DevOps for Delivery Network

Introduction:

The client was in the business of solving the challenges of last mile delivery. Their aim was to develop a consumer facing parcel network comprising of top independent delivery service companies. They wanted leading-edge technology to form the backbone on their network. Their goal was to achieve cost-effective scale and exceptional first day delivery success rate. This case study showcases the management of various DevOps tasks for the solution that we built, including user account management, infrastructure automation, and deployment oversight. The project leveraged various tools and platforms such as Microsoft 365, G Suite, JumpCloud, Google Cloud Platform (GCP), Jenkins, and GitHub.

Client Details:

Name: Confidential | **Industry:** Logistics, Software | **Location:** USA

Technologies Used:

- **Cloud Platform**: GCP, Microsoft 365, G Suite
- **CI/CD, Source Code Management**: Jenkins, GitHub
- **Security & Identity Management**: JumpCloud, LDAP, SSO, IAM, Cloud Armor, SSL Certificates (Managed via GCP Certificate Manager), IAP (Identity Aware Proxy), Secret Manager
- **Infrastructure & Deployment**: GKE (Google Kubernetes Engine), Cloud SQL, Cloud Storage, Load Balancer, Cloud Functions, Pub/Sub, Artifact Registry, Cloud DNS
- **Networking & Secure Access:** OpenVPN, LDAP & SSO Integration (JumpCloud)
- **Monitoring & Logging:** Cloud Monitoring, Prometheus, Grafana, Google Cloud Logging

DevOps for Delivery Network

Project Description:

1. User Management

- **Microsoft 365** was used to add and remove users smoothly while keeping security and access controls in place.
- **G Suite accounts** were managed to give users access to Google Cloud Platform (GCP) resources.
- **JumpCloud** was utilized for directory services, simplifying user logins with LDAP and Single Sign-On (SSO).

2. Infrastructure Management

- **GCP Resources**: Managed various parts of Google Cloud Platform, including:
  - **Compute**: Virtual Machines (VMs), Kubernetes (GKE), Cloud Functions and Pub/Sub messaging.
  - **Storage**: Cloud SQL databases, Cloud Storage and backups (Snapshots).
  - **Security**: Managed user roles (IAM), Service Accounts, security tools (like Security Command Center and Cloud Armor) and SSL certificates.
  - **Monitoring**: Used tools like Cloud Monitoring, Prometheus and Grafana to keep systems healthy and performing well.

- **Database and Networking**:
  - Set up Cloud SQL for database needs.
  - Used OpenVPN for secure database access, with LDAP and SSO integration through JumpCloud.

3. CI/CD Pipelines and Deployment

- **Jenkins**: Created and managed Jenkinsfiles to automate deploying Java applications to Kubernetes (GKE).
- **Source Code Management**:
  - Used GitHub to manage and control the source code.
  - Deployed multiple microservices to GKE through Jenkins pipelines
- **Load Balancer**: Set up GCP Load Balancers with SSL certificates to securely and efficiently handle traffic.

DevOps for Delivery Network

4. Automation, Monitoring and Security

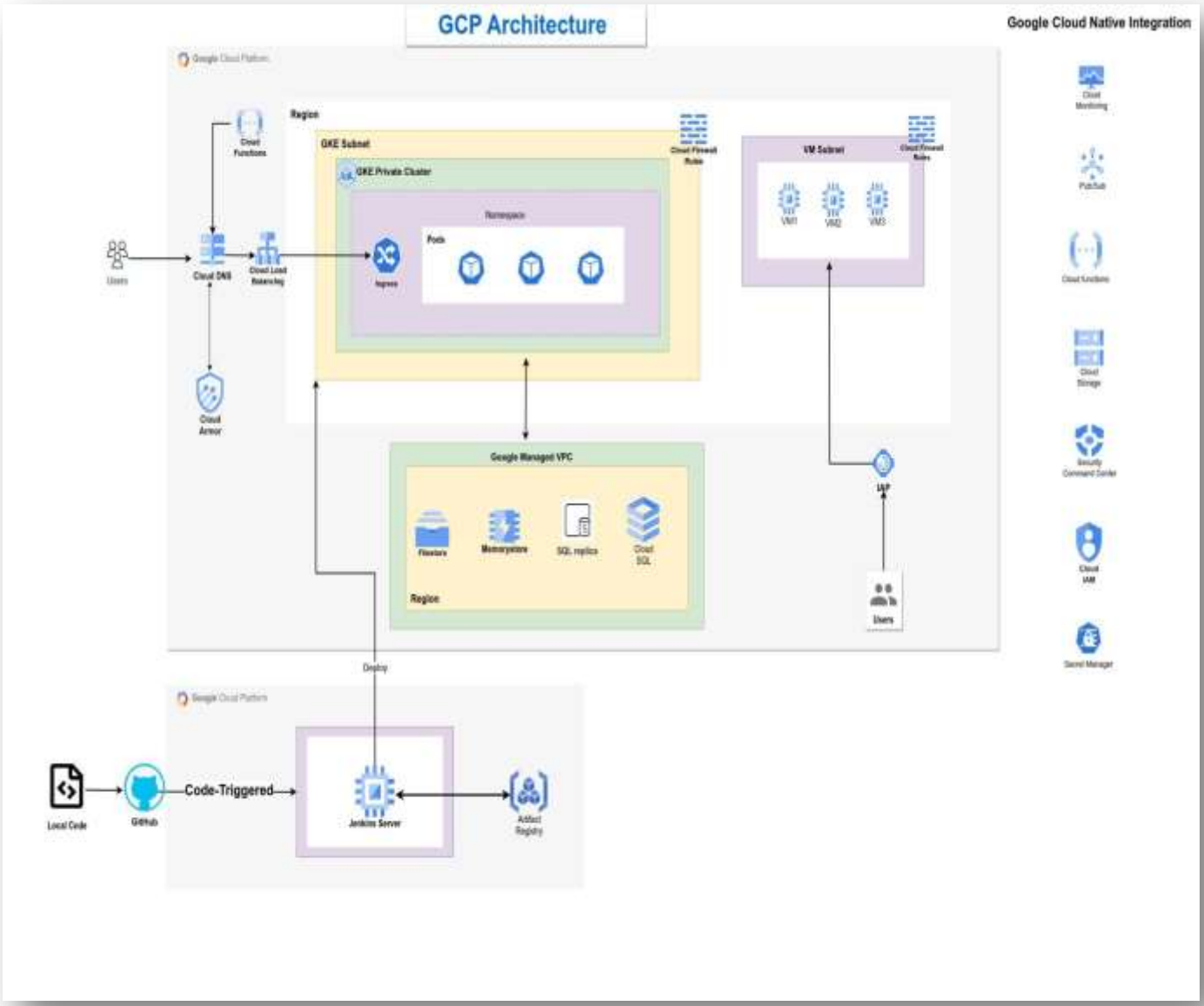
- **Automation**: Set up automated processes to reduce manual work and save time.
- **Security**: Improved security with IAM policies, Cloud Armor and centralized access controls using JumpCloud and GCP tools.
- **Monitoring**: Used Prometheus and Grafana to monitor system performance
- **Cost Optimization**: Managed GCP billing, created alerts and optimized resources to save money.

