



**KAZAKH-BRITISH
TECHNICAL
UNIVERSITY**

Assignment №4 Report

**Application Security Best Practices and Scaling Applications on
Google Cloud**

Done by:

Amen Azat, 21B030774

21.11.2024

Almaty, 2024

1. Executive Summary

The purpose of Assignment №4 is to familiarize with the best practices for ensuring application security and scaling applications in Google Cloud.

Here we have implemented methods to ensure the security of our resources using Cloud KMS, IAM, Load Balancing, Monitoring, Alerting and configured scalability parameters for my To-Do application using Google Kubernetes Engine.

In this report, I described in detail how I took every step to ensure security and scalability for cloud application.

2. Table of Contents:

- Introduction
 - Exercise №1. Application Security Best Practices
 - Exercise №2. Scaling Applications on Google Cloud
 - Conclusion
 - Recommendations
 - References
 - Appendices
-

3. Introduction

Google Cloud Platform is one of the best cloud platforms on a par with platforms such as Amazon Web Services (AWS), Microsoft Azure. The main advantage of such platforms is scalability, flexibility and fault tolerance. Also, an important advantage of such cloud platforms is a wide range of ready-made services and tools, which significantly saves development time and resources. The advantage of GCP is that it can be used to create scalable, flexible, highly available and fault-tolerant applications.

The Google Cloud Platform architecture includes a large number of servers around the world. GCP has a huge number of services and technologies for different types of tasks. GCP includes services like Google Kubernetes Engine for containerizing applications, Compute Engine for creating virtual machines, App Engine for deploying web applications, Cloud Function for creating serverless functions, Cloud Endpoints for API management, Various cloud storages such as Cloud SQL, Cloud Storage, Firestore for data storage and many other useful services and tools.

One of the key factors in the operation of applications is their **Security and Scalability**. They store very large amounts of data. And the theft of this confidential data can lead to big financial problems. Google Cloud not only makes it possible to create cloud applications, but also provides many ways to ensure their security and scalability:

- Data encryption via Cloud KMS.
- To control access to the application, there is an IAM for assigning roles for accessing resources.
- Google Kubernetes Engine services that makes easy to reach scalability.
- Monitoring tools for tracking the operation of applications.
- Notification tools where you can set up alerts for certain situations.
- And much more.

The purpose of the report is to introduce the best practices of application security and application scaling in Google Cloud. Here we have implemented methods to ensure the security of our resources using cloud CMS to create encryption keys, IAM for role allocation, and a load balancer for working over HTTPS connections. We also set up scalability parameters for my To-Do application using the Google Kubernetes engine, used auto and horizontal/vertical scaling. We have implemented monitoring and notification tools to track the operation of the application. We also took care of optimizing performance and costs.

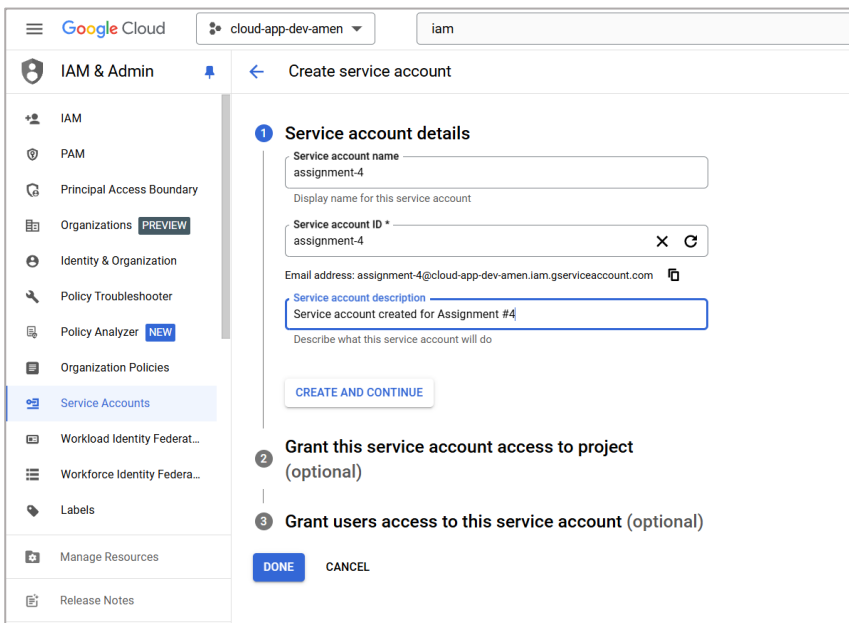
4. Exercise №1 Application Security Best Practices

1) Overview of Cloud Security

During this task, we will use methods to ensure security. First of all, we will create an IAM service for distributing roles according to the principle of least privilege. We will encrypt the data in the cloud storage using key encryption in Google key management. We will configure traffic for our application using a load balancer and install secure HTTPS protocol using an SSL certificate to ensure the encryption of our application on the network.


2) IAM Configuration

Creating of service account for IAM configuration.



