Documentation for Kubernetes-Based Microservices Deployment

The goal was to deploy a set of microservices in the Google Cloud environment using Kubernetes. The services (Collector, Image Analysis, Face Recognition, Section, and Alert service) were containerized and then orchestrated on Google Kubernetes Engine (GKE).

# Communication/ Data Flow:

The **Collector service** receives **frame data** from various camera services, including timestamps, image data, section IDs, events, and unique frame UUIDs. After analyzing the data, it sends **person detection results** to the **Section service** for statistics and logging and forwards **alerts** on detected persons to the **Alert service.**

**Communication Method:** The communication between services is **synchronous** - each service must respond before the Collector proceeds. This synchronous setup allows real-time processing but could introduce delays under heavy load. For future improvement, an **asynchronous message queue** (e.g., RabbitMQ, Kafka) could help balance loads more effectively, enabling Collector to send data asynchronously to Alert and Section services.

**Message Delivery:** Messages sent from camera services successfully reach **Alert**

and **Section** services. (Logs and alerts validated this data flow)

**Service Visibility:** Services can locate each other using **Kubernetes DNS -** each service, deployed with a **ClusterIP**, has a consistent name that Kubernetes DNS resolves within the cluster (e.g., collector-service, alert-service). This setup allows services to communicate directly using service names as URLs, while keeping traffic internal to the cluster.

# Scalability and Bottlenecks:

The application is scalable, as Kubernetes enables autoscaling based on Pod replicas and resource utilization. Scaling focuses on Collector, Image Analysis, and

Face Recognition services, which benefit the most from it due to high data-processing demands. Potential bottlenecks include **image data processing** in Collector and **frame analysis** in Image Analysis services. Autoscaling helps address these, but a distributed architecture or partitioned processing may further improve scaling.

**Increased Camera Services:** Adding more camera services increases data flow to Collector. The script supports multiple streams by processing each frame independently, but scaling Collector and analysis services helps manage high frame rates (e.g., with 4 cameras sending 100 frames). The system can process more images by increasing the replicas of resource-intensive services like Image Analysis and Face Recognition to distribute the load effectively.

**Scaling Strategy:** Using Kubernetes HorizontalPodAutoscaler, autoscaling was set based on CPU/memory thresholds for Collector, Image Analysis, and Face Recognition services. Cloud Run could provide a serverless option for **Alert service** for efficient event handling without pre-allocating resources.

# Configuration of GKE Cluster:

The GKE cluster configuration included a single-zone setup with standard node pools, which allows basic testing of scalability and performance. Node autoscaling was enabled to dynamically adjust the available resources.

# Kubernetes Objects Used:

Deployments (for each microservice), Services ( ClusterIP services were used for internal communication between services, whereas LoadBalancer services exposed Collector and Grafana for external access), Pods(unit running each service container).

# Resource Constraints:

The most computationally intensive services were Image Analysis and Face Recognition. Resource limits were set to ensure these services received sufficient memory and CPU to handle image processing.

# Ingress Configuration:

Ingress was configured to forward HTTP traffic to Collector, what allows external access without exposing multiple LoadBalancer services. This reduces costs and maintains a single entry point into the cluster.

# Cost Analysis:

Based on GKE’s pricing, provisioning this cluster (assuming it runs 24/7) could cost approximately **$500–$1,000 per year** depending on usage and scaling settings.

