Cloud Computing

Midterm Project: Building a Cloud-Based File Sharing Application on Google Cloud Platform

Shayakhmetova Balzhan

27.10.24

1. Executive Summary

The project aimed to create a secure and scalable file-sharing application, allowing users to upload, download, rename, and delete files with ease. By leveraging modern technologies, the project not only addressed the need for efficient file management but also ensured high availability and robust security.

In terms of the technology stack, Java served as the backbone of the application. Java is known for its portability and strong community support, making it an ideal choice for building enterprise-level applications. Specifically, we utilized Spring Boot, a powerful framework that simplifies the development of Java applications. Spring Boot's opinionated approach allowed us to configure the application with minimal boilerplate code, enabling rapid development and deployment. It also provided built-in features such as dependency injection, which facilitated the management of application components and made the codebase more modular and easier to test.

For the frontend, we chose Thymeleaf, a modern server-side Java template engine. Thymeleaf integrates seamlessly with Spring Boot, enabling us to create dynamic web pages that respond to user actions in real time. The combination of Java and Thymeleaf allowed us to create a user-friendly interface, which significantly enhanced the user experience.

On the storage side, we utilized Google Cloud Platform, specifically Google Cloud Storage, to handle file uploads and storage. This cloud-based solution provided a scalable and secure environment, which was essential for managing potentially large volumes of files. Setting up the Google Cloud Storage bucket required careful configuration, including the management of access credentials to ensure that only authorized users could perform operations on the stored files.

One significant challenge we faced during the project was related to deploying the application on a Google Cloud VM. While the application built and ran successfully on my local machine, it failed to locate the necessary credential files when deployed on the VM. Initially, I had used cached credentials from the Google Cloud Console, which did not work in the VM environment. After resolving this issue by generating a new set of credentials from my local Google Cloud Console, I was able to properly configure the application for cloud access.

Another challenge involved securing the Virtual Private Cloud (VPC) network. Configuring the necessary firewall rules and ensuring secure access to the application were critical steps that required careful planning and execution. Additionally, I noticed that file uploads were initially slow when using the Google Cloud Console. This issue prompted me to optimize the upload process and explore solutions to enhance performance.

To ensure the application was robust and reliable, I implemented various testing methodologies, including unit tests using JUnit. These tests validated the backend functionalities, ensuring that all components interacted correctly. Moreover, I conducted manual testing to verify connectivity and overall performance, which provided a comprehensive understanding of how the application operated under different conditions.

Reflecting on the outcomes of the project, I believe that we successfully developed a fully operational file-sharing application that met its objectives. The integration of Java and Spring Boot allowed for a seamless backend experience, while Thymeleaf enhanced the user interface. The project not only provided practical experience in cloud deployment but also highlighted the importance of security and performance optimization.

Looking ahead, I see opportunities for future enhancements. Incorporating Docker and Kubernetes could significantly improve the application's scalability and deployment process, especially in handling traffic spikes or increasing user demand. While these technologies may introduce additional costs, they offer substantial benefits in terms of containerization and orchestration, which would be valuable for a production-level deployment.

2. Introduction

Cloud computing is a model of delivering computing services over the internet, where resources such as servers, storage, databases, software, and applications are provided as a service to users on-demand. This approach has become a crucial aspect of modern application development, offering numerous benefits.

In modern application development, cloud computing provides scalability, flexibility, cost-effectiveness, and collaboration. It allows developers to scale their applications up or down to meet changing demands, without having to worry about provisioning or managing physical infrastructure. Additionally, cloud computing provides access to a wide range of services and tools, enabling developers to choose the best tools for their specific needs. Furthermore, it eliminates the need for upfront capital expenditures and reduces operational costs, making it a more cost-effective option for many organizations. Finally, cloud computing enables teams to collaborate more effectively, as they can access the same resources and data from anywhere, at any time.

When selecting a cloud platform for a project, several factors are considered, including security, scalability, cost, integration, and support. Google Cloud Platform (GCP) is an attractive choice for many developers and organizations due to its strong focus on security, scalability, cost-effectiveness, and integration. GCP provides a range of scalable services, including Compute Engine, App Engine, and Cloud Functions, which enable developers to build and deploy applications quickly and efficiently. Additionally, GCP offers a pay-as-you-go pricing model, which can help reduce costs and improve budgeting.

GCP is particularly well-suited for projects that require machine learning, big data, serverless computing, or containerization. Its machine learning services, including TensorFlow, AutoML, and Cloud AI Platform, enable developers to build and deploy intelligent applications. Its big data services, including Bigtable, Cloud Storage, and Cloud Dataflow, provide a scalable and secure way to store and process large datasets. Furthermore, GCP's serverless computing services, including Cloud Functions and Cloud Run, enable developers to build and deploy applications without worrying about the underlying infrastructure. Finally, GCP's containerization services, including Kubernetes Engine and Cloud Run, provide a secure and scalable way to deploy and manage containerized applications.

3. Project Objectives

In today's digital landscape, the need for secure and efficient file-sharing solutions has never been more crucial. As part of my project, I aimed to develop a comprehensive file-sharing application that leverages the robust capabilities of Google Cloud products to ensure a secure, scalable, and user-friendly experience. The motivation behind this endeavor was not only to create a functional application but also to harness the powerful tools offered by Google Cloud Platform (GCP) to facilitate seamless file management and storage.

My vision for the project was to establish a backend system using Java and Spring Boot, which would serve as the backbone for all file operations, including uploading, renaming, deleting, and retrieving files. This choice of technology was driven by the need for a reliable and flexible framework that could handle the intricacies of file management while ensuring high performance. By implementing RESTful APIs, I aimed to create an intuitive interface for the frontend, which I developed using Thymeleaf. This would allow users to interact with the application effortlessly, performing tasks such as uploading files and managing their existing files with ease.

To enhance the application's reliability, I recognized the importance of scalability and high availability. Therefore, I turned to GCP for deployment. By utilizing Google Cloud Storage for file storage, I could take advantage of its scalable infrastructure, ensuring that the application could handle a growing number of users and file transactions without compromising performance. Additionally, I integrated Google Application Credentials to secure access to the cloud resources, reinforcing the application's security.

Throughout this journey, I was determined to implement best practices in security to protect user data and maintain trust. The application was designed with HTTPS to secure data in transit, and strict authentication mechanisms were enforced to prevent unauthorized access. Furthermore, by hosting the project on GitHub with a private repository, I ensured that the source code remained secure and maintained.

Ultimately, this project was more than just an exercise in software development; it was an opportunity to explore the potential of Google Cloud products and how they can be leveraged to build a secure and scalable file-sharing application. I aimed not only to meet the functional requirements but also to create a platform that users could rely on for their file management needs, all while reinforcing my own understanding of cloud technologies and best practices in application development.

My project objectives were:

- Develop a secure backend application using Java and Spring Boot for managing file operations.
- Create a user-friendly frontend interface with Thymeleaf to facilitate file uploads, renaming, deletion, and retrieval.
- Utilize Google Cloud Platform (GCP) to ensure high availability and scalability of the application.
- Implement security best practices, including HTTPS and Google Application Credentials, to protect user data.
- Set up version control with a private GitHub repository for secure code management.
- Conduct comprehensive testing to ensure all functionalities operate smoothly and securely.

4. Cloud Computing Overview

Cloud computing is a model of delivering computing services over the internet, where resources such as servers, storage, databases, software, and applications are provided as a service to users on-demand. The principles of cloud computing are based on five essential characteristics:

1. **On-demand self-service**: Users can independently provision and manage computing resources as needed without requiring human intervention from the service provider. This automation enhances productivity and accelerates deployment by allowing resources to be allocated and managed through a self-service portal.
2. **Broad network access**: Cloud resources are accessible from any location with an internet connection. This flexibility enables access from a variety of devices, including mobile phones, laptops, desktops, and tablets, ensuring that users can work from anywhere.
3. **Resource pooling**: Cloud providers utilize multi-tenancy, pooling computing resources to serve multiple customers simultaneously. These resources (e.g., storage, memory, CPU) can be dynamically allocated and reallocated, adapting to fluctuating demands without requiring physical relocation.
4. **Rapid elasticity**: Cloud resources can be scaled up or down quickly in response to real-time demand changes. This capability is particularly beneficial for businesses with variable workloads, as it allows them to pay only for what they use while being able to handle surges.

5. **Measured service**: Cloud providers track resource usage with a metered system, charging users based on actual consumption (e.g., compute time, storage). This approach optimizes cost-effectiveness and resource usage, as users only pay for what they use, eliminating wasted capacity.

There are three main deployment models for cloud computing:

1. **Public Cloud**:
    a. Managed by third-party providers (e.g., AWS, Azure, GCP), public clouds are accessible over the internet to multiple users.
    b. Public clouds are scalable and ideal for handling peak loads, making them a cost-effective option for businesses that do not require stringent data security or compliance.
2. **Private Cloud**:
    a. A private cloud is dedicated to a single organization, offering enhanced control, security, and privacy.
    b. It can be hosted on-premises or by a third party. This model is suitable for businesses handling sensitive information or adhering to strict regulatory requirements.
3. **Hybrid Cloud**:
    a. Hybrid clouds combine public and private cloud elements, allowing data and applications to be shared across both.
    b. This approach enables businesses to scale on-demand while maintaining security and control over critical data, providing the flexibility to deploy specific workloads on public or private resources as needed.

There are also three main service models for cloud computing, the differences described in Table 1 from Assignment 1.

