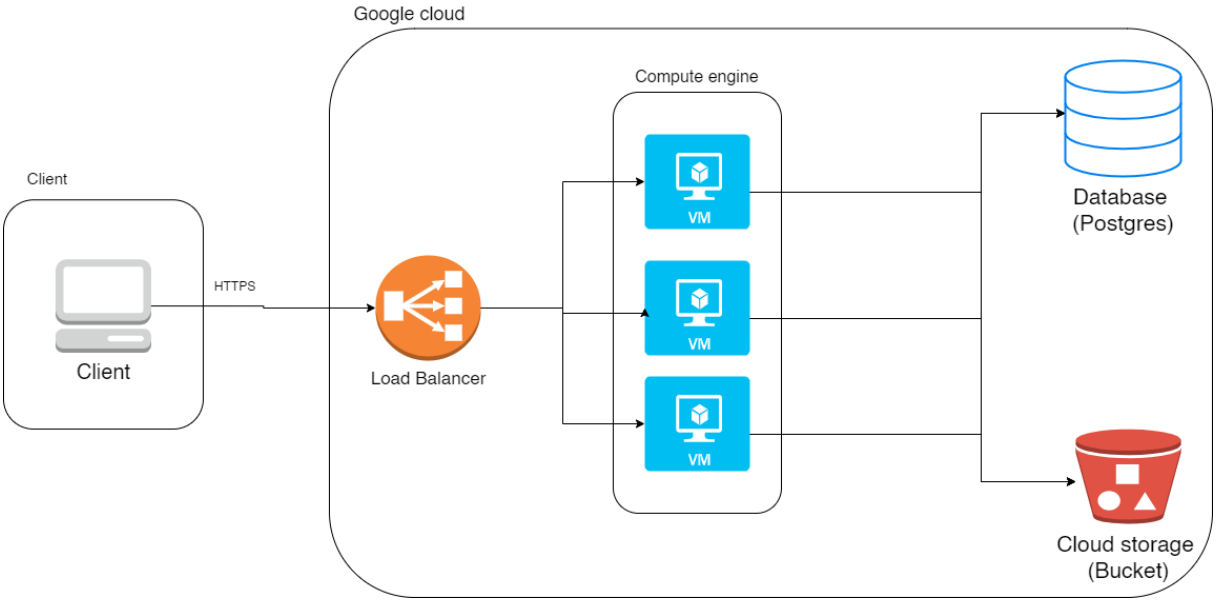
Deployment Analyzation & Architecture

By Kobe Wijnants

Kobe.wijnants@student.kdg.be

Inhoud

Deployment scheme



Connectivity

Connection to a private Gitlab repository

There are 2 ways to clone from a private Gitlab repository

Using ssh keys

1. Generate a ssh key:

```
ssh-keygen -t rsa -b 4096 -C your_email@example.com
```

2. Add ssh key to Gitlab: Settings > ssh keys and add the public key
3. Clone the repository:

```
git clone git@<gitlab_hostname>:<username>/<repository_name>.git
```

This can only be used to clone to a certain machine where the ssh key is made

Using PATs (Personal Access Tokens)

1. Generate Personal Access Token:

Log in to your GitLab account, go to "Settings" > "Access Tokens," and generate a new personal access token with the necessary scopes (e.g., read repository).

2. Clone Repository with Token:

Clone your private repository using the HTTPS URL and the generated token:

```
git clone
https://<username>:<personal_access_token>@<gitlab_hostname>/<username>/
<repository_name>.git
```

This can be used on any machine if the right PAT is used

Connection to the database

The database itself will run in Google's cloud SQL, this gives us an IP and a port to connect to.

- Install the Npgsql NuGet package of entity framework
- To this package, we will provide a connection string that contains the important information of the database.

Connection string:

```
"Server=[ip of server];Port=[port of server (5432)];Database=[database name];User Id=[username of database user];Password=[password of database user];"
```

Later we will also remove this string from the project and pass it to the application using environment variables.

Connection to other Google Cloud-services

Bucket

We will also use Google's cloud storage in the form of buckets, this way we can store bigger files of our application without filling up the VM's storage.

- Install the cloud storage NuGet package of google
- Get the JSON key as an environment variable
- Initialize the storage Client object

Google Compute engine metadata server

The Google Compute Engine Metadata Server is used to obtain metadata information within virtual machines (VMs) running on Google Cloud Platform (GCP). This metadata can be used for various purposes, such as configuring applications, managing access credentials, and dynamically adjusting application behavior based on environment conditions. Additionally, leveraging the metadata server allows developers to design compute engine instances in a stateless manner, where configurations and necessary information are fetched from the metadata server during instance startup, promoting scalability and flexibility in cloud-based deployments.

Risk analyzation and mitigation actions

Security Risks:

1. **Data Breach:** There's a risk of unauthorized access to sensitive data stored in the application's database or transmitted over the network.
    a. **Access Control:** Implement robust access controls to limit access to sensitive data to authorized users only.
    b. **Network Security:** Utilize firewalls
2. **Injection Attacks:** Exploiting input fields of the application like SQL injection, XSS, and other attacks may exploit vulnerabilities in the application.
    a. **Input Validation:** Implement strict input validation on all user inputs to prevent injection attacks.
3. **Authentication and Authorization:** Inadequate authentication and authorization mechanisms can lead to unauthorized access to resources.

    a. **Token-based Authentication:** Use tokens for authentication and ensure they have limited lifetimes to prevent unauthorized access.
    b. **Session Management:** Implement secure session management practices to prevent session hijacking and fixation attacks.
4. **DDoS Attacks:** The application may be vulnerable to Distributed Denial of Service attacks, leading to downtime and service unavailability.
    a. **DDoS Protection Services:** Utilize DDoS protection services provided by your cloud provider or third-party vendors.
    b. **Traffic Filtering:** Implement traffic filtering mechanisms to drop or throttle suspicious traffic during DDoS attacks.
    c. **Load Balancers:** Distribute incoming traffic across multiple servers using load balancers to mitigate the impact of DDoS attacks.

