# **As-Built Documentation: AKS Observability Stack (eddv3-hbt)**

## **1. Infrastructure Overview**

The observability stack is deployed into the **eddv3-hbt** namespace of an Azure Kubernetes Service (AKS) cluster. The environment exposes its services through a **Nginx Ingress** controller and is fronted by **Azure API Management (APIM)**. Secrets such as Azure Monitor credentials are stored in **Azure Key Vault** and injected into the pods via Kubernetes secrets. Azure Pipelines orchestrate the Helm deployments (pipeline YAMLs and Key Vault references are pending). The cluster uses a TLS secret (tls-secret) to terminate HTTPS for the domain, and DNS records point the public FQDN at the ingress controller.

## **2. Deployment Automation**

Helm charts are used for each service:

* **Grafana** – deployed from grafana/grafana chart. Custom values override persistence, datasource configuration, dashboards, Azure AD authentication and secret injection. The chart labels show that version **9.2.10** is used for the Helm release (see helm.sh/chart: grafana-9.2.10 in pod labels). The image repository is grafana/grafana. Deployment settings include one replica, ClusterIP service on port 80 and a separate service account (grafana-sa) backed by a managed identity.
* **Prometheus & kube-prometheus-stack** – used to provide metrics scraping and alerting. The provided values file enables additional scrape jobs for the OpenTelemetry collector and adds a ServiceMonitor for the collector. Custom resource definitions (CRDs) are installed.
* **Loki** – deployed in **simple-scalable** mode with one backend and read/write components. MinIO is enabled for object storage and the retention period is set to 31 days (744h).
* **Jaeger** – deployed mostly disabled with only the **all-in-one** component running in memory storage. This provides a lightweight trace UI for development.
* **OpenTelemetry Operator & Collector** – the operator is installed with RBAC roles and bindings. A custom OpenTelemetryCollector CRD defines a deployment-mode collector running two replicas with OTLP, Jaeger and Loki receivers, multiple processors (batching, memory limiter, resource enrichment, transforms and metric transforms) and exporters to Loki (otlphttp), Prometheus (prometheus exporter) and Jaeger (otlp/jaeger). A separate Helm chart is available for a generic opentelemetry-collector deployment (values commented for dev/testing). These components enable unified collection of logs, metrics and traces.

Deployments are executed via Azure Pipelines. Pipeline stages typically perform the following tasks (pipeline YAMLs to be supplied):

* Set up Azure CLI and Helm.
* Retrieve secrets from Key Vault and create Kubernetes secrets.
* Deploy or upgrade the Helm releases for each component.
* Validate that services are healthy via kubectl or helm test suites.

