Kubernetes Observability Architecture with Grafana, Prometheus, and Loki

Author: Arman Singh Kshatri

March 31, 2025

1 High-Level Design Decisions and Tradeoffs

This observability architecture leverages **Kubernetes** to deploy and manage key monitoring components:

- **Grafana**: Visualization and dashboarding.
- **Prometheus**: Metrics collection and storage.
- **Loki**: Log aggregation, storage, and querying.

1.1 Design Choices

- **Specialized Applications**: Prometheus for metrics, Loki for logs, and Grafana for visualization, ensuring clear separation of concerns.
- **Kubernetes-Native Approach**: Services are deployed within the same namespace for easier discovery. Application and Dashboard use LoadBalancer, while Prometheus and Loki are internal (ClusterIP).
- **Single Namespace**: I've Kept everything inside a single namespace for simple service discovery.
- **Structured Logging**: JSON-based logs with request tracing for better querying.
- **Storage Strategy**: PersistentVolumes ensure log, metrics and dashboards are saved across restarts and rebuilds.
- **Probing**: Liveness & Readiness Probes ensure unhealthy pods restart automatically.

1.2 Tradeoffs

- **Complexity vs. Capability**: A multi-component setup introduces operational overhead but provides a **scalable and automated** observability stack.
- **Storage vs. Retention**: Prioritizing high-value logs optimizes storage but may limit historical analysis.
- **Kustomize instead of Helm**: Instead of Helm charts, I used Kustomize to manage raw Kubernetes manifests. This gives fine-grained control without the extra abstraction of Helm. However Requires manual config updates, but is more flexible.

2 Proof of Solution

This architecture **automates monitoring and debugging**, addressing slow and inconsistent troubleshooting.

- **Automated Monitoring**: Prometheus scrapes API metrics, Loki collects logs, and Grafana unifies the data into a single dashboard.
- **Performance Tracking**: Monitors response time, error rates, Go Routines, memory usage per container, and network usage in terms of response times etc.

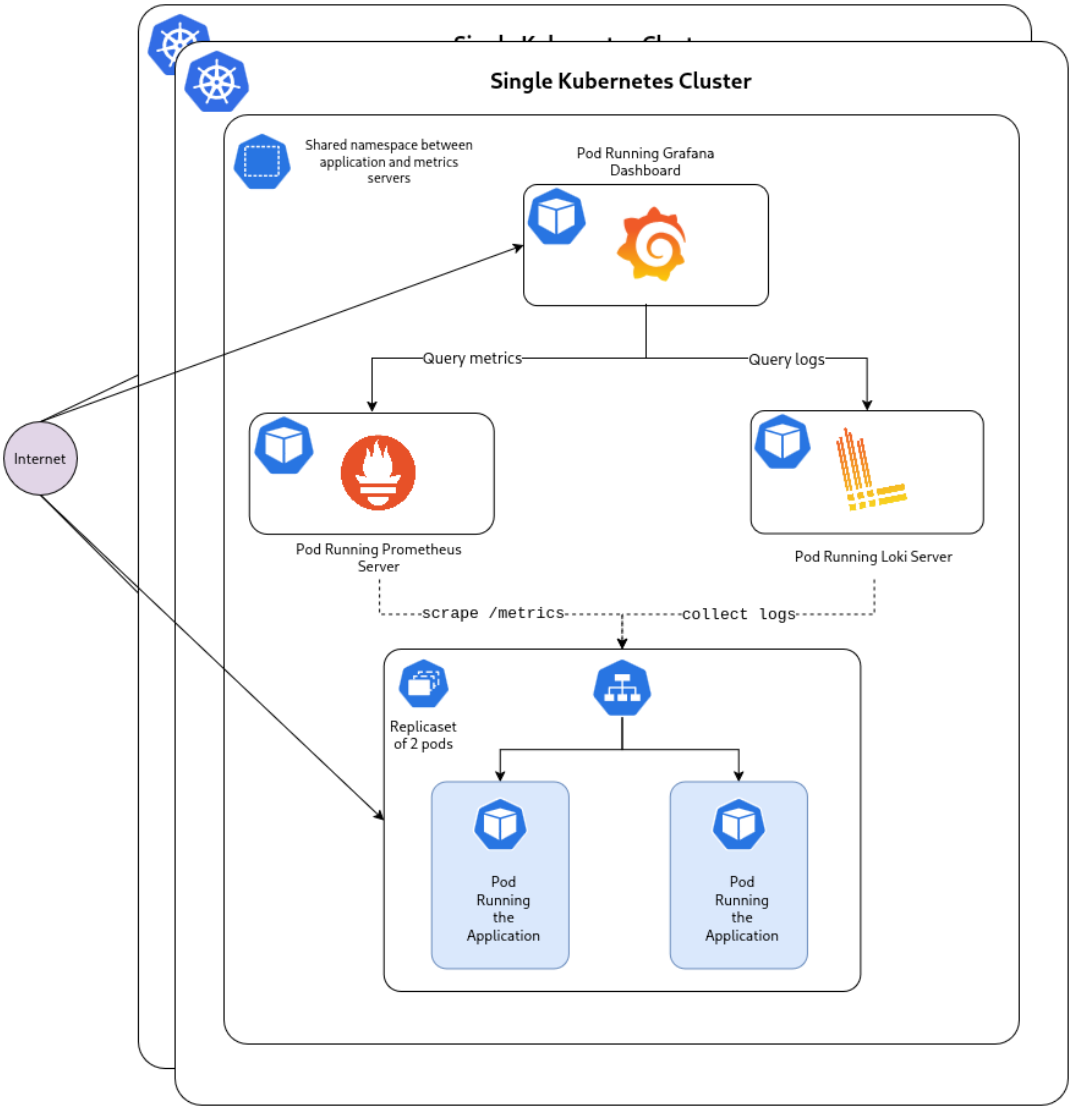

Figure 1: High-Level Design of Kubernetes Observability Architecture

3 Known Gaps and Justification

- **Limited Kubernetes Infrastructure Visibility**: Focuses on **application-level observability**; node-level monitoring is not a priority. Reason being Application has become a bottleneck, priority wise it should be handled first.
- **Historical Trend Analysis**: Limited retention(7d) since the focus is operational troubleshooting, However this is configurable and can be changed to more days.

4 Future Enhancements

- **Setup Alerting Rules**
- **Prebuilt Dashboard Templates**
- **Improved Query Performance**

5 Conclusion

This architecture **reduces MTTR**, enhances visibility, and **centralizes observability** for Kubernetes applications. By leveraging **Grafana, Prometheus, and Loki**, engineering teams gain an automated and scalable monitoring solution that minimizes manual debugging efforts.