	Infrastructure as a Service (IaaS)	Platform as a Service (PaaS)	Software as a Service (SaaS)
Control	High	Medium	Low
Flexibility	High	Medium	Low
Use cases	● Provides virtualized computing resources (e.g., servers, storage, networks) ● Users have full control	● Provides a complete development and deployment environment for applications	● Provides software applications over the internet ● Users access the software applications through a web

	over the infrastructure and can configure it to meet their specific needs ● Users are responsible for managing and maintaining the infrastructure ● Examples: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP)	● Users have control over the application and data, but not the underlying infrastructure ● Users do not need to manage or maintain the infrastructure ● Examples: Heroku, Google App Engine, Microsoft Azure App Service	browser or mobile app ● Users do not have control over the underlying infrastructure or application code ● Examples: Salesforce, Microsoft Office 365, Google Workspace
GCP services	■ Compute Engine: provides virtual machines (VMs) and bare-metal instances ■ Google Kubernetes Engine (GKE): provides managed Kubernetes clusters ■ Cloud VPN: provides virtual private networks (VPNs) for	■ App Engine: provides a managed platform for building web applications ■ Google Cloud Functions: provides a serverless platform for building event-driven applications ■ Cloud Run: provides a fully managed platform for building	■ Google Workspace (formerly G Suite): provides a suite of productivity applications, including Gmail, Google Drive, Google Docs, and more ■ Google Cloud Search: provides a managed search platform for

	secure connectivity ■ Cloud Interconnect: provides dedicated connections to Google's network	containerized applications ■ Cloud Tasks: provides a managed platform for building background tasks and workflows	searching and indexing data ■ Google Cloud Identity and Access Management (IAM): provides a managed identity and access management platform for securing resources ■ Google Cloud Data Fusion: provides a managed data integration platform for integrating and processing data

Table 1: IaaS, PaaS, SaaS differences

5. Google Cloud Platform: Core Services

1. Compute Engine is a service provided by Google Cloud Platform (GCP) that allows you to create and manage virtual machines (VMs) in the cloud. The primary use case of Compute Engine is to run applications that require a lot of processing power, memory, or specific software configurations. This can include:
    - High-performance computing: Compute Engine is perfect for tasks that require intense processing power, such as scientific simulations, data analytics, or machine learning model training.
    - Customized environments: You can create VMs with specific operating systems, software, and configurations to match your application's requirements.
    - Disaster recovery: Compute Engine allows you to create redundant VMs in different regions, ensuring that your applications remain available even in the event of an outage.

- Development and testing: You can use Compute Engine to create temporary VMs for development, testing, or staging environments, and then shut them down when you're done.

2. Google Kubernetes Engine (GKE) simplifies the management of containerized applications in several ways:
    - Automated Deployment: GKE automates the deployment of containerized applications, ensuring that your application is always running and available. You can simply define your application's configuration and let GKE handle the rest.
    - Orchestration: GKE provides a built-in orchestration engine that manages the lifecycle of your containers, including scaling, rolling updates, and self-healing.
    - Resource Management: GKE provides a scalable and flexible resource management system, allowing you to easily allocate resources to your containers and ensure that your application is running efficiently.
    - Networking: GKE provides a managed networking system, allowing you to easily configure and manage network policies for your containers.
    - Monitoring and Logging: GKE provides built-in monitoring and logging capabilities, allowing you to easily track the performance and health of your containers.
    - Security: GKE provides a secure environment for your containers, with features such as network policies, secret management, and identity and access management.
    - Scalability: GKE allows you to easily scale your containers horizontally (add more replicas) or vertically (increase resources), ensuring that your application can handle changing workloads.
    - Rolling Updates: GKE allows you to perform rolling updates of your containers, ensuring that your application remains available during updates.
    - Self-Healing: GKE provides self-healing capabilities, automatically restarting containers that fail or become unresponsive.
    - Integration with Other GCP Services: GKE integrates seamlessly with other GCP services, such as Cloud Storage, Cloud SQL, and Cloud Pub/Sub, making it easy to build complex applications.
3. Cloud Storage offers several advantages for data management, including:
    - Scalability: Cloud Storage allows you to store large amounts of data without worrying about running out of storage space. You can easily scale up or down as needed.
    - Flexibility: Cloud Storage provides a range of storage options, including object storage, block storage, and file storage, allowing you to choose the best fit for your data.
    - Accessibility: Cloud Storage provides secure and easy access to your data from anywhere, at any time, using a web browser or API.
    - Durability: Cloud Storage provides built-in redundancy and data replication, ensuring that your data is safe and available even in the event of hardware failure or data corruption.

- Cost-Effectiveness: Cloud Storage provides a pay-as-you-go pricing model, allowing you to only pay for the storage and bandwidth you use.
- Security: Cloud Storage provides robust security features, including encryption, access controls, and auditing, to protect your data from unauthorized access.
- Integration: Cloud Storage integrates seamlessly with other Google Cloud services, such as Google Cloud Dataflow, Google Cloud Bigtable, and Google Cloud SQL, making it easy to build data pipelines and workflows.
- Data Retention: Cloud Storage provides flexible data retention policies, allowing you to set retention periods for your data and automatically delete or archive data when it reaches a certain age.
- Data Analytics: Cloud Storage provides built-in analytics and reporting capabilities, allowing you to track data usage, storage capacity, and other metrics.
- Disaster Recovery: Cloud Storage provides disaster recovery capabilities, allowing you to easily recover data in the event of a disaster or data loss.

4. A business would choose BigQuery for their data analysis needs for several reasons:
    - Scalability: BigQuery can handle massive amounts of data, making it an ideal choice for businesses with large datasets.
    - Speed: BigQuery is designed for fast query performance, allowing businesses to quickly analyze their data and get insights.
    - Cost-Effectiveness: BigQuery provides a pay-as-you-go pricing model, which means businesses only pay for the data they analyze, making it a cost-effective solution.
    - Ease of Use: BigQuery is a fully-managed service, which means businesses don't need to worry about managing the underlying infrastructure, allowing them to focus on analyzing their data.
    - Integration: BigQuery integrates seamlessly with other Google Cloud services, such as Google Cloud Dataflow, Google Cloud Storage, and Google Cloud SQL, making it easy to build data pipelines and workflows.
    - Security: BigQuery provides robust security features, including encryption, access controls, and auditing, to protect sensitive data.
    - SQL Support: BigQuery supports standard SQL, making it easy for businesses to analyze their data using familiar tools and techniques.
    - Data Visualization: BigQuery provides built-in data visualization capabilities, allowing businesses to easily create reports and dashboards to gain insights from their data.
    - Machine Learning: BigQuery provides built-in machine learning capabilities, allowing businesses to build predictive models and gain insights from their data.
    - Collaboration: BigQuery provides collaboration features, allowing businesses to easily share data and insights with team members and stakeholders.

6. Virtual Machines in Google Cloud

Virtual machines (VMs) in Google Cloud are scalable, customizable computing resources that run on Google's infrastructure. They allow users to deploy and manage virtualized operating systems, offering flexible configurations for processing power, storage, and networking.
Creating VM:

1. Set up authentication. Authentication is the process by which your identity is verified for access to Google Cloud services and APIs
2. Setting up VM instance in Image 1