## **3. Service-Specific Configuration**

### **3.1 Grafana**

* **Chart & Image** – uses grafana/grafana image (tag inherited from chart appVersion) and Helm chart **v9.2.10**. It deploys a single replica with persistent storage (10 Gi PVC) and configures readiness/liveness probes on /api/health.
* **Namespace & Service** – runs in eddv3-hbt namespace. A ClusterIP service named grafana exposes port 80, targeting container port 3000.
* **Ingress & DNS** – an Nginx Ingress named grafana-ingress exposes Grafana at host **eddv3-hbt** and path **/grafana**. The ingress uses annotations to rewrite the path (nginx.ingress.kubernetes.io/rewrite-target: /$1) and disables forced HTTPS redirect. TLS is terminated via tls-secret. A commented alternative path pattern (/|$)(.\*) indicates earlier experimentation to support deeper rewrites. An optional ConfigMap (grafana-headers) can supply custom X-Forwarded-\* headers, but it is commented out.
* **Configuration (grafana.ini)** – the values file sets root\_url = https://eddv3-hbt/grafana and serve\_from\_sub\_path = true. This instructs Grafana to serve all assets and redirect responses from the /grafana base path – a documented pattern for running behind a reverse proxy or subpath[grafana.com](https://grafana.com/tutorials/run-grafana-behind-a-proxy/#:~:text=If%20you%20don%E2%80%99t%20want%20or,true). Grafana logs and data paths are standard (/var/log/grafana, /var/lib/grafana). analytics.check\_for\_updates is enabled and log level is set to debug.
* **Authentication** – Azure AD OAuth is configured (auth.azuread.enabled: true) using client ID, client secret and tenant ID variables. These values are injected via environment variables referencing an azure-monitors secret. Grafana admin credentials can be supplied via a secret (admin user/password) or using the adminUser: admin defaults.
* **Datasources** – multiple datasources are provisioned via datasources.yaml:  
  + **Azure Monitor** – uses the grafana-azure-monitor-datasource plugin with tenant ID, client ID and client secret (values pulled from Key Vault) and references an Azure Log Analytics workspace. The plugin collects metrics and logs from AKS.
  + **Loki** – configured to read logs from loki-read.eddv3-hbt.svc.cluster.local:3100.
  + **Jaeger** – points to jaeger-query.eddv3-hbt.svc.cluster.local:16686 for trace queries.
  + **Prometheus & Alertmanager** – connect to the kube-prometheus stack (kube-prometheus-stack-prometheus.eddv3-hbt.svc.cluster.local:9090 and alertmanager on port 9093).
* **Dashboards** – a dashboard named **“AKS Node & Pod Health”** is embedded directly into the values file. It uses Azure Monitor queries to count ready nodes, running pods and CrashLoopBackOff pods in the Log Analytics workspace. This dashboard is placed in the “AKS eddv3-hbt” folder.
* **Plugins & Sidecars** – the grafana-azure-monitor-datasource plugin is installed. The sidecar settings for dashboards, alerts, datasources and notifiers are present but disabled.
* **Persistence & Security** – persistence uses an RWO PVC of 10 Gi. The pod runs as user and group 472 with seccomp profiles and no privilege escalation. A service account (grafana-sa) is specified rather than auto-created, enabling integration with a managed identity. Secrets such as Azure AD credentials and Grafana admin password should be stored in Key Vault and referenced via Kubernetes secrets.

### **3.2 Prometheus**

* **Chart** – part of the kube-prometheus-stack (CRDs installed). The values file defines an additionalScrapeConfigs job called otel-collector scraping the OTEL collector metrics at otelcollector-collector.eddv3-hbt.svc.cluster.local:8889 every 10 seconds. An additional ServiceMonitor monitors the OpenTelemetry collector pods in the eddv3-hbt namespace. Prometheus is integrated with Alertmanager.

### **3.3 Loki**

* **Chart & Deployment Mode** – deployed in SimpleScalable mode. One backend replica handles storage, with one read and two write replicas. MinIO is enabled for object storage and allow\_structured\_metadata: true. The retention period is **31 days** (744h). Most unused single-binary components are explicitly set to 0 replicas to save resources. A gateway with OTLP receivers is enabled to accept OTLP logs on gRPC/HTTP ports 4317/4318.