Outcomes and Impact:

- **Efficiency**: Made user onboarding and off boarding faster and smoother.
- **Scalability**: Simplified deploying microservices on GKE, helping the organization grow.
- **Security**: Improved security with LDAP, SSO and GCP's strong IAM tools.
- **Cost Optimization**: Kept costs under control with alerts and resource monitoring for better budget management.

DevOps for Delivery Network

GCP Architectural Diagram:



DevOps for Delivery Network

Problem Statements and Solutions:

Problem 1: High GCP Costs from Storing All Application Logs

Our application was configured to store all log levels (info, warning, error, debug, etc.) in Google Cloud Logging. This unrestricted logging caused an exponential increase in log storage, resulting in higher GCP costs.

Solution Implemented:

1. **Application Level Changes:**
    - Adjusted the logging configuration in the application to log only error level messages.
    - This reduced the volume of logs sent to GCP Logging, focusing on critical issues only.
2. **GCP Level Changes:**
    - Applied log filters in Google Cloud Logging to store only the logs with severity levels of *ERROR* or *CRITICAL*.
    - This further minimized storage use and ensured that unnecessary logs were excluded.
3. **Billing Alerts:**
    - Configured GCP Billing Alerts to monitor the costs associated with logging.
    - Alerts were set to notify the team when costs reached predefined thresholds, allowing proactive management.

Outcome:

- Log storage costs were reduced by approximately 60%, achieving significant cost savings.
- Improved monitoring and control over GCP billing related to logging.
- Enhanced focus on critical logs for faster troubleshooting.

DevOps for Delivery Network

Problem 2: Jenkins Disk Consumption Issue Due to Docker Image Building

Our Jenkins server was experiencing excessive disk usage due to the accumulation of old Docker images and unnecessary containers left in an "Exited" state. This led to performance degradation and frequent disk space warnings.

Solution Implemented:

1. **Cron Job for Docker Cleanup:**
    - Configured a cron job to automatically clean up old, unused Docker images on a regular schedule.
    - This ensured that disk space was regularly freed without manual intervention.
2. **Removing Unnecessary Containers:**
    - Implemented a cleanup process to remove containers in an Exited state, which were unnecessarily consuming disk space.

Outcome:

- Jenkins disk usage was significantly reduced, freeing up space for active builds and improving system performance.
- Automated cleanup ensured ongoing disk space optimization.

DevOps for Delivery Network

Problem 3: Pods Restarting Repeatedly and Newly Created Pods Not Starting on GKE Cluster

Pods in the GKE cluster were frequently restarting and newly created pods often failed to start. This instability affected application availability and performance.

Solution Implemented

1. **Resource Allocation Adjustments:**
    - Updated the memory and CPU resource limits in the pod YAML configuration files to better align with the application's requirements.
    - This prevented resource exhaustion and reduced pod restarts.
2. **Backend Configuration for Health Checks:**
    - Configured health checks in the BackendConfig to ensure pods met specific requirements before being marked as ready.
    - These health checks allowed the cluster to automatically manage pod readiness and availability.

Outcome:

- Improved pod stability with significantly fewer restarts.
- New pods consistently started successfully, enhancing overall application reliability.
- Enhanced resource management reduced operational overhead.

Lessons Learned:

1. **Proactive Monitoring:** Implementing monitoring and alert systems can prevent issues before they escalate.
2. **Selective Logging:** Logging only critical data is essential to balance operational insights and cost efficiency.
3. **Automation:** Automating cleanup processes ensures consistent application of policies and minimizes manual intervention.
4. **Resource Management:** Proper resource allocation and health checks are vital for ensuring system stability and reliability.

By addressing these challenges systematically, we achieved cost optimization, improved system reliability and enhanced operational efficiency.