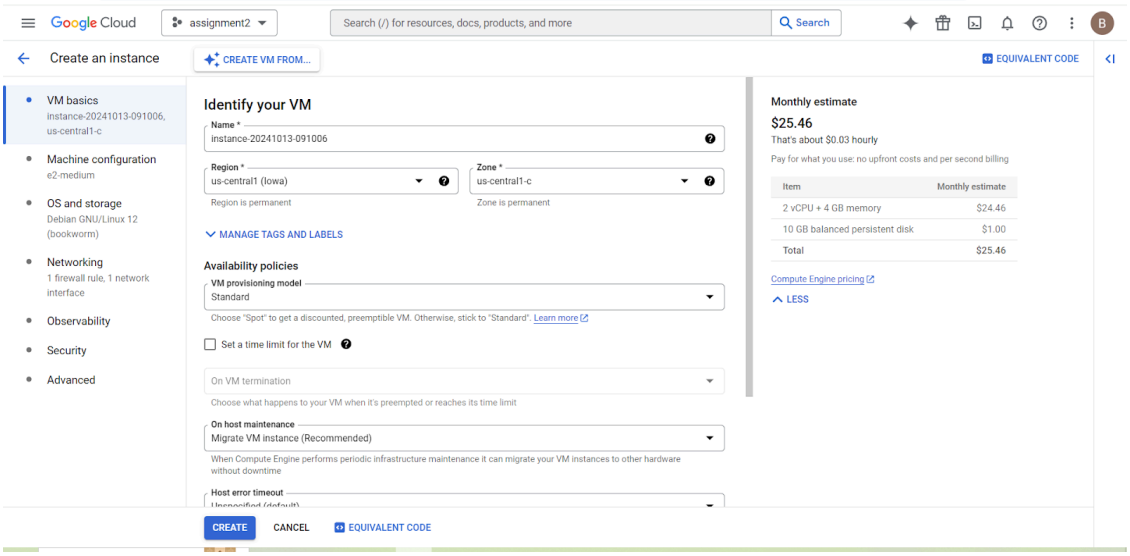


Image 1: Setting up VM

3. Setting up Firewall rules in Image 2

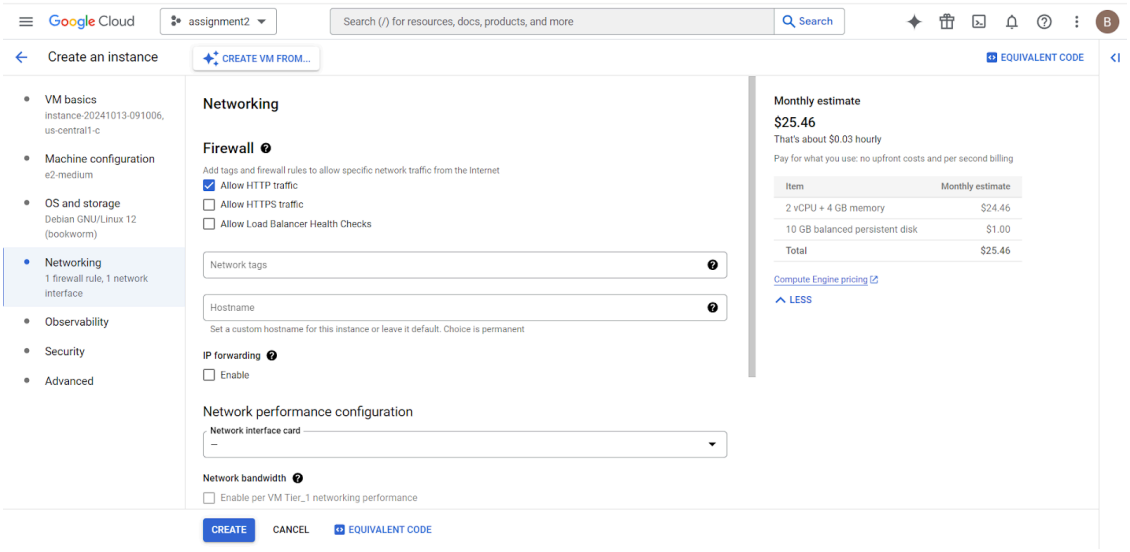


Image 2: Firewall rules

4. Successful creation in Image 3

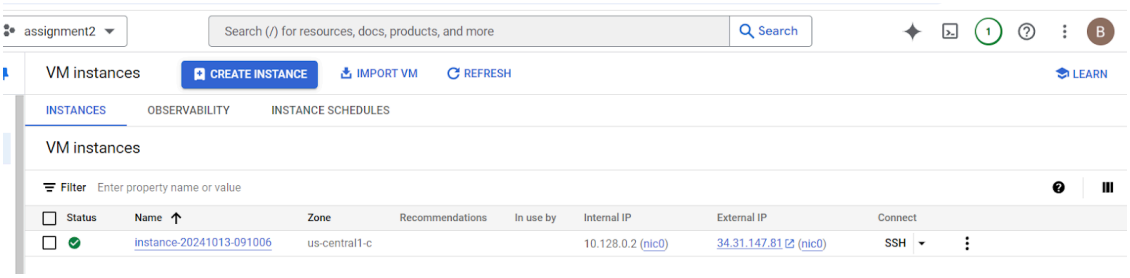


Image 3: Successful creation of the VM

5. Accessing VM instance via ssh in Image 4

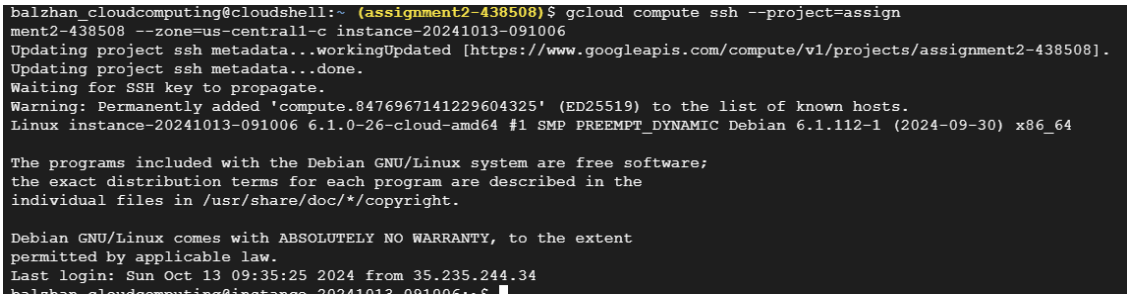


Image 4: VM instance via ssh

6. Updating the packages in Image 5

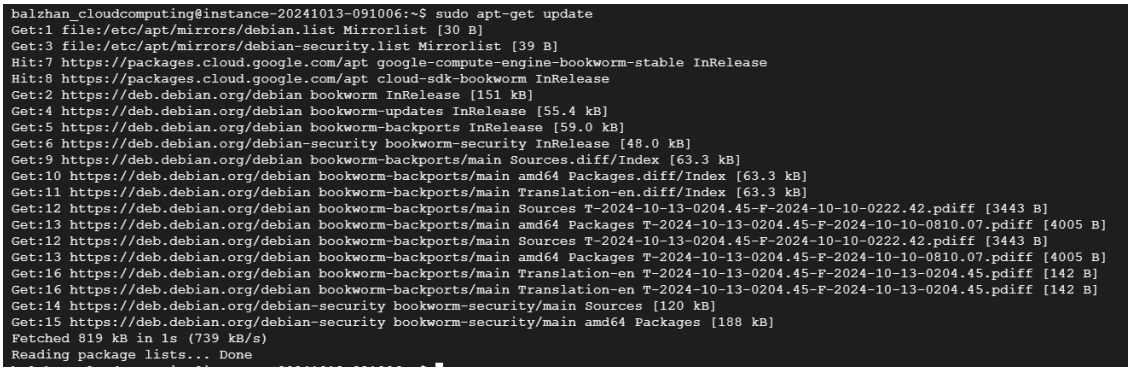


Image 5: Updating packages

7. Installing apache on the VM in Image 6

```
balzhan_cloudcomputing@instance-20241013-091006:~$ sudo apt-get install apache2
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap libjansson4 liblua5.3-0 ssl-cert
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom www-browser
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap libjansson4 liblua5.3-0 ssl-cert
0 upgraded, 11 newly installed, 0 to remove and 0 not upgraded.
Need to get 2379 kB of archives.
After this operation, 8468 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 file:/etc/apt/mirrors/debian.list Mirrorlist [30 B]
Get:2 file:/etc/apt/mirrors/debian-security.list Mirrorlist [39 B]
Get:3 https://deb.debian.org/debian bookworm/main amd64 libapr1 amd64 1.7.2-3 [102 kB]
Get:4 https://deb.debian.org/debian bookworm/main amd64 libaprutil1 amd64 1.6.3-1 [87.8 kB]
Get:5 https://deb.debian.org/debian bookworm/main amd64 libaprutil1-dbd-sqlite3 amd64 1.6.3-1 [13.6 kB]
Get:6 https://deb.debian.org/debian bookworm/main amd64 libaprutil1-ldap amd64 1.6.3-1 [11.8 kB]
Get:7 https://deb.debian.org/debian bookworm/main amd64 libjansson4 amd64 2.14-2 [40.8 kB]
```

Image 6: Installing apache

8. Installing java on the VM in Image 7

```
balzhan_cloudcomputing@instance-20241013-091006:~$ sudo apt update
sudo apt install openjdk-17-jdk -y
java -version
Get:1 file:/etc/apt/mirrors/debian.list Mirrorlist [30 B]
Get:2 file:/etc/apt/mirrors/debian-security.list Mirrorlist [39 B]
Hit:7 https://packages.cloud.google.com/apt google-compute-engine-bookworm-stable InRelease
Hit:3 https://deb.debian.org/debian bookworm InRelease
Hit:8 https://packages.cloud.google.com/apt cloud-sdk-bookworm InRelease
Get:4 https://deb.debian.org/debian bookworm-updates InRelease [55.4 kB]
Get:5 https://deb.debian.org/debian bookworm-backports InRelease [59.0 kB]
Get:6 https://deb.debian.org/debian-security bookworm-security InRelease [48.0 kB]
Get:9 https://deb.debian.org/debian bookworm-backports/main Sources.diff/Index [63.3 kB]
Get:10 https://deb.debian.org/debian bookworm-backports/main amd64 Packages.diff/Index [63.3 kB
]
Get:11 https://deb.debian.org/debian bookworm-backports/main Translation-en.diff/Index [63.3 kB
]
Get:12 https://deb.debian.org/debian bookworm-backports/main Sources T-2024-10-26-2007.03-F-202
4-10-26-0206.54.pdiff [1121 B]
Get:13 https://deb.debian.org/debian bookworm-backports/main amd64 Packages T-2024-10-26-2007.0
3-F-2024-10-26-0206.54.pdiff [1689 B]
Get:12 https://deb.debian.org/debian bookworm-backports/main Sources T-2024-10-26-2007.03-F-202
4-10-26-0206.54.pdiff [1121 B]
Get:13 https://deb.debian.org/debian bookworm-backports/main amd64 Packages T-2024-10-26-2007.0
3-F-2024-10-26-0206.54.pdiff [1689 B]
Get:17 https://deb.debian.org/debian bookworm-backports/main Translation-en T-2024-10-26-2007.0
3-F-2024-10-26-2007.03.pdiff [662 B]
Get:17 https://deb.debian.org/debian bookworm-backports/main Translation-en T-2024-10-26-2007.0
3-F-2024-10-26-2007.03.pdiff [662 B]
Get:14 https://deb.debian.org/debian-security bookworm-security/main Sources [123 kB]
Get:15 https://deb.debian.org/debian-security bookworm-security/main amd64 Packages [190 kB]
Get:16 https://deb.debian.org/debian-security bookworm-security/main Translation-en [116 kB]
Fetched 785 kB in 1s (675 kB/s)
```

Image 7: Installing java on the VM

9. Uploading jar on VM instance in Image 8

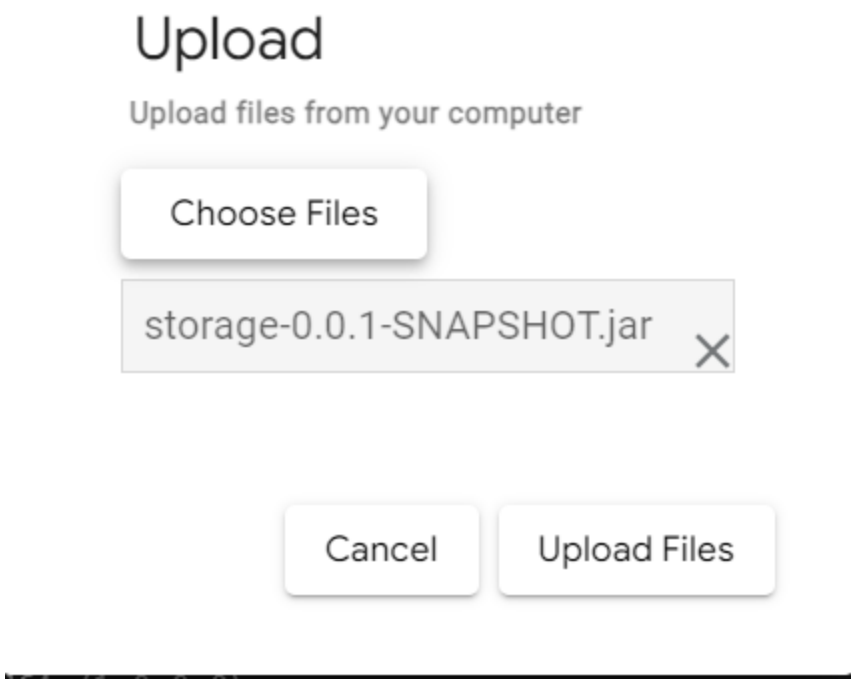


Image 8: Uploading jar on the VM instnce

10. Transferring file to VM instance in Image 9

```
balzhan_cloudcomputing@instance-20241013-091006:~$ gcloud compute scp storage-0.0.1-SNAPSHOT.ja
r instance-20241013-091006:~/
Did you mean zone [us-central1-c] for instance: [instance-20241013-091006] (Y/n)?  y

storage-0.0.1-SNAPSHOT.jar                                   100%   70MB  44.3MB/s   00:01
balzhan_cloudcomputing@instance-20241013-091006:~$
```

Image 9: Upload jar file

11. Deploying application but getting error of finding file in the location in Image 10. During the deployment of the application, I encountered persistent issues with locating the Google Cloud credentials file. Initially, I tried using a relative path within the code to access the file, aiming to make the configuration more portable. This approach worked locally but failed during deployment on other environments. To address this, I moved the credentials file to the `config` directory under `src/main/resources`, assuming it would resolve the issue by keeping it within the resources folder accessible to the application. I also tried explicitly specifying the file location in the `application.properties` file, hoping this configuration would ensure the application could locate the file during runtime. Despite these adjustments, the application continued to throw an error indicating that it couldn't find the credentials file. I then incorporated the file with the `ResourceLoader` in

Spring, which should have allowed the application to locate the file within the classpath. However, even with these efforts, the error persisted during deployment. This led me to suspect that additional adjustments, possibly in the deployment environment's configuration, were necessary to ensure the credentials file's availability across different machines and deployment setups.