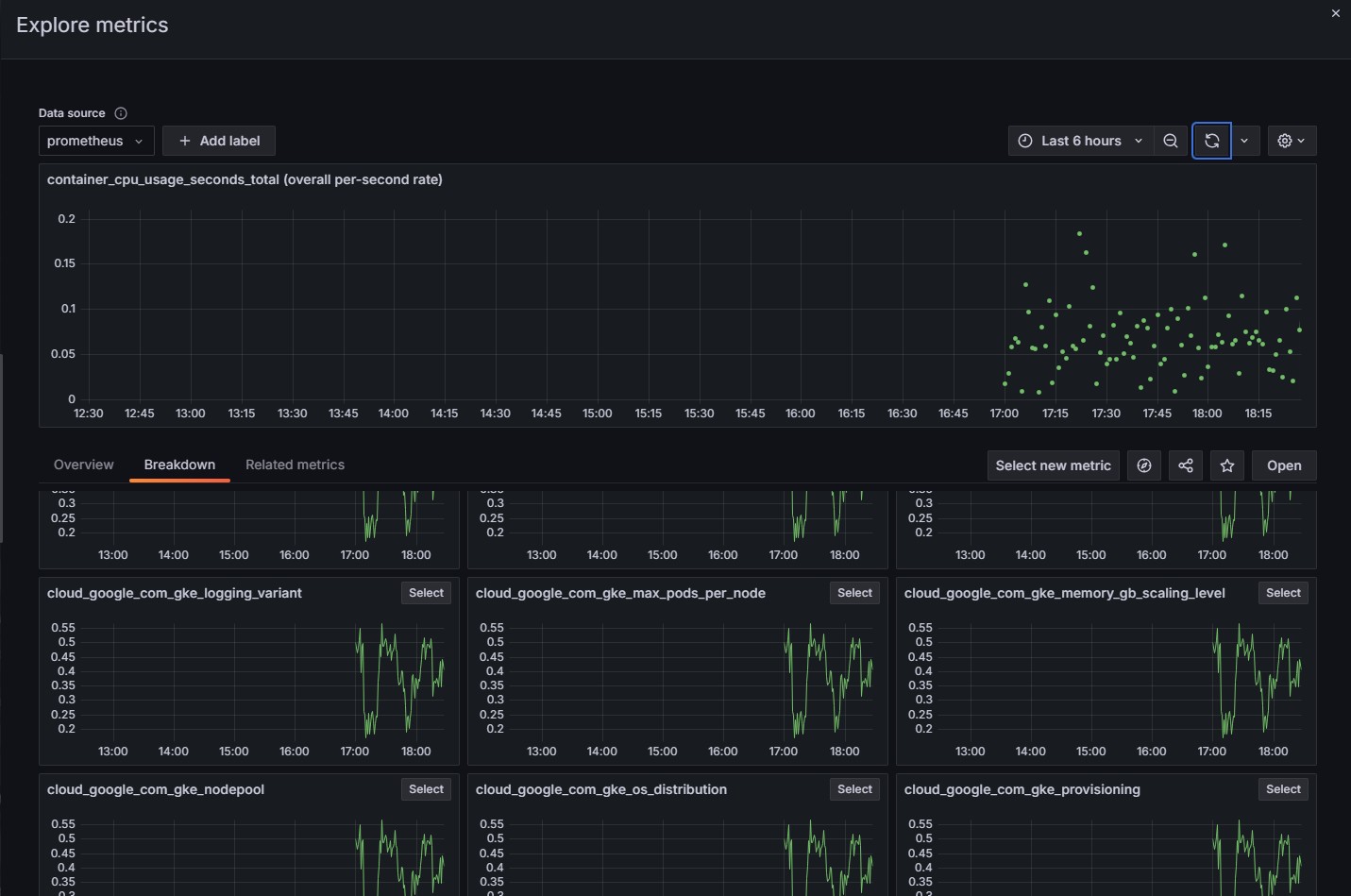
**GKE reduces upfront costs** and avoids hardware maintenance. However, for high- use cases, dedicated hardware may become more cost-effective in the long term.

**Cloud Run** or **serverless** options could lower costs for low-usage applications by scaling to zero. In contrast, GKE provides a consistent environment for complex, interconnected services, making it ideal for high-scale scenarios.

