day 2 Governance: Reliability & Availability

Observing the Observer- Split Brain Architecture



The Strategy: Having a light weight Observability stack in application cluster.

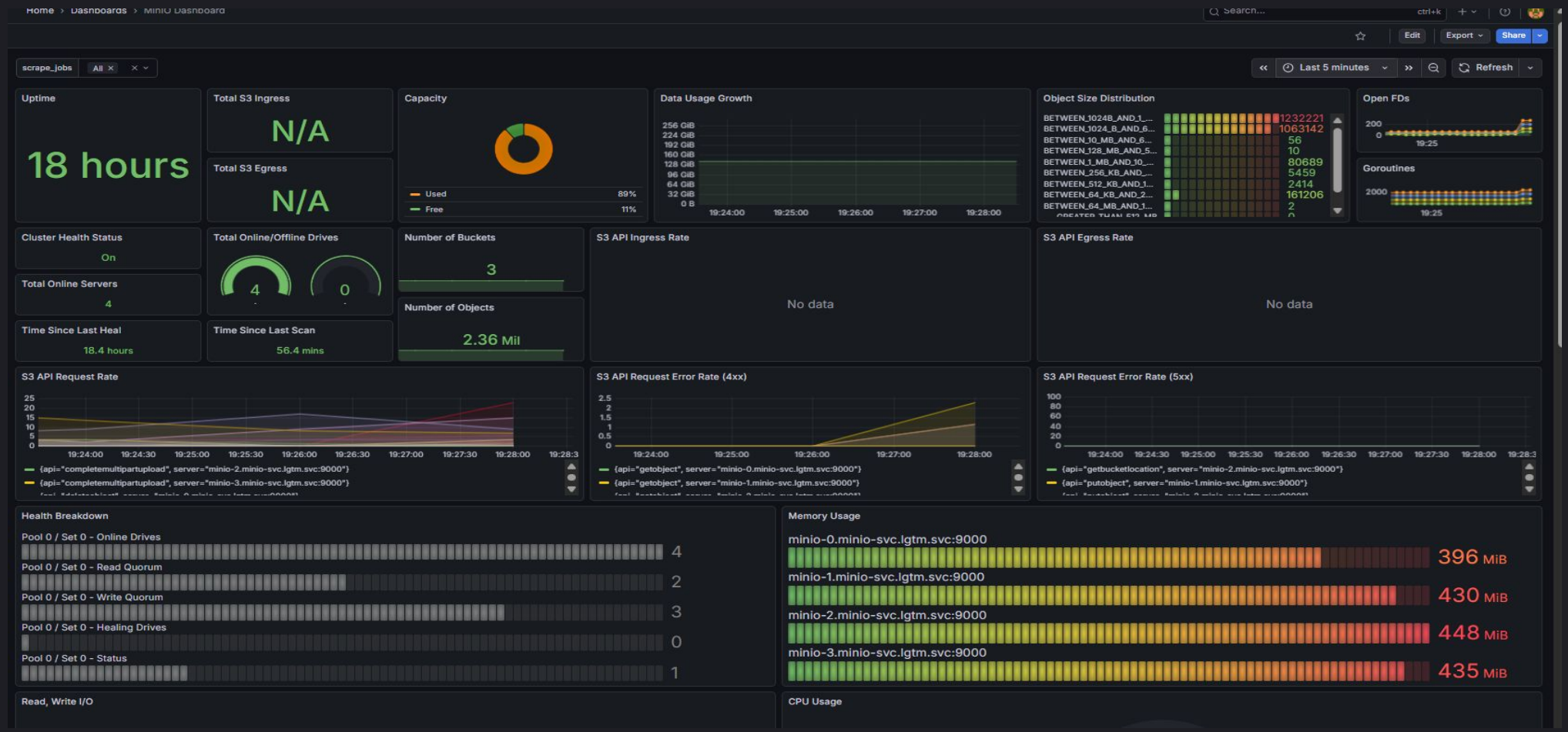
Day 2 Governance: Reliability & Availability