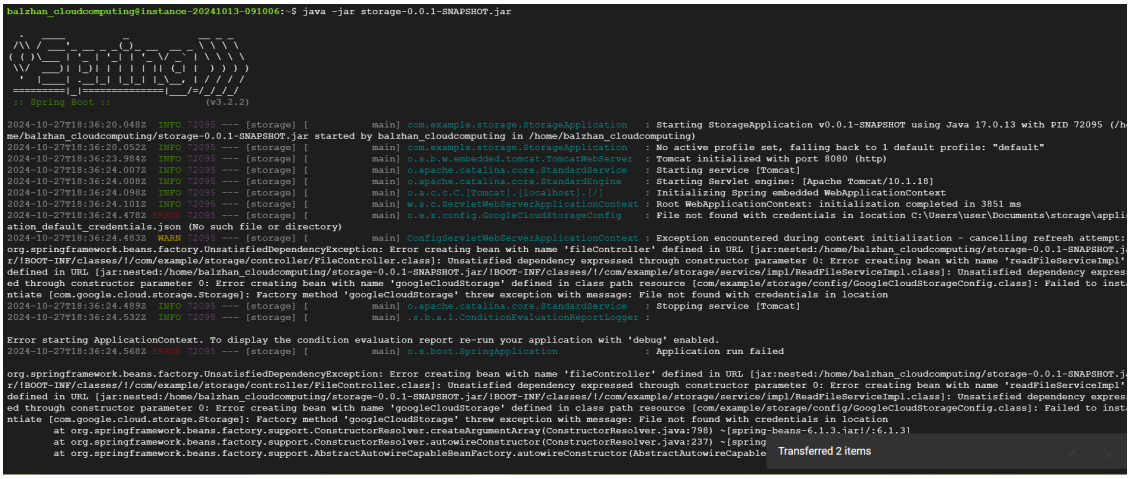


Image 10: Deploying file

12. But locally the application is running perfectly in Image 11

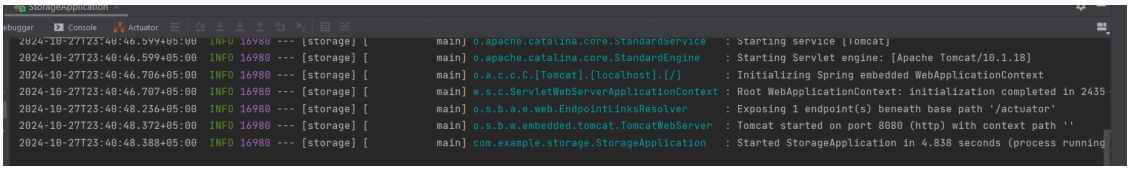


Image 11: The run on the local machine

13. Upload using the api in Image 12

Image 12: Upload

14. Rename in Image 13

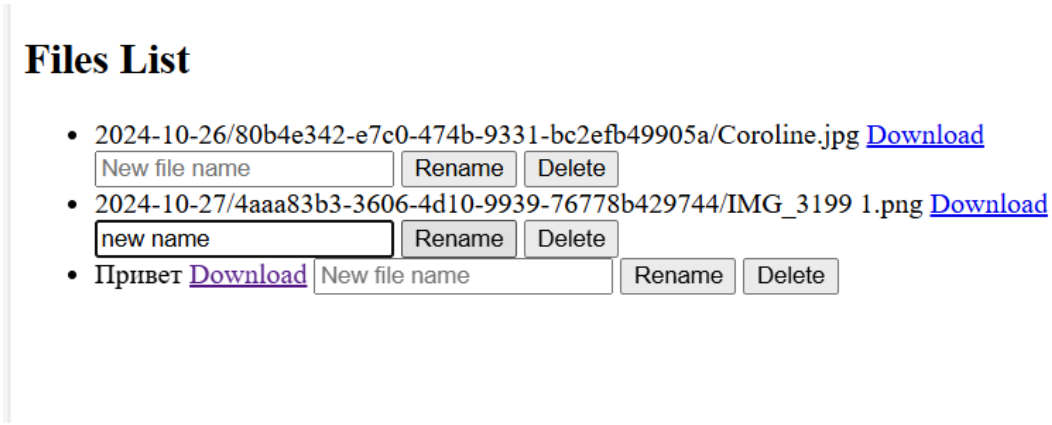


Image 13: Rename

15. Rename successful in Image 14

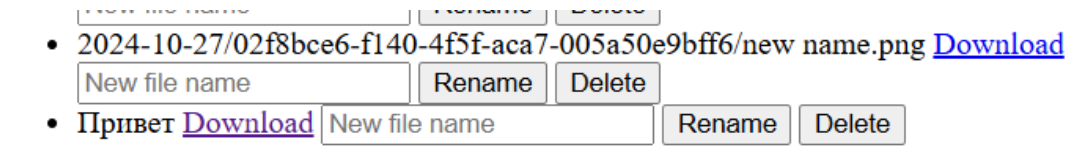


Image 14: Rename

15. Delete of the file in Image 15

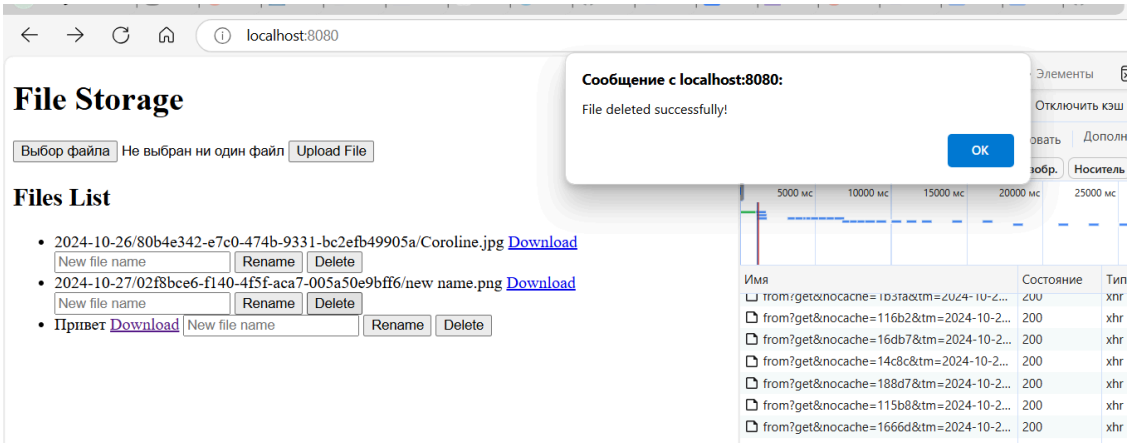


Image 15: Delete of the file

16. The organisation of the files in the bucket in Image 16

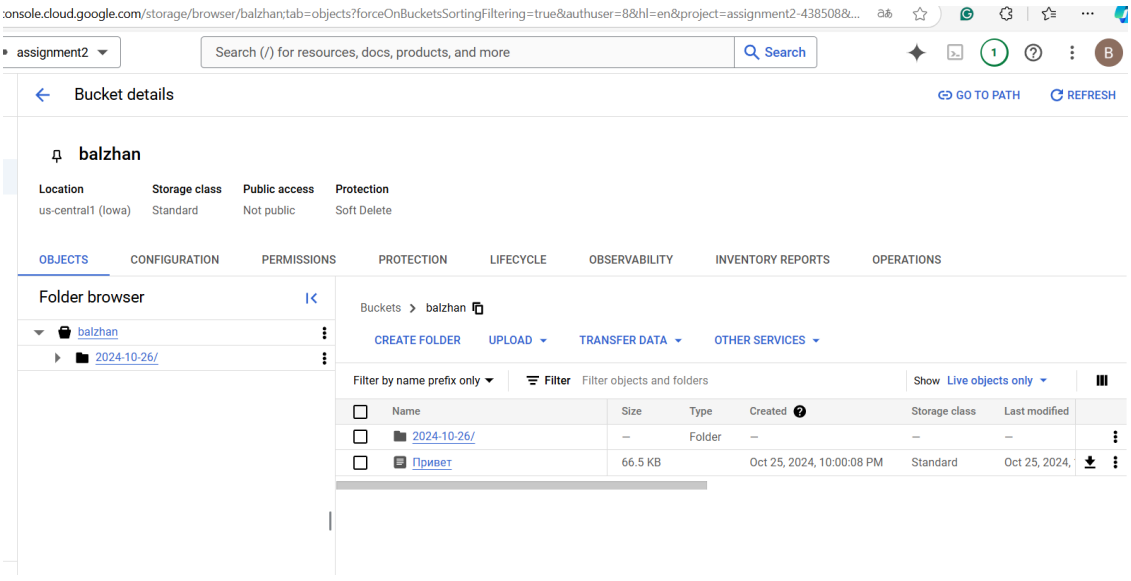


Image 16: Bucket on cloud

Because of the error with deployment I deployed Hello World project

17. Deploying a web page with Hello World! In Image 17

Image 17: Deploying a web page

18. Accessing deployed page in Image 18

Image 18: Hello World! page

7. Storage Solutions in Google Cloud

Cloud Storage in Google Cloud is really useful for a wide range of tasks, especially when dealing with large amounts of data. One of the main use cases is data backup and archiving. Companies can use it to securely store their important data without worrying about running out of space, and it's great for long-term archiving. Another use case is serving static content, like images or videos for websites or apps, which makes things like hosting media files easier and more efficient, especially with Google's global infrastructure.

For data-heavy projects, such as analytics or machine learning, Cloud Storage comes in handy too. It allows engineers and data scientists to store massive datasets and integrate them with other Google Cloud services. Plus, media companies often use it to store and deliver content, like video streaming services.

When it comes to lifecycle management, it adds a lot of benefits. It automates things like moving older data to cheaper storage classes (like Coldline or Archive) when you don’t need to access them frequently, which saves on costs. I think this is one of the best ways to ensure you're not paying more than necessary for storage you don’t need all the time. It also helps with data retention, especially when you need to delete data after a certain time to meet compliance rules. Overall, lifecycle management helps you stay organized and efficient without having to manually track your data storage.