Next give principle of least privilege. The principle of least privilege means that every user will have the minimum access that needed only for his/her responsibilities. For example:

- User can only View Cloud SQL databases
- View Storage
- Encrypt data using Cloud KMS

	assignment-4@cloud-app-dev-amen.iam.gserviceaccount.com	assignment-4	Cloud KMS CryptoKey Encrypter	
			Cloud SQL Viewer	
			Editor	
			Environment and Storage Object User	

3) Data Protection

We will use **Google Cloud KMS** to encrypt our resources:

1. Firstly, we need to create key ring where we will collect our encryption keys:

Google Cloud | cloud-app-dev-amen | Search (/) for resources, docs, products, and more

Security

Security Command Center

- Risk Overview
- Threats
- Vulnerabilities
- Compliance
- Assets
- Findings
- Sources
- Posture Management ...

Detections and Controls

- Google SecOps

Create key ring

Key rings group keys together to keep them organized. In the next step, you'll create keys that are in this key ring. [Learn more](#)

Project name
cloud-app-dev-amen

Key ring name *
assignment-4-key-ring

Location type

☒ **Region**
Lower latency within a single region

☐ **Multi-region**
Highest availability across largest area

Region *
us-central1 (Iowa)

CREATE **CANCEL**

<input type="checkbox"/>	Name ? ↑	Location	Keys ?
<input type="checkbox"/>	assignment-4-key-ring	us-central1	assignment-4-encryption-key

2. After that, create encryption key in this key ring:

Google Cloud | cloud-app-dev-amen | Search (/) for resources, docs, products, and more

Security

Security Command Center

- Risk Overview
- Threats
- Vulnerabilities
- Compliance
- Assets
- Findings
- Sources
- Posture Management ...

Detections and Controls

- Google SecOps
- reCAPTCHA

Marketplace

Release Notes

Create key

A cryptographic key is a resource that is used for encrypting and decrypting data or for producing and verifying digital signatures. A key can have multiple versions. [Learn more](#)

Name and protection level

Key name *
assignment-4-encryption-key

Protection Level

☒ **Software**
Cryptographic operations are performed on software

☐ **HSM**
Cryptographic operations are performed on a Hardware Security Module (HSM)

☐ **External**
Cryptographic operations are performed using a key stored in an external key manager. [Learn more](#)

CONTINUE

Key material

Key material: Generated

Purpose and algorithm

Purpose: Symmetric encrypt/decrypt
Algorithm: Google symmetric key

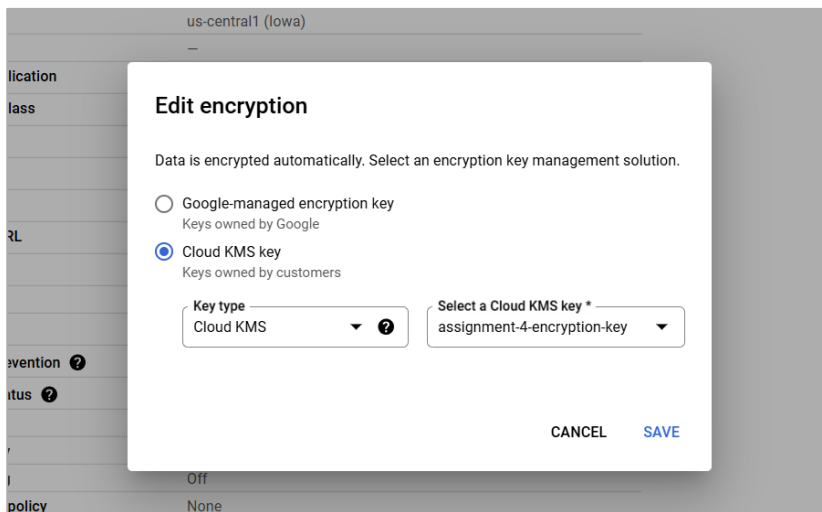
CREATE **CANCEL**

Name ↑	Status ?	Protection level ?	Purpose ?
assignment-4-encryption-key	✓ Available	Software	Symmetric encrypt/decrypt

So, we successfully created our encryption key called “assignment-4-encryption-key”. Lets use it to encrypt our cloud storage bucket:

<input type="checkbox"/>	ass-4-bucket	Nov 19, 2024, 8:11:10 PM	Region	us-central1	Standard
--------------------------	------------------------------	--------------------------	--------	-------------	----------

1. Open setting of your bucket and edit encryption. Select Cloud KMS key and assign key that you created.



2. How you can see, we successfully encrypt our cloud storage bucket. So our data will be safe and secure.

Cloud Console URL	https://console.cloud.google.com/storage/browser/ass-4-bucket
gsutil URI	gs://ass-4-bucket
Permissions	
Access control	Fine-grained
Public access prevention	Not enabled by org policy or bucket setting
Public access status	Subject to object ACLs
Protection	
Soft delete policy	7 days
Object versioning	Off
Bucket retention policy	None
Object retention	Disabled
Encryption type	Customer-managed
Default encryption key	cloud-app-dev-amen/us-central1/assignment-4-key-ring/assignment-4-encryption-key
Object lifecycle	

Next is, **use HTTPS for data in transit and configure load balancers to enforce SSL:**

1. The first step is to create SSL certificate to ensure our web app protection in network

← Create certificate

Certificate Name

Name *

ass-4-ssl

?

Lowercase, no spaces.

Description (optional)

Description

Location

Certificate can be used globally or regionally.

☒ Global

☐ Regional

Scope

Key distribution scope defines on which data center cert will be deployed.

Default

▼

Certificate type
You can have Google issue and manage the certificate or you can upload a certificate issued by a third-party.

☐ Create Self-managed certificate

☒ Create Google-managed certificate

Certificate Authority type
You can have Google Issue and manage the certificate or you can upload a certificate issued by a third-party

☒ Public

☐ Private

Domain names *
Use the input box to specify your domain name(s), use comma to separate different ones.

Authorization type

☐ DNS Authorization

☒ Load Balancer Authorization

Labels

[+ ADD LABEL](#)

[CREATE](#) [CANCEL](#)

- Next, let's create a load balancer in Network Services/Load Balancing for To-do application on GKE:

Network Services

← Create a load balancer

1 Type of load balancer

☒ Application Load Balancer (HTTP/HTTPS)

Choose an Application Load Balancer when you need a flexible feature set for your applications with HTTP and HTTPS traffic.

[NEXT](#)

Load balancing

- Cloud DNS
- Cloud CDN
- Cloud NAT
- Cloud Service Mesh (Traffic ...)
- Service Directory
- Cloud Domains
- Private Service Connect
- SSL policies
- Service Extensions

- Configure Frontend part:

New Frontend IP and port

Name
load-balancer-frontend

Lowercase, no spaces.

Description

Protocol
HTTPS (includes HTTP/2 and HTTP/3)

Select HTTPS to support clients that support HTTP/2. The load balancer automatically offers HTTP/2 as part of the TLS handshake.

Network Service Tier
Premium

Global HTTP(S) load balancing only supports the Premium Network Service tier.
[Learn more](#)

IP version
IPv4

IP address
Ephemeral

Port *
443

Application load balancing supports all TCP ports. [Learn more](#)

Certificate *
ass4-ssl

Here I used created SSL certificate for HTTPS and data encryption.

4. Configure Backend part:

Backend configuration

Create or select a backend service for incoming traffic. You can add multiple backend services and back

Backend services & backend buckets
load-balancer-backend

✓ CROSS-PROJECT BACKEND SERVICES

Backend services

Name	Region	Instance groups/Network endpoint groups
load-balancer-backend	us-central1	1 instance group

Here I used my To-do on GKE.

So, we created and configured load balancer for network security.

4) Security Testing

In this task we need to implement security scanning tool in the CI/CD pipeline. So, for this I will use github actions with OWASP ZAP tool. When we will do commits it will be analyze and generate a report for us.

For that we need to create configuration file:

```

1  name: OWASP_ZAP_Scan
2
3  on:
4  push:
5    branches:
6      - master
7
8  jobs:
9    zap_scan:
10     runs-on: ubuntu-latest
11
12     steps:
13       - name: Checkout code
14         uses: actions/checkout@v2
15
16       - name: Set up OWASP ZAP
17         uses: zaproxy/action-full-scan@v0.1.0
18         with:
19           target: "https://cloud-app-dev-amen.uc.r.appspot.com"
20           cmd_options: "-r report.html"
21
22       - name: Upload OWASP ZAP Report
23         uses: actions/upload-artifact@v3
24         with:
25           name: zap-report
26           path: report.html

```

So, when you do commit in your github actions will be created OWASP ZAP action. And it will generate worloads, them will looks like that, and you need to download report file.



After you download it, you need check alerts and analyze them, detect problems and solve it.

Risk Level	Number of Alerts
High	0
Medium	2
Low	3
Informational	1
False Positives:	0

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	4
Missing Anti-clickjacking Header	Medium	2
Permissions Policy Header Not Set	Low	4
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	4
X-Content-Type-Options Header Missing	Low	2
Storable and Cacheable Content	Informational	4

For example, let's analyze some of the list:

- The absence of a Content Security Policy (CSP) represents a medium risk:**
You need to add the Content-Security-Policy header. You need to specify reliable sources for scripts and images to limit the loading of unsafe resources.
- The absence of a header that protects against click interception is a medium risk:**
There is no protection against click-interception attacks, when attackers can embed an invisible iframe on our site. To solve the problem, you need to use the X-Frame-Options header with the value DENY/SAMEORIGIN to limit the ability to load your site into an IFRAME from third-party resources.
- There is no header access policy, which is a low risk:**
This header controls the browser's access to various functions, such as the camera and microphone.
- The server version is displayed in the "Server" header, low risk:**
The server discloses information about the server version, which can help attackers identify vulnerabilities related to a specific software version.

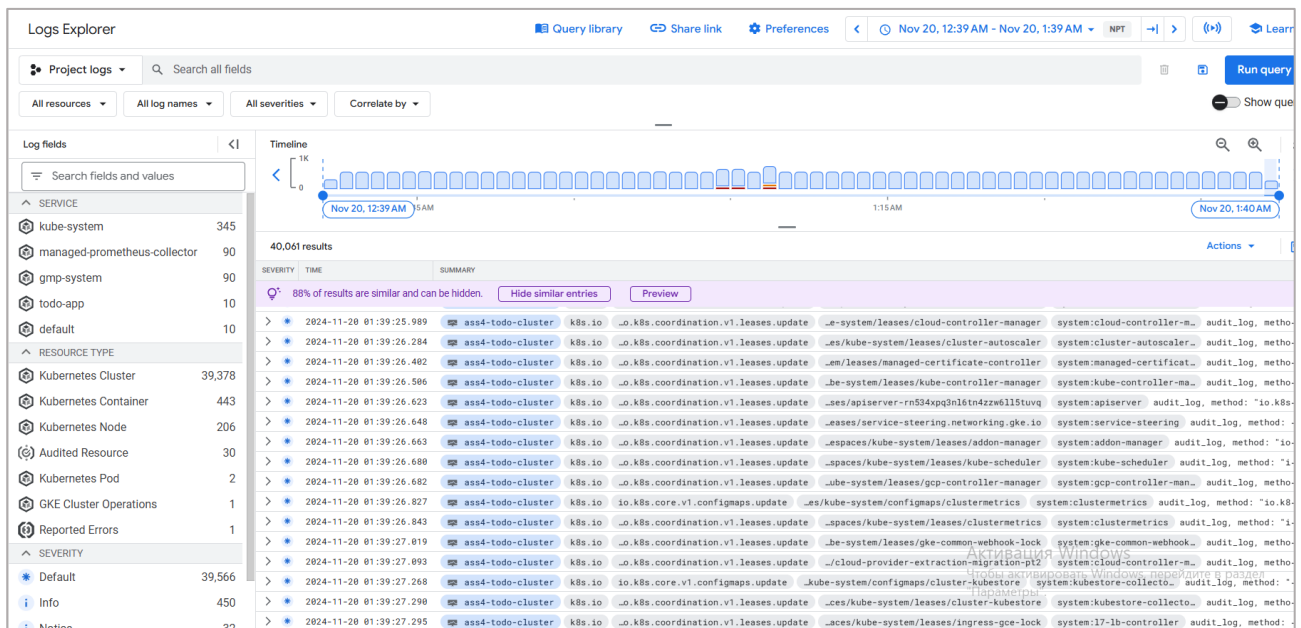
Checking the security in the CI/CD pipeline is an important step before deploying an application in a production environment. The generated report is an important document that we must fully analyze. Because if we don't do it, it can lead to big problems like information leakage, money loss and others. After fixing all the vulnerabilities, we have to restart the check to make sure that our application is safe. Then we can deploy our application in a production environment.

5) Monitoring and Logging

At this step, I will configure cloud monitoring and alerting for my To-Do application.

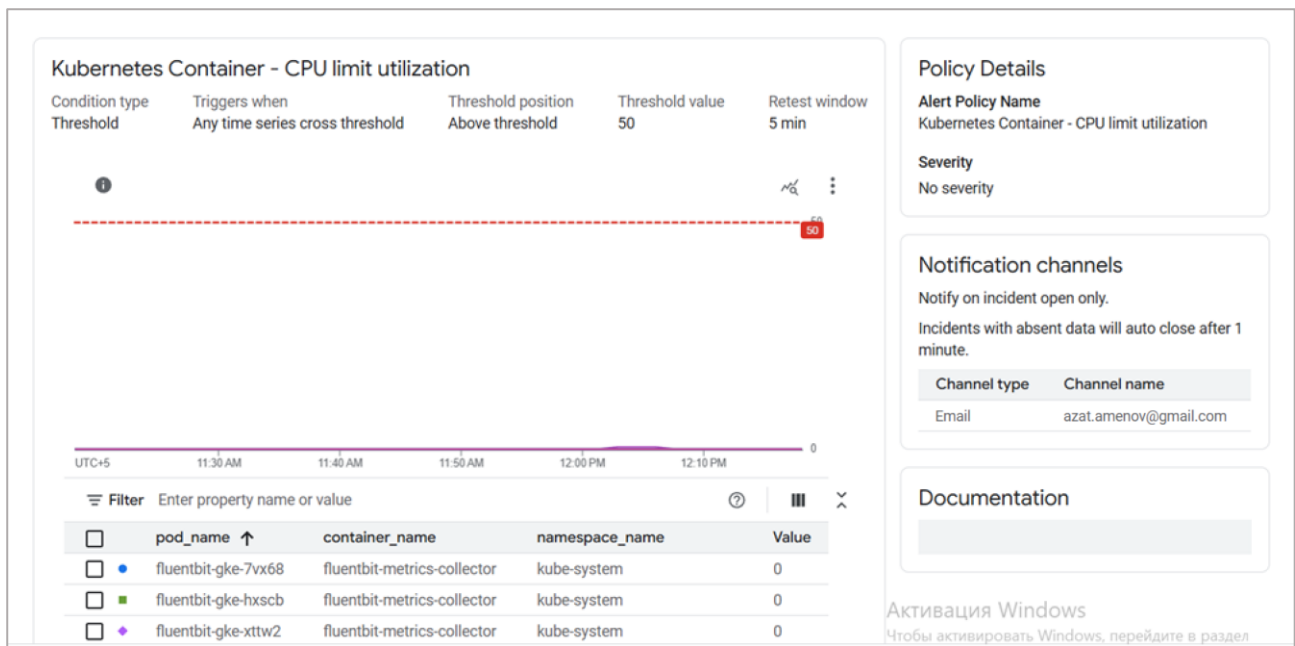
Enable Google Cloud Audit Logs:

At this step, we enable Audit Logs. To do this you need to enable audit logs for your cloud application, and after go to the Logs Explorer. How you can see, everything is working well. Logs related to To-Do application on GKE are listing.



Set up Alerts using Google Cloud Monitoring:

So, at this step I created Alert Policy in Cloud Monitoring. The purpose of the Alerting is to send a notification to the channels you added if any of the situations you described (For example, system is overloaded) exist. It gives us opportunity to quickly fix errors.



This is my Alert Police – “Kuberneted Container - CPU limit utilization”. The alert will send if the CPU percent of any of containers in Kubernetes Cluster will exceed 50%. And alerts will send to azat.amenov@gmail.com channel by email.

6) Incident Reponse

For example, our goal is to minimize the damage from some incident in the Google Cloud and restore the application. Stages of incident response:

1. Detection: Fixing an incident through monitoring and notification tools.
2. Stop the spread of the threat. For example, disable a vulnerable account or service.
3. Elimination of all errors.
4. Data recovery using, for example, backup.
5. Analysis: Documentation of the incident and implementation of measures to prevent further recurrence.

Let's do simulation of an incident:

1. Notifications about strange activities in the storage have been received through monitoring/notification tools.
2. Disabling the account from which there was activity.
3. Update the IAM configuration.
4. Data recovery via backup.
5. Documentation of the incident.

5. Exercise №2 Scaling Application on Google Cloud

1) Overview of Scalability

The scalability of applications running in cloud or Kubernetes clusters ensures efficient workload management. This allows applications to cope with increased load during peak periods and, conversely, reduce resource usage when demand decreases. Thanks to the approaches of cloud platforms such as GCP, businesses pay only for those resources that are necessary for the operation of the application: during normal times, resources are maintained at an average level for stable operation, during peak periods, expansion is provided to handle high loads, and during periods of

low activity costs are minimized, excluding overpayments. I will use my To-do list application and show how to reach scalability for it.

2) Application Design

- **Description of the web application created and the architecture used:**

For this assignment I will use To-Do List web application written on Python Flask. It's simple web application that allows to users create and manage tasks.

- **Architecture of my project:**

First, my project uses the REST methodology and implements a Restful API, where each endpoint is responsible for some action on the To-Do List data, data in JSON format.

Functionality of my app:

- **CREATE:** Creating a task, endpoint: /add
- **DELETE:** Completing the task, endpoint: /delete/<todo_id>
- **UPDATE:** Update the task, endpoint: /update/<todo_id>
- **GET:** View all tasks, endpoint: /

Task structure:

```
todo = { # Task in JSON format
    "id": str(uuid.uuid4()), # unique ID, generates using uuid library
    "task": request.get_json()['task'] # task text with string type
}
```

Second, I use Kubernetes Cluster provided by Google Cloud Kubernetes Engine (GKE). He is responsible for the deployment of my application and for its scaling and availability.

I use following recourses: **Pods** with my to-do app container, **Deployment** for configuration and **Service** that provides **Load Balancer**.

<input type="checkbox"/> Status	Name ↑	Location	Tier ?	Number of nodes	Total vCPUs
<input type="checkbox"/>	ass4-todo-cluster	us-central1	Standard	3	6

<input type="checkbox"/>	Name ↑	Status	Type	Pods	Namespace	Cluster
<input type="checkbox"/>	todo-app	OK	Deployment	1/1	default	ass4-todo-cluster

Exposing services ?		
Name ↑	Type	Endpoints
todo-service	Load balancer	34.56.17.107:80

Assignment 4 > deployment.yaml

```

3 metadata:
4   name: todo-app
5 spec:
6   replicas: 2
7   selector:
8     matchLabels:
9       app: todo-app
10  template:
11    metadata:
12      labels:
13        app: todo-app
14    spec:
15      containers:
16      - name: todo-app
17        image: gcr.io/cloud-app-dev-amen/todo-app:v1
18        ports:
19        - containerPort: 8080
20      resources:
21        requests:
22          cpu: "100m"
23          memory: "100Mi"
24        limits:
25          cpu: "200m"
26          memory: "200Mi"