Observing the Observer- Split Brain Architecture - Key Indicators

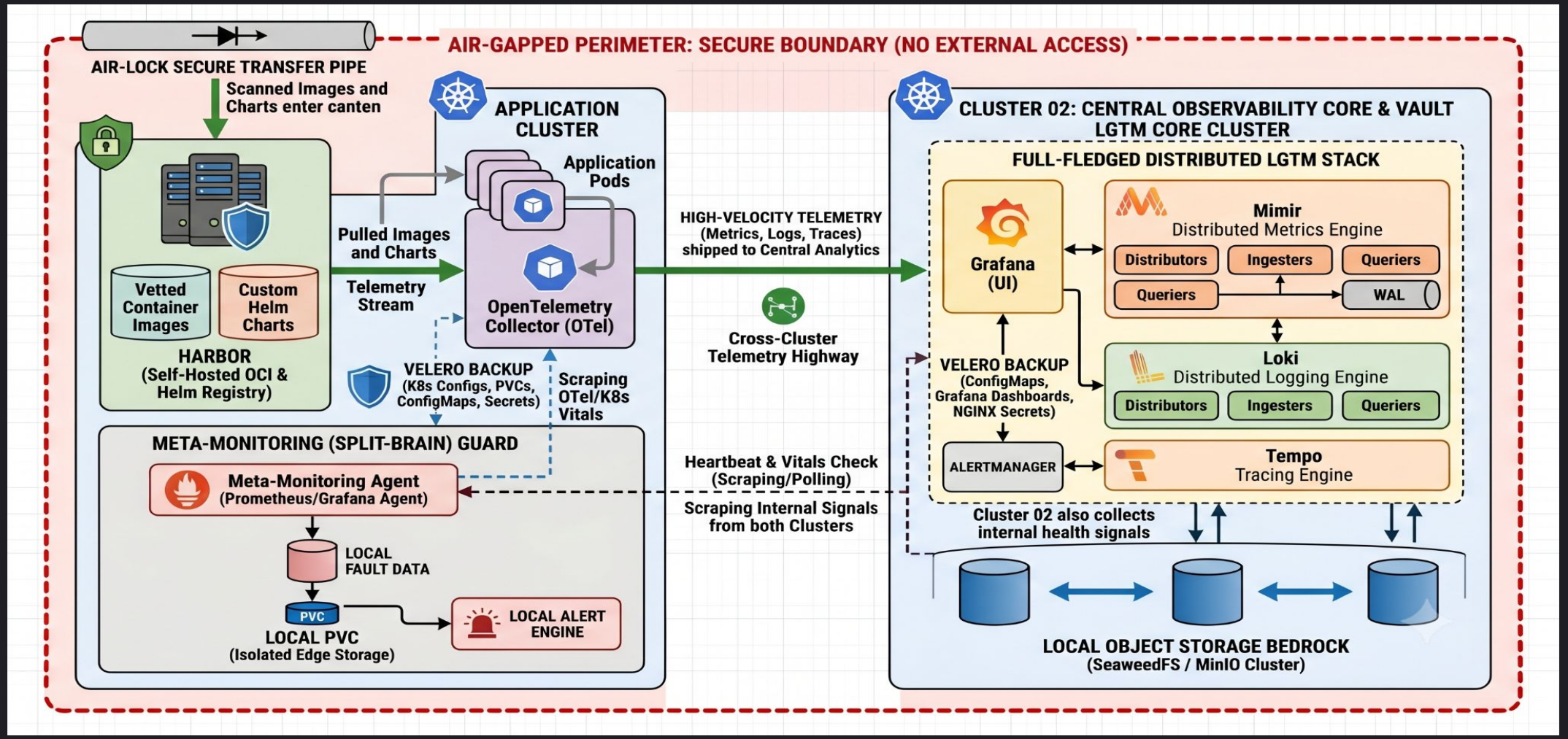
- **Collector Vitals** - Ingestion drops, Pipeline related drops and exporter failure
- **Mimir Vitals** - Distributor Errors, flush queue and hash ring integrity
- **Loki Vitals** - Discarded sample/Drop rate, flush Queue/ingestor memory saturation
- **Compactor Vitals** - Compactor Lags
- **Tempo Vitals** - Span drop rates and block queue
- **Minio Storage Vitals** - API operation failure, Storage IOPS latency, Storage size/usage
- **Complete Silence** - Meta/Split brain being silent on Signals.

Day 2 Governance: Reliability & Availability

Observing the Observer- Split Brain Architecture - Key Indicators



The Production Ready Architecture Blueprint



Key Architecture Decision

Workload Equality
Build for reliability & Scale, treat your Observability platform at part with your workload

Secure the Front Door
Every image and helm chart is vetted before its deployed

Automate the Life Cycle
automatic retention policy, auto scale t k6s

Use the power of processors
Otel provide full control on the monitoring data

Observe the Observer
Observe the observability tool

Demo : Bring Mimir image and Helm chart locally & deploy


LOOKING
FOR AN
ANSWER?

Summary

Takeaways for the Air-Gapped Administrator

1. **Workload Equality:** Treat your observability tools with the exact same resource priority, HA replication, and architectural respect as your primary application stack. Split brain architecture helps.
2. **Automate the Life Cycle:** Local storage fills up quickly. Configure strict compaction, retention thresholds, and automated backup schedules from Day 1.
3. **Use the power of processors**: OpenTelemetry processor provide a lot of complex processing which can help mimic some of the backend platform capabilities of managed services.
4. **Secure the Front Door:**
5. **Observe the Observer**