1. Creating a bucket in Image 19

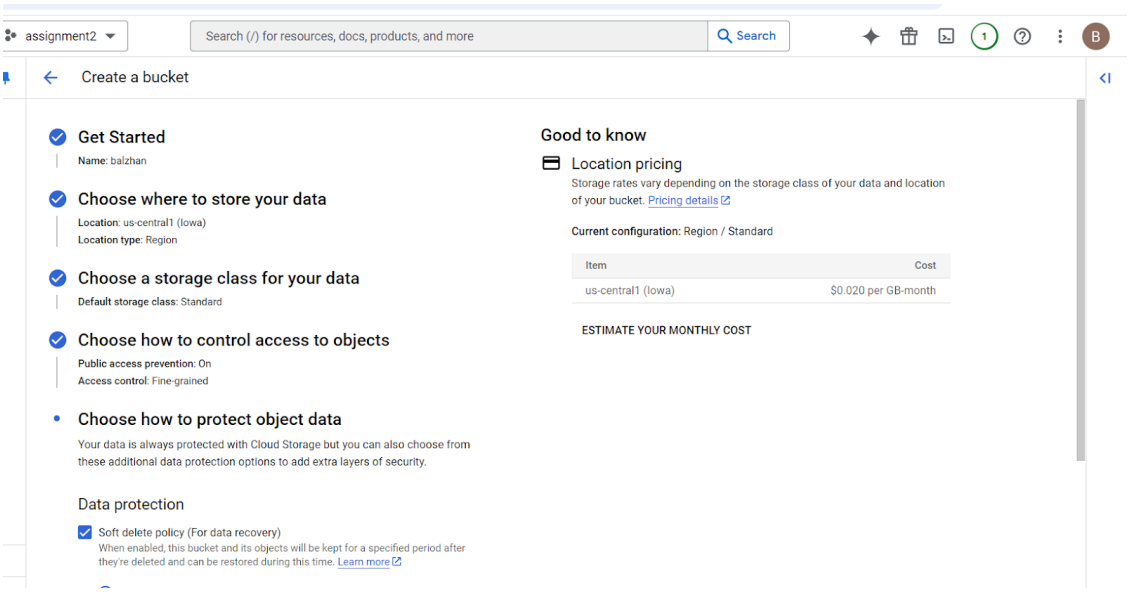


Image 19: Creating bucket

2. Successful creation of the bucket

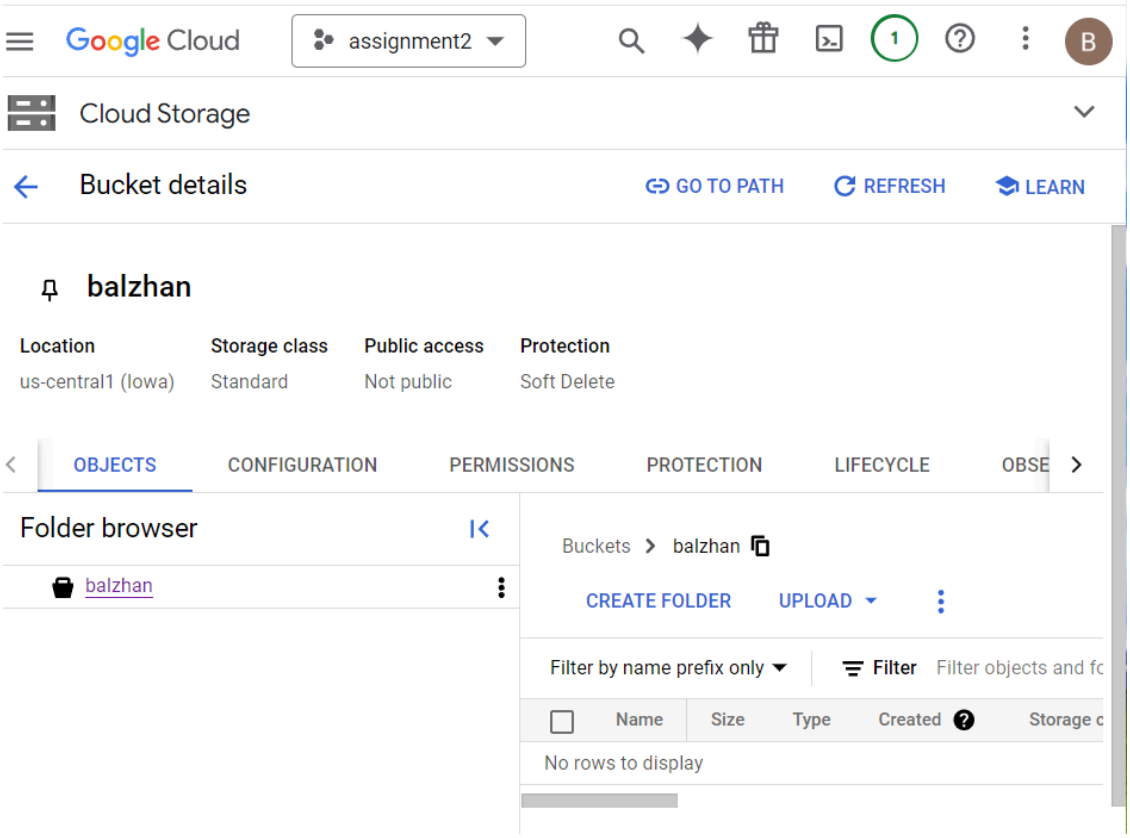


Image 20: Creation of the bucket

3. Setting bucket as public or private in Image 21

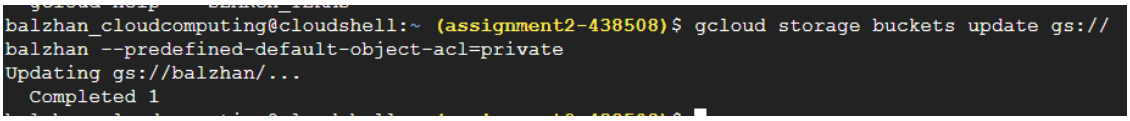


Image 21: Private bucket

4. Uploading object in Image 22

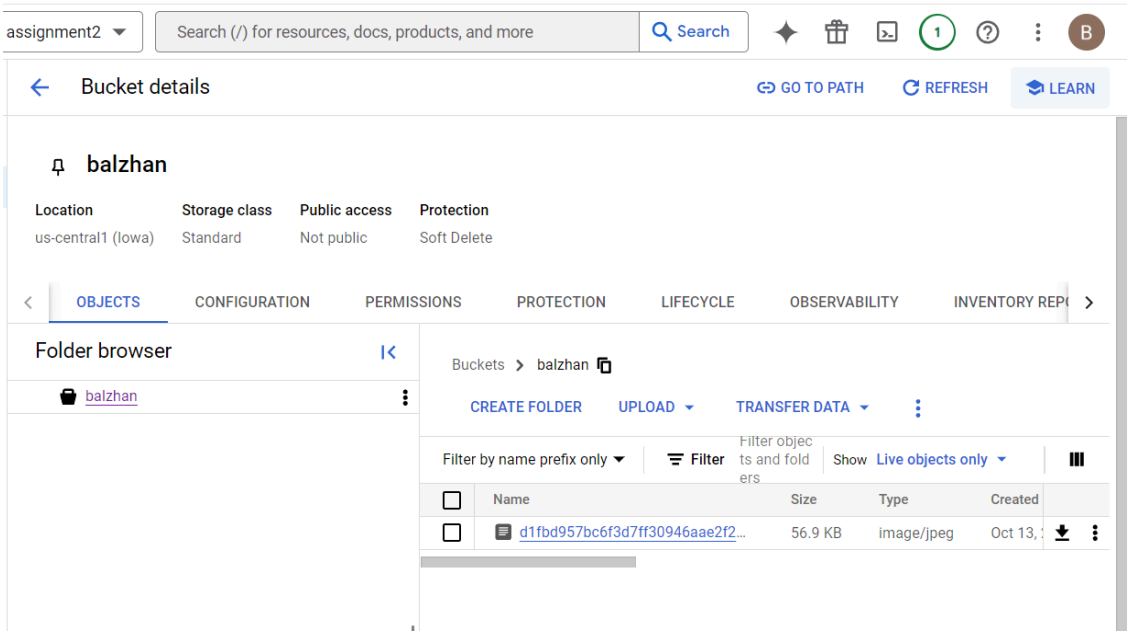


Image 22: Uploading file

5. Adding lifecycle rule in Image 23 to delete objects after 30 days

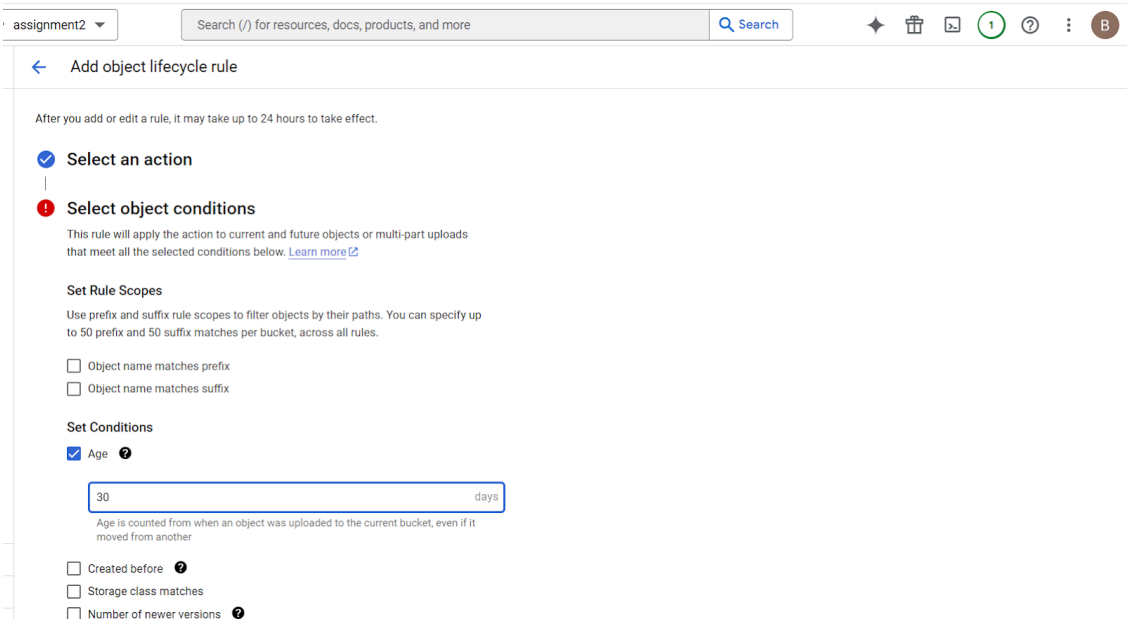


Image 23: Setting up lifecycle rules

In developing the file-sharing application, I implemented a well-organized bucket structure within Google Cloud Storage to enhance the management of uploaded files. Each file is stored using a naming convention that incorporates a UUID, the upload date, and the original file name. This approach ensures uniqueness and prevents any potential naming conflicts.