Infrastructure Risks

1. **Availability:** The Google Cloud Compute Engine infrastructure may face downtime due to maintenance, hardware failures, or unexpected outages.
    a. **High Availability Architecture:** Design your application with redundancy across multiple zones or regions within Google Cloud to minimize the impact of failures in a single location.
    b. **Monitoring and Alerts:** Implement comprehensive monitoring like health probes and alerting systems to quickly detect and respond to infrastructure issues before they affect availability.
2. **Scalability:** Inadequate scalability planning may result in performance issues during peak loads.
    a. **Auto-scaling:** Configure auto-scaling policies to automatically add or remove resources based on demand, ensuring that your application can handle peak loads efficiently.
    b. **Horizontal Scaling:** Design your application to scale horizontally by distributing workloads across multiple instances or microservices, rather than relying solely on vertical scaling.
3. **Vendor Lock-in:** Dependency on Google Cloud Compute Engine may pose risks in terms of vendor lock-in and limited flexibility in migrating to other platforms.
    a. **Use of Standards:** Utilize open standards and interoperable technologies wherever possible to minimize dependency on proprietary Google Cloud services.
    b. **Containerization:** Containerize your application using technologies like Docker and Kubernetes, which offer portability and flexibility to migrate between different cloud providers or on-premises environments.

Data management Risks

1. **Data Loss:** Inadequate backup and disaster recovery mechanisms may result in data loss in case of system failures.
    a. **Regular Backups:** Implement automated backup procedures for critical data and ensure backups are performed regularly, preferably with multiple copies stored in different locations.

    b. **Offsite Storage:** Store backups in geographically diverse locations or utilize Google Cloud's multi-region storage options to mitigate the risk of data loss due to regional disasters.
2. **Data Corruption:** Errors in data processing or storage may lead to data corruption or integrity issues.
    a. **Data Validation:** Implement data validation checks at various stages of data processing to identify and prevent errors that could lead to data corruption.
3. **Data Privacy:** Mishandling of sensitive data may violate privacy regulations and damage the organization's reputation.
    a. **Access Controls:** Implement fine-grained access controls and role-based permissions to restrict access to sensitive data to authorized users only.
    b. **Compliance with Regulations:** Stay informed about relevant data privacy regulations such as GDPR, CCPA, or HIPAA, and ensure compliance through policies, procedures, and technical controls.

Performance Risks

1. **Latency:** Network latency between users and the Google Cloud Compute Engine data center may affect application responsiveness.
    a. **Edge Computing:** Leverage Google Cloud's edge computing services to deploy application logic closer to end-users, reducing round-trip times and improving responsiveness.
    b. **Global Load Balancing:** Use Google Cloud's global load balancers to route traffic to the nearest available data center, reducing latency for users in different regions.
2. **Resource Contention:** Shared resources on the Compute Engine may lead to performance degradation during peak usage periods.
    a. **Horizontal Scaling:** Implement horizontal scaling by adding more instances or distributing workloads across multiple instances or microservices, allowing resources to be dynamically allocated based on demand.

Dependency Risks

1. **Third-party Components:** Dependency on third-party libraries or services may introduce vulnerabilities or compatibility issues.
    a. **Regular Updates:** Stay informed about updates and security patches for all third-party libraries and components used in your application. Regularly update these components to ensure that known vulnerabilities are addressed promptly.
2. **API Dependencies:** Integration with external APIs may lead to risks related to service availability, security, or changes in API versions.
    a. **Error Handling:** Implement robust error handling mechanisms to gracefully handle errors and exceptions encountered when interacting with external APIs. Provide informative error messages to users and log detailed diagnostic information for troubleshooting.

Operational Risks:

1. **Human Error:** Misconfigurations or human errors during deployment or maintenance activities may cause service disruptions or security vulnerabilities.
    a. **Automation:** Automate deployment and maintenance tasks
2. **Monitoring and Logging:** Inadequate monitoring and logging may delay the detection and mitigation of security incidents or performance issues.
    a. **Comprehensive Monitoring:** Implement comprehensive monitoring solutions to track system performance, resource utilization, and application health. Monitor key metrics such as CPU usage, memory consumption, and network traffic to identify potential issues.

Cost Risks:

1. **Resource Overprovisioning:** Poorly optimized resource allocation may lead to unnecessary costs.
    a. **Auto-scaling:** Implement auto-scaling policies to dynamically adjust resource allocation based on demand, scaling resources up during periods of high usage and down during periods of low usage.
2. **Cost Escalation:** Unexpected increases in usage or resource costs may exceed budget expectations.
    a. **Budget Alerts:** Set up budget alerts and notifications to monitor spending against predefined budget thresholds. Receive alerts when spending approaches or exceeds budget limits, allowing for timely intervention.