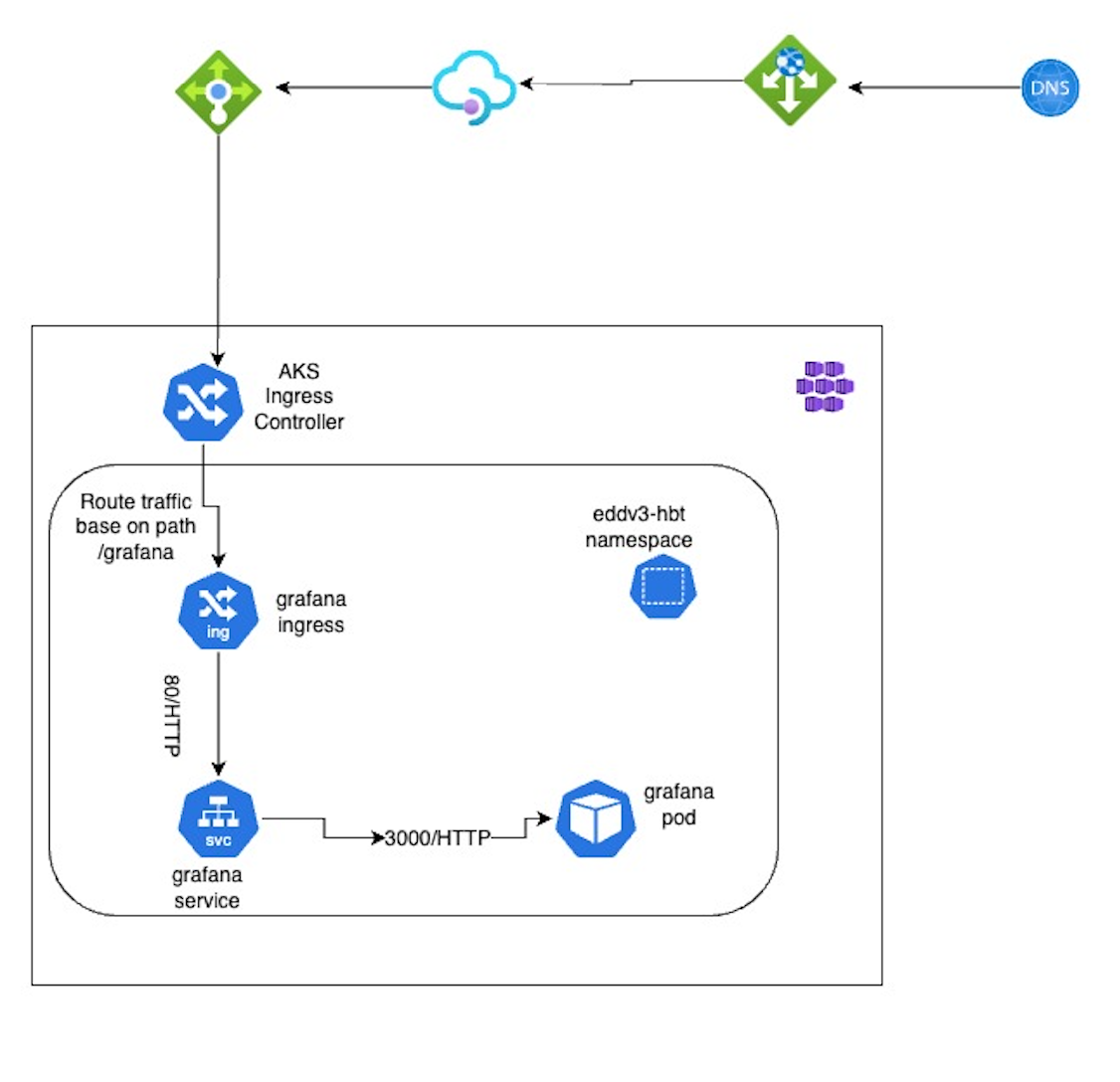
### **3.4 Jaeger**

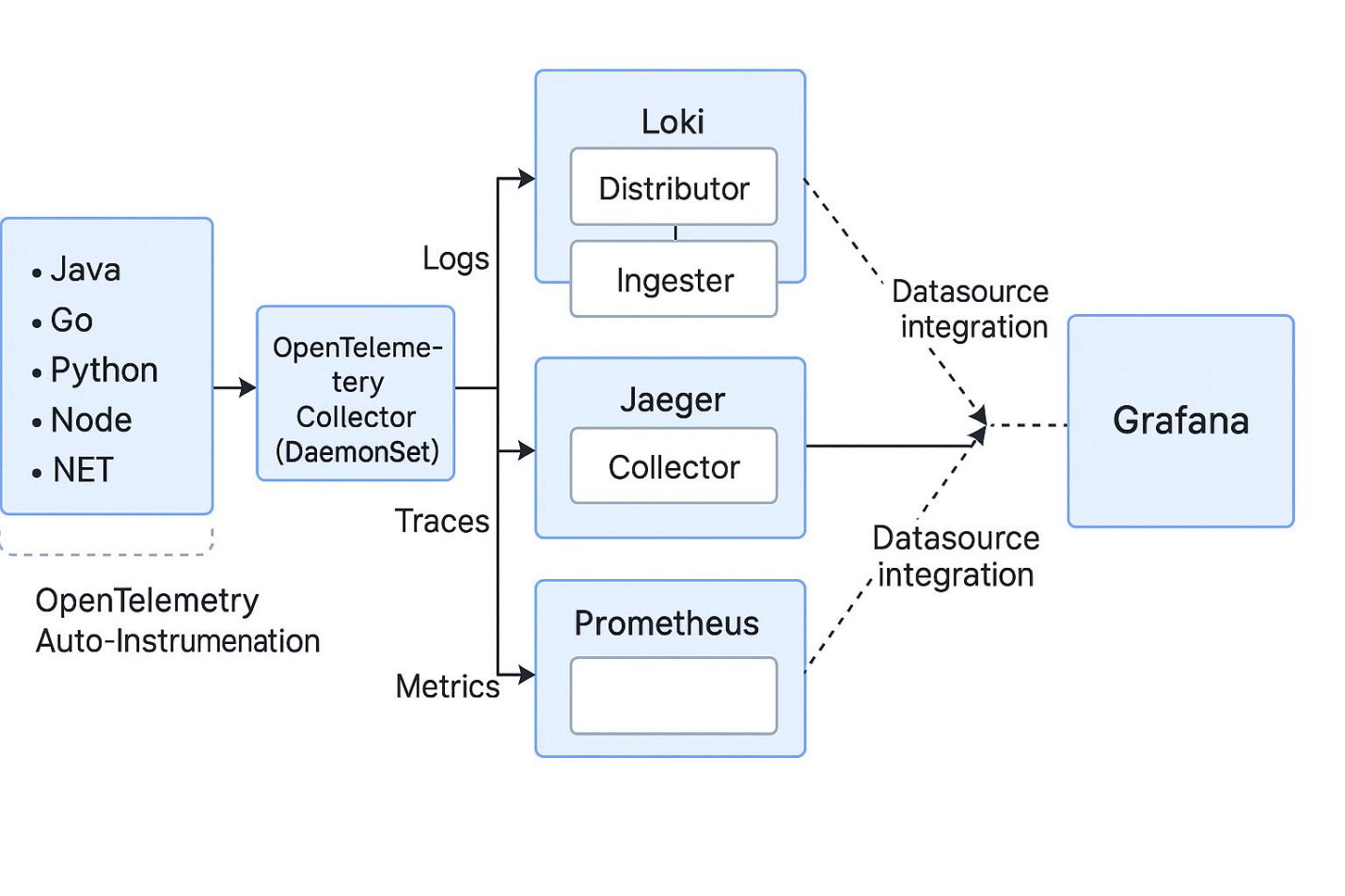
* **Chart** – the Jaeger Helm release is configured with allInOne mode **enabled** (memory storage). Other components (cassandra, elasticsearch, kafka) are disabled. In this mode the collector and query endpoints run inside a single pod, but the provided values appear to disable query and collector services (only allInOne would run). The environment uses Jaeger only for local development; production would require a distributed deployment and external storage.

### **3.5 OpenTelemetry Operator & Collector**

* **Operator** – A ServiceAccount (opentelemetry-operator), ClusterRole and ClusterRoleBinding grant permissions to watch pods, namespaces, nodes and custom resources (servicemonitors, podmonitors, pod disruption budgets). An additional ClusterRole (otel-operator-extra) and its binding provide access to OpenTelemetry custom resources. The operator uses image otel/opentelemetry-operator:v0.127.0. RBAC creation is enabled.
* **Collector (CRD)** – an OpenTelemetryCollector named **otelcollector** runs in **deployment** mode with two replicas. It uses image otel/opentelemetry-collector-contrib:0.128.0 and service account opentelemetry-collector. Several environment variables expose pod and node metadata.  