The use of a UUID (Universally Unique Identifier) provides a reliable way to uniquely identify each file, regardless of its original name. By appending the upload date to the filename, we can also easily track when each file was uploaded, which can be particularly useful for version control and auditing purposes. The combination of these elements creates a structured naming convention that not only maintains uniqueness but also allows for easier retrieval and management of files within the bucket.

This organized bucket structure aligns with best practices for cloud storage management and facilitates seamless integration with the application, ensuring that file operations such as upload, download, and deletion can be performed efficiently and effectively.

8. Networking in Google Cloud

In Google Cloud, a Virtual Private Cloud (VPC) is a scalable and flexible network that allows users to create an isolated environment for their resources. With a Google Cloud VPC, users can define their own IP ranges, subnets, and control routing rules, enabling secure communication between different Google Cloud services. It offers features like global networking, where a single VPC can span multiple regions, and built-in security through firewall rules and private access options. This ensures that users can manage and secure their workloads efficiently while maintaining flexibility for both internal and external connectivity.

1. Creating VPC network in Image 24

```
balzhan_cloudcomputing@cloudshell:~ (assignment2-438508)$ gcloud compute networks create balz
han-network --subnet-mode=custom --bgp-routing-mode=regional --mtu=1460
Created [https://www.googleapis.com/compute/v1/projects/assignment2-438508/global/networks/ba
lzhan-network].
NAME: balzhan-network
SUBNET_MODE: CUSTOM
BGP_ROUTING_MODE: REGIONAL
IPV4_RANGE:
GATEWAY_IPV4:

Instances on this network will not be reachable until firewall rules
are created. As an example, you can allow all internal traffic between
instances as well as SSH, RDP, and ICMP by running:

$ gcloud compute firewall-rules create <FIREWALL_NAME> --network balzhan-network --allow tcp,
udp,icmp --source-ranges <IP_RANGE>
$ gcloud compute firewall-rules create <FIREWALL_NAME> --network balzhan-network --allow tcp:
22,tcp:3389,icmp
```

Image 24: Creating VPC network

2. Creating firewall rule in Image 25

```
balzhan_cloudcomputing@cloudshell:~ (assignment2-438508)$ gcloud compute firewall-rules creat
e balzhan-firewall-network-rule --network balzhan-network --allow tcp:22,tcp:3389,icmp
Creating firewall...working..Created [https://www.googleapis.com/compute/v1/projects/assignme
nt2-438508/global/firewalls/balzhan-firewall-network-rule].
Creating firewall...done.
NAME: balzhan-firewall-network-rule
NETWORK: balzhan-network
DIRECTION: INGRESS
PRIORITY: 1000
ALLOW: tcp:22,tcp:3389,icmp
DENY:
DISABLED: False
```

Image 25: Applying firewall rule

3. Adding subnet in Image 26

```
balzhan_cloudcomputing@cloudshell:~ (assignment2-438508)$ gcloud compute networks subnets create balzhan-subnet \
    --network=balzhan-network \
    --range=10.128.0.0/20 \
    --region=us-central1
Created [https://www.googleapis.com/compute/v1/projects/assignment2-438508/regions/us-central1/subnetworks/balzhan-subnet].
NAME: balzhan-subnet
REGION: us-central1
NETWORK: balzhan-network
RANGE: 10.128.0.0/20
STACK_TYPE: IPV4_ONLY
IPV6_ACCESS_TYPE:
INTERNAL_IPV6_PREFIX:
EXTERNAL_IPV6_PREFIX:
```

Image 26: Creating subnet

4. Adding VM instance to the network in Image 27

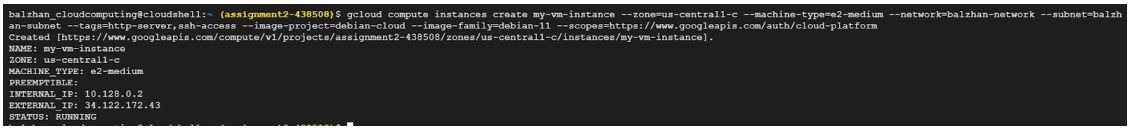


Image 27: Adding VM instance to the network

5. Accessing the internet (pinging www.google.com) in Image 28

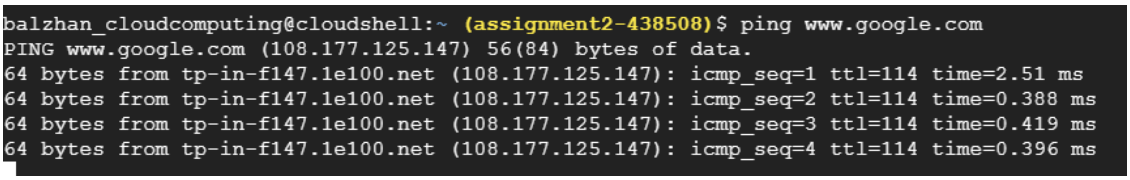


Image 28: Pinging api

Networking is a critical component of cloud infrastructure because it enables communication between cloud services, applications, and users. A well-designed cloud network ensures secure, efficient, and reliable data transmission, supporting scalability, load balancing, and global access while minimizing latency and downtime. Without robust networking, cloud environments would struggle to deliver the performance and flexibility that businesses rely on for modern, distributed applications.

In building a secure and optimized networking environment for the file-sharing application deployed on Google Cloud Platform (GCP), two critical components were addressed: firewall rules and load balancing. Both play a vital role in safeguarding the application from external threats while ensuring reliable performance.

Firewall rules are essential for controlling the traffic flow to and from the virtual machine instances hosting the application. In GCP, firewall rules allow us to define specific conditions under which traffic is permitted or denied based on IP addresses, protocols, and ports. The following strategies were implemented:

1. **Restricting Incoming Traffic**:
    - **Default Deny Policy**: By default, all incoming traffic was denied, and only specific rules were added to allow traffic. This approach minimizes the risk of unauthorized access.
    - **Allowing HTTP/HTTPS Traffic**: Firewall rules were created to permit incoming traffic on ports 80 (HTTP) and 443 (HTTPS), ensuring that users can access the application securely. This allows clients to communicate with the application while maintaining security.
2. **Restricting Outgoing Traffic**:

    - **Controlled Egress**: Outgoing traffic was restricted to only necessary services, reducing exposure to potential external threats. For instance, only traffic needed for updates or necessary external APIs was allowed.
3. **IP Whitelisting**:
    - **Trusted IP Ranges**: For administrative access, firewall rules were established to only allow specific trusted IP ranges, further securing access to the backend and management interfaces.
4. **Logging and Monitoring**:
    - **Traffic Logs**: Enabled logging for the firewall rules to monitor and analyze incoming and outgoing traffic patterns. This helps in identifying any suspicious activities or attempts to breach security.

Load balancing is crucial for distributing incoming traffic across multiple instances of the application, enhancing performance and reliability. In this project, Google Cloud Load Balancer was utilized for the following reasons:

1. **High Availability**:
    - **Instance Group Management**: By creating a managed instance group, the application could automatically scale based on traffic demands. This ensures that the application remains available even during traffic spikes.
2. **Traffic Distribution**:
    - **Global Load Balancing**: The global load balancer was configured to distribute incoming traffic intelligently across multiple backend instances, optimizing response times and minimizing latency for users.
3. **Health Checks**:
    - **Automated Health Checks**: Load balancer health checks were implemented to monitor the health of the backend instances continuously. If an instance becomes unresponsive or fails, the load balancer automatically reroutes traffic to healthy instances, ensuring uninterrupted service.
4. **SSL Termination**:
    - **Secure Connections**: The load balancer was set up to handle SSL termination, offloading the SSL decryption from the backend instances. This improves performance by reducing the processing burden on the application instances while ensuring secure data transmission.
5. **Simplified Management**:
    - **Single Endpoint**: With load balancing, users can access the application through a single external IP address, simplifying access management and enhancing user experience.

By implementing robust firewall rules and effective load balancing, the networking security of the file-sharing application was significantly enhanced. These measures not only secured the

application against unauthorized access and potential threats but also optimized its performance and reliability, ensuring a seamless user experience. This comprehensive approach to networking security is vital for any application that aims to protect sensitive data while providing high availability and scalability.

9. Identity and Security Management

Setting up access permissions for the file uploaded in Image 29

```
balzhan_cloudcomputing@cloudshell:~ (cloud-computing-436714)$ gcloud storage objects update gs://balzhan/fb51dfb7fdcd52a2db0af8bd312358fc.jpg --predefined-acl=private
Patching gs://balzhan/fb51dfb7fdcd52a2db0af8bd312358fc.jpg...
  Completed 1
```

Image 29: Setting up access permissions to the file

Setting up access permissions to whole bucket in Image 30

```
balzhan_cloudcomputing@cloudshell:~ (cloud-computing-436714)$ gcloud storage buckets update gs:
//balzhan --predefined-default-object-acl=projectPrivate
Updating gs://balzhan/...
  Completed 1
```

Image 30: Setting up access permissions to the bucket

Verifying access permissions of the bucket in Image 31 and Image 32

```
balzhan_cloudcomputing@cloudshell:~ (cloud-computing-436714)$ gcloud storage objects describe g
s://balzhan/d1fbd957bc6f3d7ff30946aae2f2dc4e.jpg --format="default(acl)"
acl:
- entity: project-owners-893573240319
  projectTeam:
    projectNumber: '893573240319'
    team: owners
  role: OWNER
- entity: project-editors-893573240319
  projectTeam:
    projectNumber: '893573240319'
    team: editors
  role: OWNER
- entity: project-viewers-893573240319
  projectTeam:
    projectNumber: '893573240319'
    team: viewers
  role: READER
- email: balzhan.cloudcomputing@gmail.com
  entity: user-balzhan.cloudcomputing@gmail.com
  role: OWNER
```

Image 31: Permissions of the bucket

```
balzhan_cloudcomputing@cloudshell:~ (cloud-computing-436714)$ gcloud storage buckets update gs:
//balzhan --predefined-default-object-acl=publicRead
Updating gs://balzhan/...
  Completed 1
```

Image 32: Setting public access to read

Renaming and copying uploaded file in Image 33

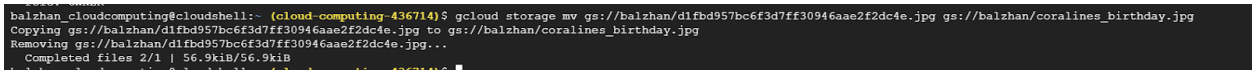


Image 34: Renaming and copying uploaded file

Deleting the file in Image 35

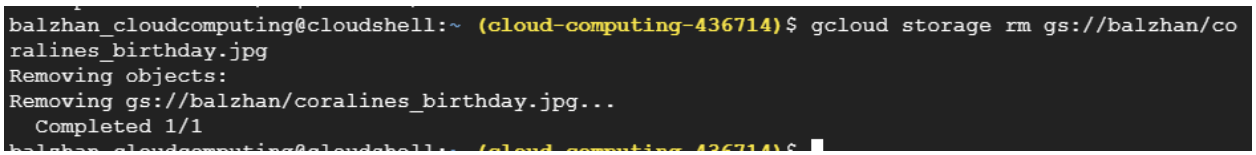


Image 35: Delete file

Identity and Access Management (IAM) plays a crucial role in ensuring the security of cloud-based applications by defining who can access resources and what actions they can perform. In the development of the file-sharing application deployed on Google Cloud Platform (GCP), a structured IAM framework was implemented to manage user roles and permissions effectively.

To establish a robust access control system, we employed Role-Based Access Control (RBAC), which involves creating specific user roles that determine access levels based on the principle of least privilege. This means users are granted only the permissions necessary to perform their tasks. For instance, we defined distinct roles such as "Administrator" and "User." Administrators were given full access to manage all resources, including configuration settings and user management, while regular users were limited to file operations such as uploading, downloading, renaming, and deleting their files.

In addition to predefined roles, we crafted custom roles tailored to meet specific application requirements. This flexibility allowed us to fine-tune permissions and ensure that users had access only to the data and functionalities relevant to their roles. IAM policies were then assigned to resources, specifying what actions users with each role could perform. This granular approach to permission management not only enhances security but also simplifies access control by providing clear guidelines on who can do what.

Another significant aspect of our IAM implementation was the use of service accounts. We created dedicated service accounts for the application to securely interact with Google Cloud resources. These service accounts were granted the minimum required permissions, adhering to the principle of least privilege. By separating the duties of different components within the application, we reduced the risk of security breaches affecting multiple areas.

To maintain accountability and transparency, we enabled audit logging for IAM activities. This feature allows us to track changes made to IAM policies and monitor user access to resources.

By having detailed logs of who accessed what and when, we can quickly identify any suspicious activities or unauthorized changes, thereby strengthening our security posture.

Alongside IAM, several robust security measures were implemented to ensure data protection within the file-sharing application. One of the primary strategies employed was data encryption. To safeguard files stored in Google Cloud Storage, we utilized automatic encryption at rest, ensuring that data is rendered unreadable to unauthorized users. This encryption is managed by Google, providing an additional layer of security without requiring extensive manual configuration.

In addition to encryption at rest, we enforced encryption in transit by using SSL/TLS protocols for all data transmitted between clients and the server. This practice protects sensitive information from interception during transmission, significantly reducing the risk of unauthorized access.

Access control policies were meticulously crafted to enforce strict access guidelines. By applying IAM policies, we ensured that only authorized users could access sensitive resources, adhering to the principle of least privilege. To further enhance security, we implemented Multi-Factor Authentication (MFA) for users accessing the application. This additional verification step beyond the standard password requirement greatly diminishes the risk of unauthorized access, as it requires users to authenticate through a secondary method.

To ensure the resilience of data, we established regular automated backups of files and application data. This proactive measure guarantees that in the event of accidental deletion or data corruption, we have the means to restore the system to its previous state. These backups were also encrypted, reinforcing data security.

Finally, we recognized the importance of ongoing vigilance in maintaining a secure environment. Regular security audits and vulnerability scans were conducted to identify and address potential weaknesses within the application. This proactive approach ensures that the application remains resilient against emerging threats and continues to uphold its security standards.

By leveraging GCP's IAM capabilities and adopting strong data protection strategies, we have not only safeguarded sensitive user information but also built trust among users, reinforcing the application's integrity and reliability. This comprehensive approach to security is essential for any cloud-based application in today's data-driven landscape, where protecting user data is paramount.

10. Testing and Quality Assurance

In this project, various testing methodologies were employed to ensure the application’s functionality, performance, and reliability.

JUnit was used extensively to perform unit testing on the backend application. This framework allowed for the systematic testing of individual components, ensuring that each piece of functionality operated as expected. Tests were written for critical services and controllers, validating both the logic and the expected outputs. By isolating components, we could identify issues early in the development process, which ultimately facilitated a smoother integration phase.

In addition to unit tests, integration tests were conducted to verify that different components of the application interacted correctly with one another. These tests helped ensure that the application's services could communicate with the database and Google Cloud Storage effectively. Integration testing played a vital role in identifying issues related to data flow between components, ensuring that the overall architecture functioned as intended.

Manual testing was also performed to validate connectivity and user interactions within the application. This included testing the file upload and download functionalities, as well as the rename and delete operations. Manual testing allowed for real-time assessment of the user experience and provided insights that automated tests might overlook, such as usability and performance under realistic conditions.

While load testing was not explicitly mentioned in the initial phases, future iterations of the project would benefit from including this methodology to evaluate performance under stress. Load testing could help identify potential bottlenecks and ensure the application can handle a high number of simultaneous users, particularly in a production environment.

Overall, the combination of unit tests, integration tests, and manual testing provided a robust framework for validating the application's functionality, ensuring a reliable and user-friendly experience. As the project evolves, implementing load testing will be an essential addition to further enhance performance evaluation.

11. Monitoring and Maintenance

Google Cloud Monitoring is a fully managed monitoring service that provides real-time insights into application performance, latency, and errors. It offers metrics, logs, and traces to help identify issues. For example, you can use Google Cloud Monitoring to track the response time of your application, identify slow queries, and detect errors. Google Cloud Logging is a logging service that collects, stores, and analyzes logs from applications, services, and infrastructure. It helps identify issues, troubleshoot problems, and monitor application performance. For example, you can use Google Cloud Logging to track errors, warnings, and informational messages from your application, and use this data to identify trends and patterns. Google Cloud Debugger is a debugging tool that allows developers to debug applications in real-time, without stopping or restarting the application. For example, you can use Google Cloud Debugger to step through your code, inspect variables, and set breakpoints. Stackdriver is a monitoring and logging

platform that provides real-time insights into application performance, latency, and errors. It offers metrics, logs, and traces to help identify issues. For example, you can use Stackdriver to track the performance of your application, identify bottlenecks, and detect errors.

Continuous Monitoring: Monitor applications and services in real-time to detect issues early and prevent downtime. For example, you can use Google Cloud Monitoring to track the performance of your application and detect issues before they become critical. Automated Alerting: Set up automated alerts for critical issues, such as high latency, errors, or resource utilization. For example, you can use Google Cloud Monitoring to set up alerts for high latency, and receive notifications when your application is experiencing slow performance. Root Cause Analysis: Use monitoring data to identify the root cause of issues and take corrective action. For example, you can use Google Cloud Logging to identify the root cause of an error, and use this data to take corrective action. Proactive Maintenance: Schedule regular maintenance tasks, such as backups, updates, and scaling, to ensure application health and performance. For example, you can use Google Cloud Scheduler to schedule regular backups of your application data. Collaboration: Encourage collaboration between development, operations, and quality assurance teams to ensure a unified approach to monitoring and maintenance. For example, you can use Google Cloud Console to collaborate with your team and share monitoring data. Documentation: Maintain accurate documentation of monitoring configurations, alerting rules, and maintenance procedures. For example, you can use Google Cloud Documentation to maintain accurate documentation of your monitoring configurations. Testing and Validation: Regularly test and validate monitoring configurations and alerting rules. For example, you can use Google Cloud Testing to test and validate your monitoring configurations. Scalability and High Availability: Design applications and services to scale horizontally and provide high availability. For example, you can use Google Cloud Load Balancing to distribute traffic across multiple instances of your application. Security and Compliance: Implement security and compliance measures, such as encryption, access controls, and auditing, to ensure the integrity and confidentiality of data. For example, you can use Google Cloud Security to implement encryption and access controls for your application data. Continuous Improvement: Continuously monitor and evaluate monitoring tools and strategies. For example, you can use Google Cloud Monitoring to continuously monitor and evaluate the performance of your application.

Use a centralized monitoring platform to collect and analyze data from multiple sources. For example, you can use Google Cloud Monitoring to collect and analyze data from multiple sources, such as Google Cloud Logging and Google Cloud Debugger. Set clear alerting rules to ensure timely notification of critical issues. For example, you can use Google Cloud Monitoring to set up alerts for high latency, and receive notifications when your application is experiencing slow performance. Use automated scripts to perform routine maintenance tasks, such as backups and updates. For example, you can use Google Cloud Scheduler to schedule regular backups of your application data. Monitor for anomalies and unusual patterns in application performance and behavior. For example, you can use Google Cloud Monitoring to monitor for anomalies and

unusual patterns in your application performance. Involve the development team in monitoring and maintenance activities to ensure a unified approach to application health and performance. For example, you can use Google Cloud Console to collaborate with your development team and share monitoring data.

12. Challenges and Solutions

During the project, I encountered several significant challenges, particularly regarding the deployment and configuration of the application on a Google Cloud VM instance. One of the most pressing issues was that, while the JAR file built successfully and ran locally without problems, it failed to execute on the VM due to its inability to locate the necessary credentials file. The application was searching for the credentials in a default location that did not exist on the VM, leading to an error during startup.

Initially, I relied on credentials generated within the Google Cloud Console, which were stored in a cache on my local machine. This approach, however, proved inadequate for the VM environment. To resolve this issue, I installed the Google Cloud SDK on my local machine, enabling me to generate the required credentials file directly. Once I obtained the credentials, I ensured they were correctly referenced in the application configuration. This adjustment allowed the application to access the Google Cloud Storage bucket successfully, resolving the issue.

Another challenge I faced was securing the Virtual Private Cloud (VPC) network. Configuring firewall rules and load balancing to optimize security while maintaining performance required a deep understanding of Google Cloud's networking capabilities. I had to ensure that only authorized traffic could access the application while also providing a seamless experience for users. This involved extensive testing and adjustments to the firewall settings to strike the right balance between security and accessibility.

Additionally, I noticed that file uploads to the Google Cloud Storage bucket were occurring much slower than expected. This performance issue was frustrating, as it impacted user experience. After conducting some performance monitoring and analyzing the network traffic, I realized that the VPC settings and routing configurations could be improved. I made adjustments to optimize the network routes and ensure that the application was properly utilizing the available bandwidth. Implementing these changes resulted in significantly faster file upload speeds, enhancing the overall efficiency of the application.

By addressing these challenges, I was able to successfully deploy the application in the cloud environment, ensuring it functioned optimally while maintaining the necessary security measures. The experience provided valuable insights into cloud deployment and network management, furthering my understanding of these critical aspects of modern application development.

13. Conclusion

Reflecting on the project outcomes, I am pleased with the successful deployment of the file-sharing application on Google Cloud, which demonstrated the effectiveness of the technologies utilized, including Java, Spring, and Thymeleaf. The integration with Google Cloud Storage worked seamlessly after resolving initial credential issues, allowing for secure file uploads, downloads, and management. The implementation of firewall rules and network security measures ensured that the application remained secure while being accessible to users.

However, as with any project, there are areas for improvement and enhancement. One significant suggestion for future iterations of this project would be the incorporation of Docker and Kubernetes. Utilizing Docker would facilitate easier packaging and deployment of the application, creating a more consistent environment across local and production setups. Meanwhile, Kubernetes could provide robust orchestration capabilities, enabling automatic scaling, self-healing, and easier management of application deployments.

Additionally, considering the security aspect, implementing automatic pod enhancements could further fortify the application during deployment. Such measures could provide real-time monitoring and responsive security actions, reducing the likelihood of vulnerabilities. However, it's essential to note that these enhancements could increase costs, which might be a concern for a student project budget.

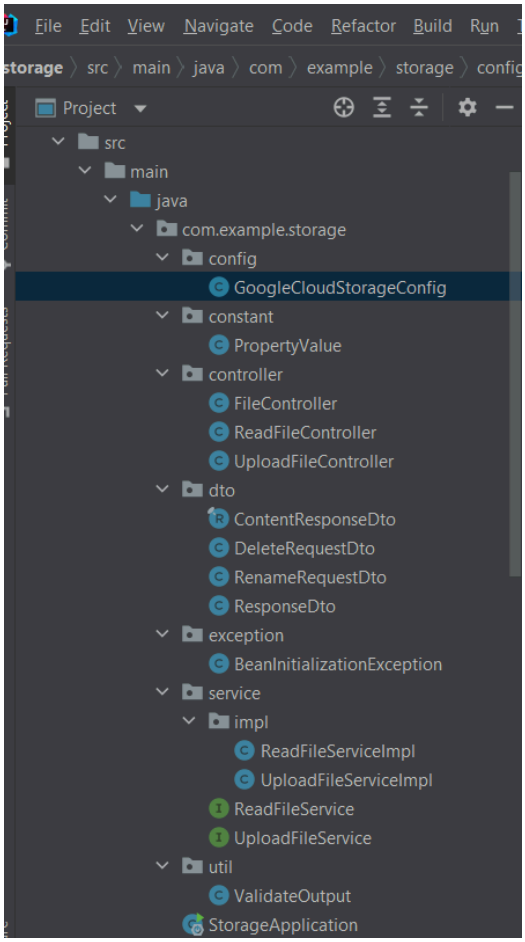
In summary, while the project achieved its primary objectives and provided valuable learning experiences, incorporating Docker, Kubernetes, and enhanced security measures in future endeavors would not only improve deployment processes but also bolster the application's resilience and scalability.

14. References

- Quickstart: View latency of app requests | Cloud Trace | Google Cloud
- Create VMs with multiple network interfaces | VPC | Google Cloud
- Create buckets | Cloud Storage | Google Cloud
- Upload objects from a file system | Cloud Storage | Google Cloud
- Object Lifecycle Management | Cloud Storage | Google Cloud
- Object Lifecycle Management | Cloud Storage | Google Cloud
- Identity and Access Management | Cloud Storage | Google Cloud

15. Appendices

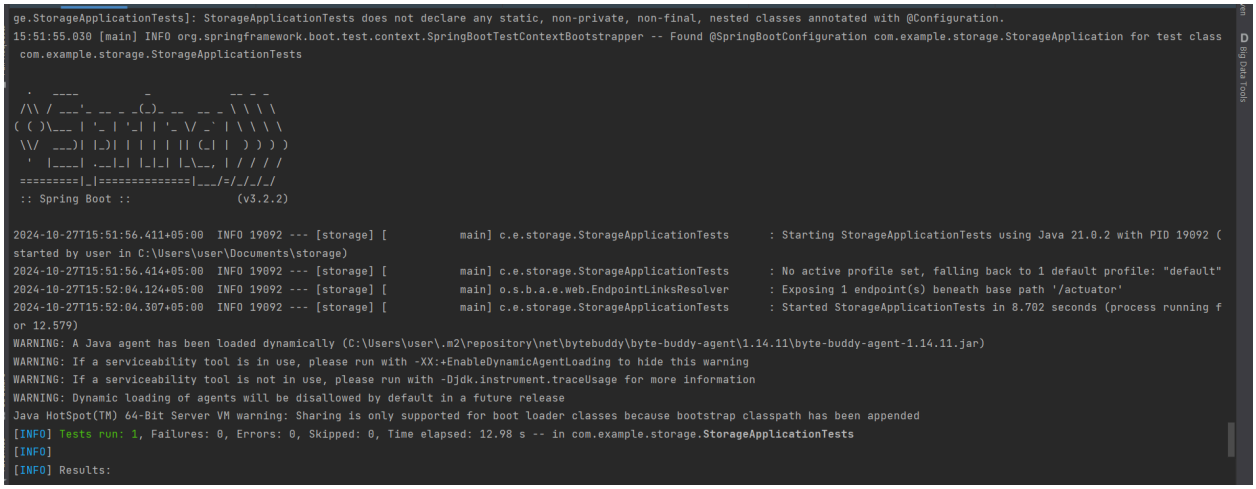
1. Project structure of the backend



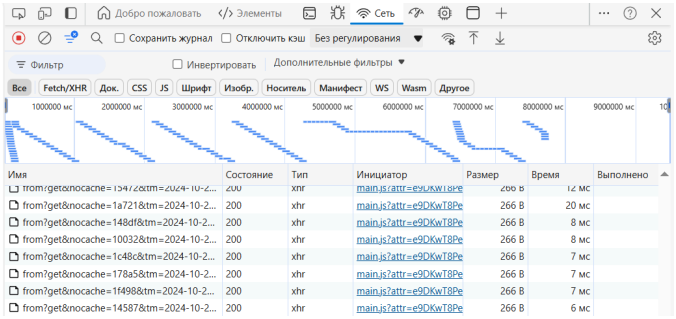
2. The google storage configuration bean

```java
@Configuration
@RequiredArgsConstructor
@Slf4j
public class GoogleCloudStorageConfig {

  private final PropertyValue propertyValue;
  private final static String fileCredentialsLocation =
      "application_default_credentials.json";

  @Bean
  public Storage googleCloudStorage() {
    try {
      Credentials credentials = GoogleCredentials.fromStream(new FileInputStream(fileCredentialsLocation));
      return StorageOptions.newBuilder().setCredentials(credentials).build().getService();
    } catch (IOException e) {
      log.error("File not found with credentials in location {}", e.getMessage());
      throw new BeanInitializationException("File not found with credentials in location");
    }
  }
}
```

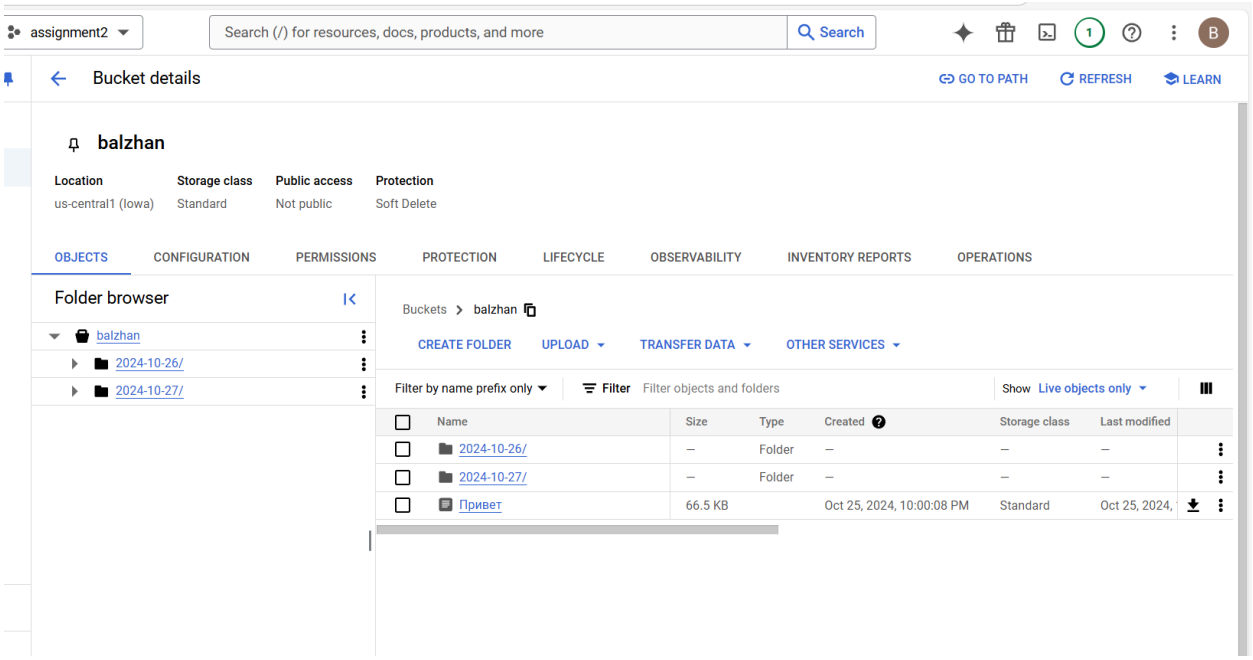
3. Successful run of the application



4. Frontend of the application



5. Bucket structure organisation



6. Github private repositories ssh git@github.com:Baljibeka/storage.git