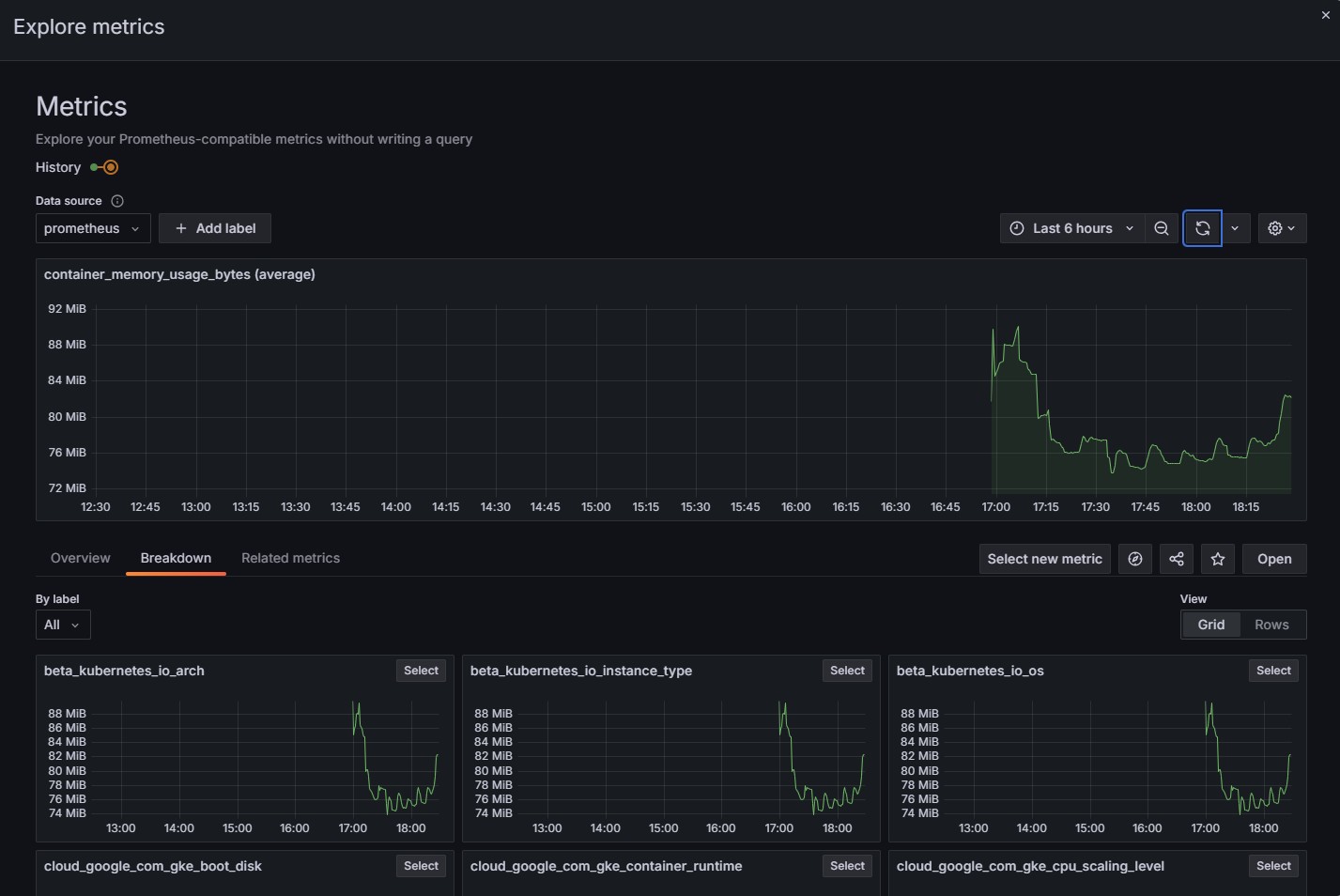
# Resource Usage:

**vCPU and Memory Usage**: Initial estimates matched actual usage, with Image Analysis and Face Recognition consuming the most resources due to processing requirements. Grafana dashboards provided insights into CPU, memory, and network utilization, which confirms efficient load distribution among services.

