DevOps case study

Distribusion Technologies

Nikolaos Lymperakis - DevOps Engineer

Scenario

You have been given the Vikunja ToDo List application, a microservices-based app that consists of three services: `frontend`, `backend`, and `database`. Your task is to automate the deployment of this application using Kubernetes, optimize the network configuration for performance and reliability, consider the deployment strategy for the database service, and optionally implement Identity and Access Management (IAM) using Keycloak for secure authentication and authorisation.

Requirements

1. Set up a Kubernetes cluster (you can use a local tool like Minikube or a cloud-based service like GKE, EKS, or AKS).
2. Consider using deployment templating tools like Helm, Kustomize, Jsonnet, or similar, for managing Kubernetes manifests and configurations. Explain your choice.
3. Create YAML manifests for deploying the three services (`frontend`, `backend`, `database`) in Kubernetes using your chosen deployment templating tool.
4. Consider whether the database service should be self-hosted or managed (e.g., self-hosted PostgreSQL vs. managed database service like AWS RDS or Google Cloud SQL). Justify your choice.
5. Optionally, implement Identity and Access Management (IAM) for secure authentication and authorization using Keycloak.
6. Ensure that the Vikunja ToDo List application is highly available and resilient to failures.
7. Implement and configure a load balancer to distribute traffic across the services.
8. Optimize the network settings for low latency and high throughput.
9. Use any necessary tools or configurations to monitor and debug the application.

Implementation

The following technologies and tools were used for the infrastructure provision and the deployment of **Vikunja ToDo List** application:

Cloud provider	GCP
Kubernetes cluster	GKE
Templating tool	Helm
Shared volume for application file storage	GCS Bucket
Database	mariadb
IaC configuration tool	Terraform
Monitoring and logging tools	Prometheus, Grafana, Loki
Certificate management	cert-manager controller
DNS provider	Cloudflare

The deployment scripts are available on Github repository at https://github.com/nlybe/vikunja-gke. It utilizes Terraform for infrastructure provisioning and a Helm chart for streamlined application and database deployment.
The production application is deployed at the following URL: [https://demo.lyberakis.gr](The production application is deployed at the following URL: http://demo.lyberakis.gr.

Vikunja application

Prior to version 0.23.0 (released February 10, 2024), the Vikunja API and frontend were maintained in separate repositories. To streamline deployment and hosting, these repositories, along with the desktop package, were consolidated into a single repository. This unified codebase now produces a single binary release artifact.
Consequently, the Vikunja application architecture has transitioned from a microservices-based system with multiple components to a simplified two-service model: Vikunja and a database. These services are deployed on Kubernetes using a Helm chart.

Deployment templating tool

Helm package manager was chosen because it is one of most popular tools for Kubernetes deployments, primarily due to its powerful templating capabilities and its ability to streamline the management of complex applications. Some of the key advantages of Helm are:

- Simplified Deployment

- Templating and Customization
- Version Control and Rollbacks
- Reusability
- Dependency Management

Database

Vikunja supports various databases, including PostgreSQL, MariaDB, MySQL, and SQLite. For this demonstration, MariaDB was chosen and deployed within the Kubernetes cluster as a self-hosted database.

In general a self-hosted database offers some advantages such:
- Provides complete administrative control, allowing for tailored configurations and security policies.
- It enables customization to precisely match the demonstration's requirements.
- For non-production environments like demos, the potential cost savings of self-hosting are a consideration.
- A self-hosted database is preferred where requirements are not strict.

On the other side, a managed database (e.g., Cloud SQL) offers important features:
- Simplified operational management, reducing administrative overhead.
- Tight integration with other cloud-based services.
- Automatic scalability to accommodate fluctuating workloads.
- Built-in backup and failover mechanisms for data protection and high availability.
- Robust user authentication and authorization through IAM integration.

These features are essential for production, staging, and pre-production environments where reliability and scalability are paramount."

In summary, a self-hosted database was chosen for this demonstration due to its cost-effectiveness and the less stringent requirements of a demonstration environment.

Vikunja application high availability

To ensure high availability for the Vikunja application, the following optimizations have been implemented:

- **Regional Redundancy:** Deployed a Kubernetes cluster across multiple availability zones within a region.
- **Elastic Scaling:** Enabled Kubernetes node pool autoscaling for horizontal scaling under high load.
- **Shared Storage:** Configured a regional Google Cloud Storage (GCS) bucket as a shared volume for Vikunja, allowing multiple instances to share files.

- **Instance Replication:** Increased Vikunja deployment replicas to run multiple instances concurrently.
- **Pod Autoscaling:** Implemented Horizontal Pod Autoscaler (HPA) to dynamically scale Vikunja pods based on load."
- **Health checks**: Implemented deployment health checks to ensure resilience against failures.
- **Resource usage**: Defined resource limits and requests for optimal resource allocation

Networking

To provide public internet access to the Vikunja application, an Ingress Controller was implemented. This decision leverages the Ingress Controller's advanced Layer 7 routing capabilities and cost-effectiveness for managing multiple services.

For optimal network performance, characterized by low latency and high throughput, it’s essential to deploy the Kubernetes cluster in a region that minimizes network distance to the majority of end users.

For this demonstration, network performance was optimized using the following configurations:

- **GKE Enhancements:**
    - Cloud DNS (cluster scope) and NodeLocal DNSCache were activated to minimize DNS resolution latency.
    - Dataplane V2 was enabled, providing significant performance gains in latency and throughput over Dataplane V1
- **Security and Efficiency**: Kubernetes Network Policies were deployed to control pod-to-pod communication, improving security posture and reducing extraneous network traffic.

To further optimize network performance, consider the following:

- Leverage Cilium Network Policies, powered by eBPF, for highly efficient and granular network policy enforcement.
- Implement Google Cloud Accelerated Networking for workloads requiring extremely low latency.
- Utilize Google Cloud's Premium Tier networking to benefit from Google's global, high-performance network and minimize latency.
- Ensure node pools are adequately resourced to handle anticipated traffic loads, preventing performance bottlenecks.
- Deploy Cloud CDN to cache and distribute static content from edge locations, reducing latency and improving throughput for end-users."

Monitoring and logging tools

Prometheus, Grafana, and Loki provide a robust monitoring and logging stack, ideal for Kubernetes environments.

- **Prometheus**: Collects time-series data from Kubernetes components and applications, leveraging Kubernetes service discovery for seamless monitoring of dynamic environments.
- **Grafana**: Visualizes Prometheus metrics through interactive dashboards, offering a user-friendly interface for analysis.
- **Loki**: Aggregates and efficiently stores logs from Kubernetes pods, providing a centralized logging solution.

This stack enables effective monitoring and troubleshooting of Kubernetes deployments, ensuring application reliability and performance. For this demonstration, Prometheus, Grafana, and Loki were deployed using their official Helm charts within the Terraform infrastructure code.

While the basic configuration and dashboards provide essential monitoring, enhancements such as alerting, Grafana SSO, and expanded dashboard coverage would further improve monitoring capabilities."