  + **Receivers** – OTLP (gRPC 4317 and HTTP 4318), Jaeger (gRPC 14250, HTTP 14268, UDP 6831), and Loki (HTTP 3500, gRPC 3600).
  + **Processors** – include batch, memory\_limiter, groupbytrace, resource (enriching with k8s and host attributes), resourcedetection, transform (enrich logs with metadata and trace IDs) and metricstransform (add labels to CPU and memory metrics).
  + **Exporters** – debug (basic logging), otlphttp sending logs to loki-write.eddv3-hbt.svc.cluster.local:3100/otlp, prometheus exporting metrics on port 8889 with cluster labels, and otlp/jaeger exporting traces to jaeger-collector.eddv3-hbt.svc.cluster.local:4317.
  + **Pipelines** – defined for **metrics** (OTLP → memory limiter → resource enrichment → metric transform → batch → debug + prometheus exporters); **traces** (OTLP & Jaeger receivers → memory limiter → resource enrichment → group by trace → batch → debug + otlp/jaeger exporters); **logs** (OTLP & Loki receivers → resource enrichment → transform → memory limiter → batch → debug + otlphttp exporters).
  + **Health & Monitoring** – health extension exposes /health on port 13133. A Prometheus exporter is exposed on port 8889 (and scraped by Prometheus). Liveness and readiness probes check port 13133.
* An alternative Helm values file (otel-values-dev) is included but commented; it shows similar configuration for the generic opentelemetry-collector chart with autoscaling enabled.