- ***CPU Usage*** across all containers, which provides an overview of the compute resources utilized within the cluster:

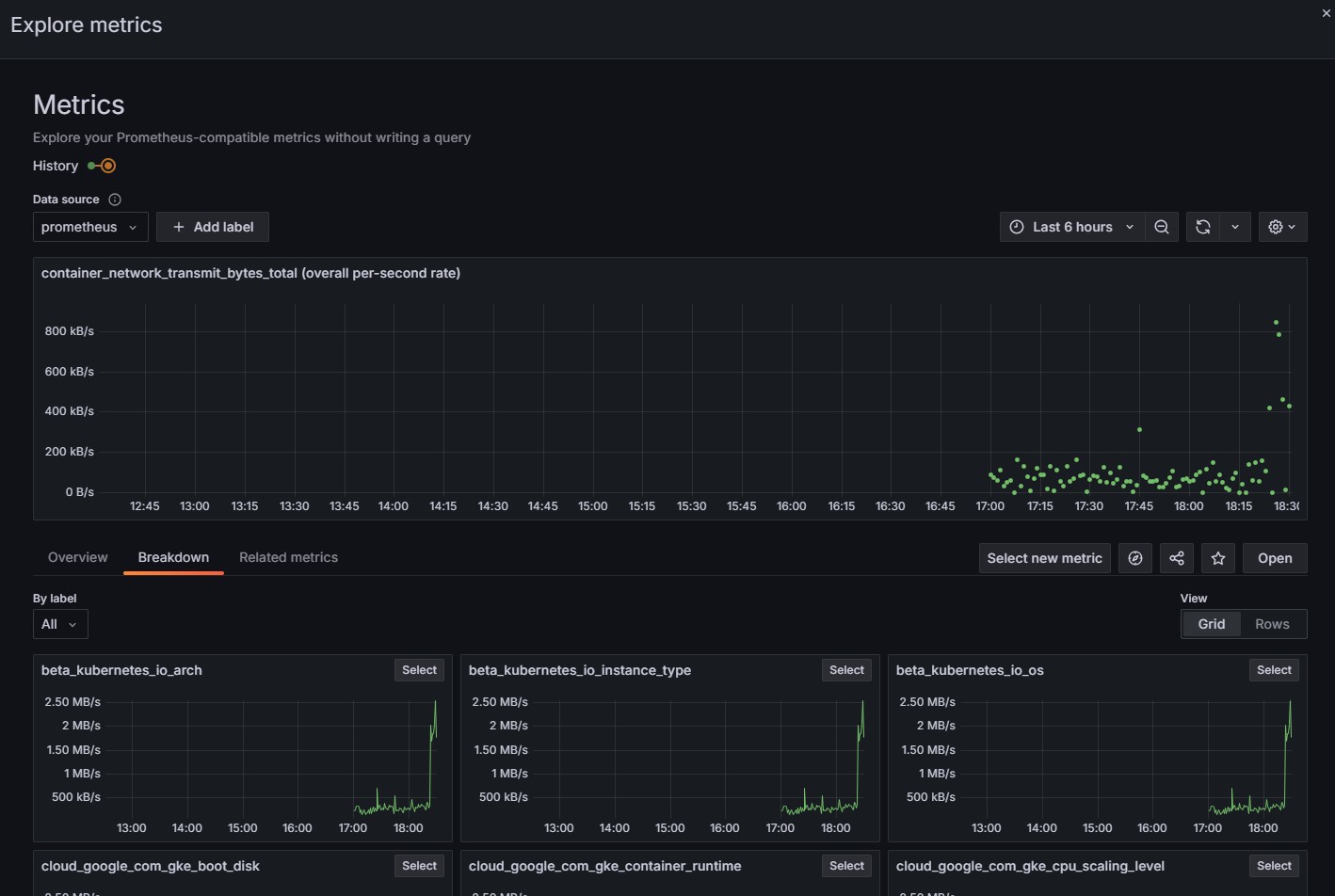


***-Memory Usage*** monitors the memory consumption across containers in the cluster, which provides an overview of resource utilization and identifies potential memory constraints:



***-Network traffic*** is divided into two primary metrics: transmitted bytes and received bytes. These metrics help monitor the data flow in and out of the cluster.

Network Transmit Traffic shows the rate of outgoing data from the containers:



Network Receive Traffic shows the rate of incoming data to the containers:

