Google Kubernetes Engine Use Case

Use Case: Scaling Modern Applications with Google Kubernetes Engine (GKE)

GENERAL CHARACTERISTICS

Intent	To deploy and scale containerized applications using Google Kubernetes Engine (GKE).
Scope	Implementation of a Kubernetes-based infrastructure to manage microservices and distributed systems.
Level	System-level.
Client	Confidential (Retail E-Commerce Platform).
Last Update	03/12/2024
Status	Finalized.
Stage	Deployment and Optimization.

ACTORS

Primary Actor	DevOps Engineer.
Secondary Actors	Application Developers, IT Operations Team, Security Team.

PREREQUISITES

Static Preconditions	- Google Cloud Project set up with Kubernetes Engine API enabled. - Containerized applications prepared for deployment.
Dynamic Preconditions	- Cluster configuration designed to handle application workloads. - Security policies and access controls defined for Kubernetes clusters.
Assumptions	- Client requires high scalability, reliability, and automated management of containerized applications. - Infrastructure must support microservices architecture.

TRIGGERS

Trigger Event	The client required a scalable solution to manage and deploy containerized applications across multiple regions.

EXPECTED OUTCOME

Success Postcondition	- Applications are deployed and scaled

	automatically based on traffic demands. - Cluster resources are optimized to minimize costs.
Failed Postcondition	- Applications experience downtime or performance issues during traffic spikes.

OPERATIONS AND CONCEPTS

Operations	1. Created a GKE cluster to host containerized applications. 2. Deployed application containers using Kubernetes manifests and Helm charts. 3. Configured auto-scaling policies for both pods and nodes. 4. Set up CI/CD pipelines for automated deployment using Cloud Build and GitHub Actions. 5. Enabled Kubernetes network policies to secure inter-service communication. 6. Monitored cluster performance using Cloud Monitoring and Prometheus.
Concepts	- Kubernetes: Orchestrates containerized applications for deployment, scaling, and management. - Auto-Scaling: Automatically adjusts the number of pods or nodes based on workload. - CI/CD Pipelines: Streamlines application deployment and updates.

MAIN SUCCESS SCENARIO

Step 1	Analyzed the client's application architecture and workload requirements.
Step 2	Configured a GKE cluster with appropriate node pools and resource limits.
Step 3	Deployed containerized microservices using Kubernetes manifests.
Step 4	Set up auto-scaling policies to handle variable traffic loads.
Step 5	Integrated CI/CD pipelines for seamless updates and deployments.
Step 6	Enabled network policies to enhance security within the cluster.
Step 7	Monitored performance and optimized cluster resources for cost efficiency.