## **4. Grafana External Access & DNS Routing**





Grafana is accessible internally at http://grafana:80 but fails when routed through APIM under /\_cluster\_ingress\_path\_/grafana because APIM strips the /\_cluster\_ingress\_path\_ prefix path. When Grafana’s root\_url is configured with a sub-path (for example, root\_url = https://example.com/grafana) and serve\_from\_sub\_path = true, the application expects all requests and redirects to include that prefix [grafana.com](https://grafana.com/tutorials/run-grafana-behind-a-proxy/#:~:text=If%20you%20don%E2%80%99t%20want%20or,true). If an upstream proxy rewrites or drops part of the path, Grafana generates incorrect redirects or 404 errors a behaviour change that has been noted in Grafana v10 and later [github.com](https://github.com/grafana/grafana/issues/72577#:~:text=What%20happened%3F).

**Root cause:** APIM forwards requests from /\_cluster\_ingress\_path\_/grafana to the ingress at /grafana, stripping /\_cluster\_ingress\_path\_. Grafana, configured to serve from /\_cluster\_ingress\_path\_/grafana, sees a mismatch between the incoming request path and its root\_url, resulting in broken links and 404 responses.

**Options considered**

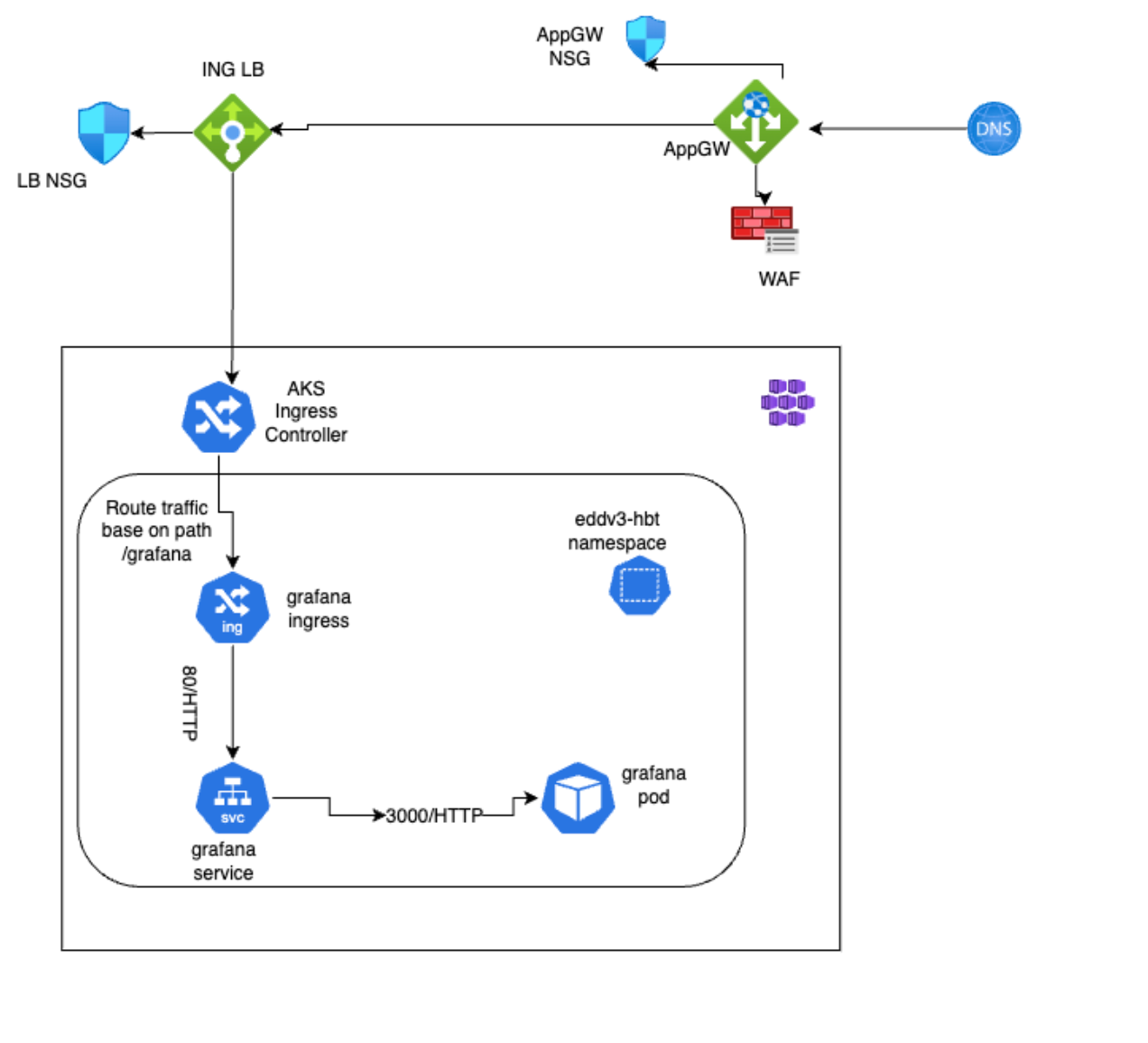
1. **Preserve path via APIM**: Feasible because APIM cannot add prefixes to proxied paths.
2. **Use a subdomain**: If permitted in this environment.
3. **Ingress path rewrite**: The current ingress uses rewrite-target to strip the /grafana prefix. This partially works but does not solve the /\_cluster\_ingress\_path\_ portion removed by APIM.

**NGINX sidecar proxy**  – Deploy a sidecar container alongside Grafana that listens on a path with /\_cluster\_ingress\_path\_/grafana, rewrites requests to /grafana internally and adjusts redirects. The Grafana configuration should set:  
  
 grafana.ini  
[server]

root\_url = https://apim.company.com/\_cluster\_ingress\_path\_/grafana

serve\_from\_sub\_path = true

1. The sidecar can use an Nginx configuration to rewrite /\_cluster\_ingress\_path\_/grafana/(.\*) to /grafana/$1 and preserve headers. This ensures that Grafana sees the expected path and generates correct links. A similar approach is documented in Grafana’s reverse proxy guide, which states that when serving from a sub-path you must set serve\_from\_sub\_path to true and include the sub-path in root\_url [grafana.com](https://grafana.com/tutorials/run-grafana-behind-a-proxy/#:~:text=If%20you%20don%E2%80%99t%20want%20or,true).
2. **Alternative Approach [Bypass APIM]** – In some environments it may be preferable to remove APIM from the data path entirely. You can map the external DNS record directly to an **Azure Application Gateway** (App GW) configured with **Web Application Firewall (WAF)** for Layer‑7 protection. App GW performs SSL termination and path‑based routing and can rewrite paths when forwarding traffic to the AKS load balancer. The flow would be:



**- DNS → App Gateway**

**- App Gateway → AKS Load Balancer**

**- Load Balancer → Ingress → Service**

This design eliminates APIM’s path stripping while still providing edge security via WAF. However, you lose APIM’s API management features (rate limiting, API keys, developer portal). If those features are not required for Grafana, routing through App GW can simplify the architecture and reduce latency. Ensure that the App GW WAF policy allows Grafana’s WebSockets and long‑running HTTP connections.

## **5. Ingress & DNS Configuration**

* **Grafana Ingress** – as described above, host **eddv3-hbt** with path /grafana. Uses Nginx annotations for regex rewrites (use-regex: "true") and path rewrite to /$1. TLS secret tls-secret holds the certificate. The commented ConfigMap grafana-headers shows recommended X-Forwarded-\* headers if required.
* **Other services** – The kube-prometheus-stack, Loki and Jaeger services are accessed internally within the cluster (ClusterIP). Jaeger all-in-one, Prometheus and Alertmanager do not expose external ingresses by default.
* **DNS** – A DNS A record should point eddv3-hbt (or the APIM-managed domain) to the public IP of the ingress controller/APIM. CNAME records can be used for sub-domains if allowed. TLS must be configured to match the domain names.

## **6. Security & Secrets**

* **Azure Key Vault** – secrets such as Azure AD client ID/secret, tenant ID, Grafana admin credentials and Azure Monitor workspace identifiers are stored in Key Vault. Azure Pipelines retrieve these secrets and create Kubernetes secrets (e.g., azure-monitors). The Grafana Helm values reference these secrets via environment variables (envFromSecret: "azure-monitors") and placeholders like $TENANTID, $CLIENTID and $CLIENTSECRET.
* **Service Accounts & RBAC** – Each component uses dedicated service accounts. Grafana references an existing service account grafana-sa, enabling pod-managed identity binding. The OpenTelemetry operator and collector have their own service accounts with cluster-wide roles and bindings to access pods, nodes, namespaces, custom resources and RBAC to create ServiceMonitors. Minimal privileges are granted for Jaeger, Prometheus and Loki as they run in the same namespace.
* **Pod Security Contexts** – Containers run as non-root where possible (Grafana runs as UID/GID 472). Capabilities are dropped (capabilities.drop: [ALL]), and seccomp profiles are set to RuntimeDefault. allowPrivilegeEscalation is false for sidecars. Liveness and readiness probes ensure containers restart if misbehaving. Network policies are disabled in the provided values, so pods can communicate freely; consider enabling ingress/egress restrictions for production.

## **7. Observability Strategy**

The observability stack collects metrics, logs and traces across the AKS cluster:

* **Metrics** – Prometheus (kube-prometheus-stack) scrapes Kubernetes components (kubelet, cAdvisor, API server) and the OpenTelemetry collector via additionalScrapeConfigs. Grafana queries Prometheus for cluster metrics and visualizes them via dashboards.
* **Logs** – Applications emit logs that are forwarded to the OpenTelemetry collector. The collector’s Loki receiver ingests logs and exports them via the otlphttp exporter to Loki (write path). Loki stores logs in MinIO and replicates data for high availability. Grafana reads logs from Loki using the loki datasource.
* **Traces** – Services instrumented with OpenTelemetry SDK send spans to the collector via OTLP (gRPC/HTTP) or Jaeger protocols. The collector processes and exports traces to Jaeger’s collector service (otlp/jaeger). Grafana queries traces from the Jaeger backend using the jaeger datasource. The environment uses a lightweight Jaeger all-in-one deployment for development; production should use a distributed setup.
* **Dashboards & Alerts** – Grafana provides dashboards such as the AKS Node & Pod Health dashboard. Alertmanager is available as part of kube-prometheus-stack but alert rules are not defined in the provided configuration. You can create PrometheusRule CRDs to configure alerts.

## **8. Known Issues & Recommendations**

1. **Grafana External Routing (APIM Path Stripping)** – As discussed in Section 4, APIM removes /\_cluster\_ingress\_path\_ from the request path, causing Grafana’s redirects to break. Implement an Nginx sidecar proxy that rewrites the request and set root\_url and serve\_from\_sub\_path accordingly [grafana.com](https://grafana.com/tutorials/run-grafana-behind-a-proxy/#:~:text=If%20you%20don%E2%80%99t%20want%20or,true).
2. **Ingress Path Rewrite** – The current ingress uses a rewrite target of /$1 with use-regex: true, but the path specification uses path: /grafana rather than a regex. Consider updating the path to /grafana(/|$)(.\*) and adjusting the rewrite-target to /$2 (or rely on the sidecar) to correctly strip the prefix when using sub-paths.
3. **Secret Management** – Ensure that all sensitive values (client IDs/secrets, admin passwords) are stored in Azure Key Vault and not embedded directly in values files. Use Azure Pipelines to fetch secrets and create Kubernetes secrets at deployment time.
4. **Resource Limits & Autoscaling** – The OpenTelemetry collector runs with static resource requests/limits. Production clusters might require autoscaling based on load; consider enabling the autoscaling section in the Helm chart and tune CPU/memory requests.
5. **Network Policies** – Network policies are disabled (networkPolicy.enabled: false). For secure environments, enable them to restrict pod communication to required ports and services.
6. **Jaeger Storage** – The Jaeger configuration uses in-memory storage (allInOne mode). For persistent traces, configure an external storage backend (e.g., Elasticsearch or Cassandra) and disable allInOne in favour of separate collector, query and storage components.