```

Assignment 4 > service.yaml

```

1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: todo-service
5 spec:
6   type: LoadBalancer
7   selector:
8     app: todo-app
9   ports:
10  - port: 80
11    targetPort: 8080

```

```

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Assig
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
todo-app-d7d4d54d5-rz741            1/1     Running   0           10m
todo-app-d7d4d54d5-smfxf            1/1     Running   0           39s

```

```

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A
$ kubectl get services
NAME            TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes      ClusterIP     34.118.224.1  <none>         443/TCP          20h
todo-service    LoadBalancer 34.118.229.214 34.56.17.107   80:32283/TCP     19h

```

How you can see, my To-Do List API and Kubernetes Cluster works well:

←

↺

🏠

⚠ Небезопасно

34.56.17.107

```

1 [
2   {
3     "id": "c1a38dca-7001-4eec-a815-d83f5e420756",
4     "task": "hello"
5   }
6 ]

```

3) Scaling Methods

In Kubernetes, you can scale the workload depending on the workload of the application. This allows the cluster to respond more flexibly and effectively to changes in resource demand.

Kubernetes supports manual scaling of workloads. These include horizontal and vertical scaling:

- **Horizontal scaling** is an approach in which we can either increase or decrease the number of replicas managed by the workload.
- **Vertical scaling** is an approach in which we can increase the resources that our replicas consume. For example, the values of CPU and memory.

Horizontal scaling example:

1. By default replica number is **1**. So when we change the replica number from default to another, we increase the power of our application. Let's do it:

```

Assignment 4 > deployment.yaml
3  metadata:
4    name: todo-app
5  spec:
6    replicas: 1
7    selector:

```

```

Assignment 4 > deployment.yaml
3  metadata:
4    name: todo-app
5  spec:
6    replicas: 2
7    selector:

```

In first image our replica number is 1 and in second image we change it from 1 to 2. And after applying changes in `deployment.yaml`. We can run `kubectl get pods` to see result.

```

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Assignment 4
$ kubectl apply -f deployment.yaml
deployment.apps/todo-app configured

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Assignment 4
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
todo-app-d7d4d54d5-rz74l            1/1     Running   0           9m37s
todo-app-d7d4d54d5-smfxf            0/1     ContainerCreating   0           3s

```

And how you can see on next image, number of our pods successfully increased to 2.

```

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Assignment 4
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
todo-app-d7d4d54d5-rz74l            1/1     Running   0           10m
todo-app-d7d4d54d5-smfxf            1/1     Running   0           39s

```

Vertical scaling example:

- I have allocated the basic resources for the module. If we want to scale our application, we need to increase the capacity of the module in which the container is running, so our application can withstand a heavy load. To do this, we will increase all resource limitation indicators, so our application will have more and more resources in stock.

```

resources:
  requests:
    cpu: "100m"
    memory: "100Mi"
  limits:
    cpu: "200m"
    memory: "200Mi"

```

```

resources:
  requests:
    cpu: "100m"
    memory: "100Mi"
  limits:
    cpu: "400m"
    memory: "400Mi"

```

So, next we need apply the changes to deployment:

```

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Assignment 4
$ kubectl apply -f deployment.yaml
deployment.apps/todo-app configured

```

And how you can see, the resources of CPU and memory increased.

OVERVIEW	DETAILS	OBSERVABILITY	REVISION HISTORY	EVENTS	LOGS	APP ERRORS (0)	YAML
129	protocol: TCP						
130	resources:						
131	limits:						
132	cpu: 400m						
133	memory: 400Mi						
134	requests:						
135	cpu: 100m						
136	memory: 100Mi						

And finally, my conclusion is (Comparison):

- The difference between them is that the horizontal scaling can be changed either manually or using the command line interface. And vertical scaling requires only manual changes. This results in horizontal scaling providing zero downtime, while vertical scaling does not. In addition, the first increases the number of pods (replicas), and the second increases the

resources consumed by the pods. Horizontal scaling is better suited for applications that require high availability and scalability, such as web applications with changing traffic. And vertical scaling is better suited for databases that require resources.

4) Load Balancing

Load balancer distributes incoming requests between pods to ensure an even distribution of all resources and high availability of the application.

Load balancer implementaion:

1. First step is to create `service.yaml` with load balancer configuration:

```
Assignment 4 > service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: todo-service
5  spec:
6    type: LoadBalancer
7    selector:
8      app: todo-app
9    ports:
10     - port: 80
11       targetPort: 8080
```

2. Secondly, we need to apply our service to GKE:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Assignment 4
$ kubectl apply -f service.yaml
service/todo-service created
```

3. Then, use `kubectl get service` command to see services. And how you can see our load balancer created

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Assignment 4 (m
$ kubectl get service
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes    ClusterIP     34.118.224.1    <none>           443/TCP          52m
todo-service   LoadBalancer 34.118.229.214  34.56.17.107    80:32283/TCP     12m
```

And finally, use external api to see that application is works:

```
← ↻ 🏠 ⚠ Небезопасно 34.56.17.107
1  [
2  {
3    "id": "c1a38dca-7001-4eec-a815-d83f5e420756",
4    "task": "hello"
5  }
6  ]
```

Health Check:

They check the status of the pods and decide whether the pods is functioning, whether they are ready to accept requests. Let's check it.

We want to make sure that the load balancer redirects the load only to those pods that are considered active and running. To do this, we need to add a Readiness Probe. In GKE, they will check the readiness of the module, and if it is ready, it will allow you to contact our server. This way, we can implement a policy of "Configure health checks so that traffic is directed only to healthy instances." To do this, we will change the `deployment.yaml` file.

1. Add `readinessProbe`:

```

    memory: "400Mi"
    readinessProbe:
      httpGet:
        path: /
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 10

```

2. Next, lets apply changes:

```

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/
$ kubectl apply -f deployment.yaml
deployment.apps/todo-app configured

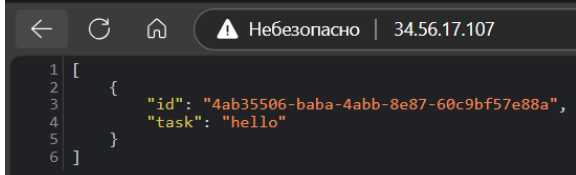
```

3. Everything is works:

```

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Appli
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
todo-app-59f7c87dc7-2r9mt          1/1     Running   0           18s
todo-app-59f7c87dc7-677h8          1/1     Running   0           23s

```



Now, lets check that **Health Check** is really work:

1. First, specify the wrong path in readinessProbe, httpGet. From right “/” path to wrong “/wrong” path:

```

readinessProbe:
  httpGet:
    path: /wrong
    port: 8080
  initialDelaySeconds: 3
  periodSeconds: 10

```

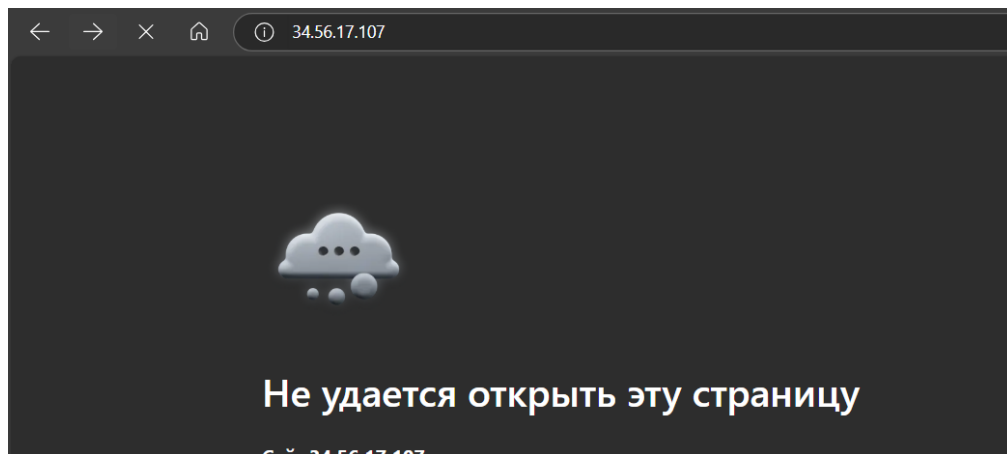
2. Apply changes and check pods:

```

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application D
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
todo-app-59f7c87dc7-2r9mt          0/1     Running   0           71s
todo-app-59f7c87dc7-677h8          0/1     Running   0           71s

```

How you can see, when I trying to access web application by load balancer external address it does not work, because Health Check decide that does not ready.



Let's go back to the way it was now:

1. Return back right path:

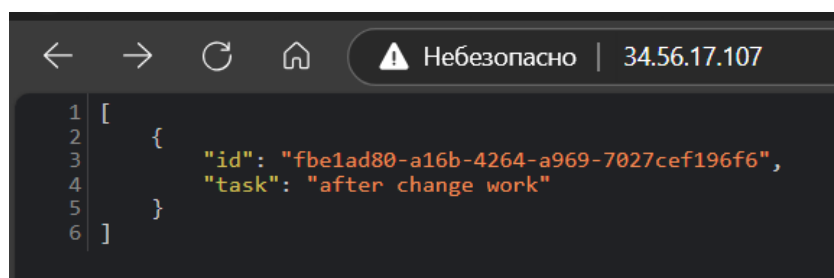
```
readinessProbe:
  httpGet:
    path: /
    port: 8080
```

2. Apply changes to deployment and check pods:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development
$ kubectl apply -f deployment.yaml
deployment.apps/todo-app created

azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
todo-app-7bdb9cc447-f4jrr           1/1     Running   0           29s
todo-app-7bdb9cc447-nshfw           1/1     Running   0           29s
```

Now I am trying to access it and its works well:



Finally, load balancing and health check works well in my To-do application.

5) Auto-Scaling Implementation

Remember, before that we did manual scaling (horizontal and vertical). Now let's move on to automatic scaling. As I said before, horizontal scaling is better suited for web applications. Therefore, I will implement automatic horizontal scaling to my GKE.

Implementation of automatic scaling:

To do this, we create an `hpa.yaml` file that contains the HorizontalPodAutoscaler manifest. HPA will receive the metrics of the processor used from the "metrics server", and if our condition is met, kubernetes will start adding the number of replicas of our module. We will also add the "Scale down" and "Zoom in" behaviors.

1. Create `hpa.yaml` file:

```
Assignment 4 > hpa.yaml
1  apiVersion: autoscaling/v2
2  kind: HorizontalPodAutoscaler
3  metadata:
4    name: todo-app-hpa
5  spec:
6    scaleTargetRef:
7      apiVersion: apps/v1
8      kind: Deployment
9      name: todo-app
10   minReplicas: 2
11   maxReplicas: 10
12   metrics:
13     - type: Resource
14       resource:
15         name: cpu
16         target:
17           type: Utilization
18           averageUtilization: 50
19   behavior:
20     scaleUp:
21       stabilizationWindowSeconds: 30
22       policies:
23         - type: Pods
24           value: 1
25           periodSeconds: 10
26     scaleDown:
27       stabilizationWindowSeconds: 30
28       policies:
29         - type: Percent
30           value: 50
31           periodSeconds: 60
```

Here we have indicated that HPA manages our deployment of "intermediate tasks". We have specified the number of replicas: minimum 2 and maximum 10. We also announced the CPU usage condition: 50 percent. We also specified the 'behavior', here we described the behavior when zooming out or zooming in automatically.

2. Next we apply our `hpa.yaml` to implement HPA.

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Appl
$ kubectl apply -f hpa.yaml
horizontalpodautoscaler.autoscaling/todo-app-hpa created
```

To test the work of the HPA, I will increase the artificial load on my To-do application, and we will observe the changes of horizontalPodAutoScaler live using `kubectl get hpa -[YOUR HPA] --watch` command:

```
azikkw@DESKTOP-T1GP8V3 MINGW64 /e/Study I KBTU/Semester VII/Cloud Application Development I Serek A/Assignment 4 (master)
$ kubectl get hpa todo-app-hpa --watch
NAME          REFERENCE          TARGETS          MINPODS  MAXPODS  REPLICAS  AGE
todo-app-hpa  Deployment/todo-app  cpu: 1%/50%      2         10        2          23m
todo-app-hpa  Deployment/todo-app  cpu: 3%/50%      2         10        2          24m
todo-app-hpa  Deployment/todo-app  cpu: 1%/50%      2         10        2          24m
todo-app-hpa  Deployment/todo-app  cpu: 13%/50%     2         10        2          29m
todo-app-hpa  Deployment/todo-app  cpu: 201%/50%    2         10        2          29m
todo-app-hpa  Deployment/todo-app  cpu: 201%/50%    2         10        2          30m
todo-app-hpa  Deployment/todo-app  cpu: 200%/50%    2         10        3          30m
todo-app-hpa  Deployment/todo-app  cpu: 200%/50%    2         10        3          30m
todo-app-hpa  Deployment/todo-app  cpu: 200%/50%    2         10        4          30m
todo-app-hpa  Deployment/todo-app  cpu: 201%/50%    2         10        5          30m
todo-app-hpa  Deployment/todo-app  cpu: 201%/50%    2         10        5          31m
todo-app-hpa  Deployment/todo-app  cpu: 201%/50%    2         10        6          31m
todo-app-hpa  Deployment/todo-app  cpu: 201%/50%    2         10        7          31m
todo-app-hpa  Deployment/todo-app  cpu: 201%/50%    2         10        7          31m
todo-app-hpa  Deployment/todo-app  cpu: 134%/50%    2         10        8          31m
todo-app-hpa  Deployment/todo-app  cpu: 58%/50%     2         10        8          32m
todo-app-hpa  Deployment/todo-app  cpu: 4%/50%      2         10        8          33m
todo-app-hpa  Deployment/todo-app  cpu: 4%/50%      2         10        8          33m
todo-app-hpa  Deployment/todo-app  cpu: 4%/50%      2         10        4          33m
todo-app-hpa  Deployment/todo-app  cpu: 1%/50%      2         10        4          34m
todo-app-hpa  Deployment/todo-app  cpu: 1%/50%      2         10        4          34m
todo-app-hpa  Deployment/todo-app  cpu: 1%/50%      2         10        2          34m
todo-app-hpa  Deployment/todo-app  cpu: 1%/50%      2         10        2          35m
```

Активация
Чтобы активир
"Параметры".

So, this is HPA work image:

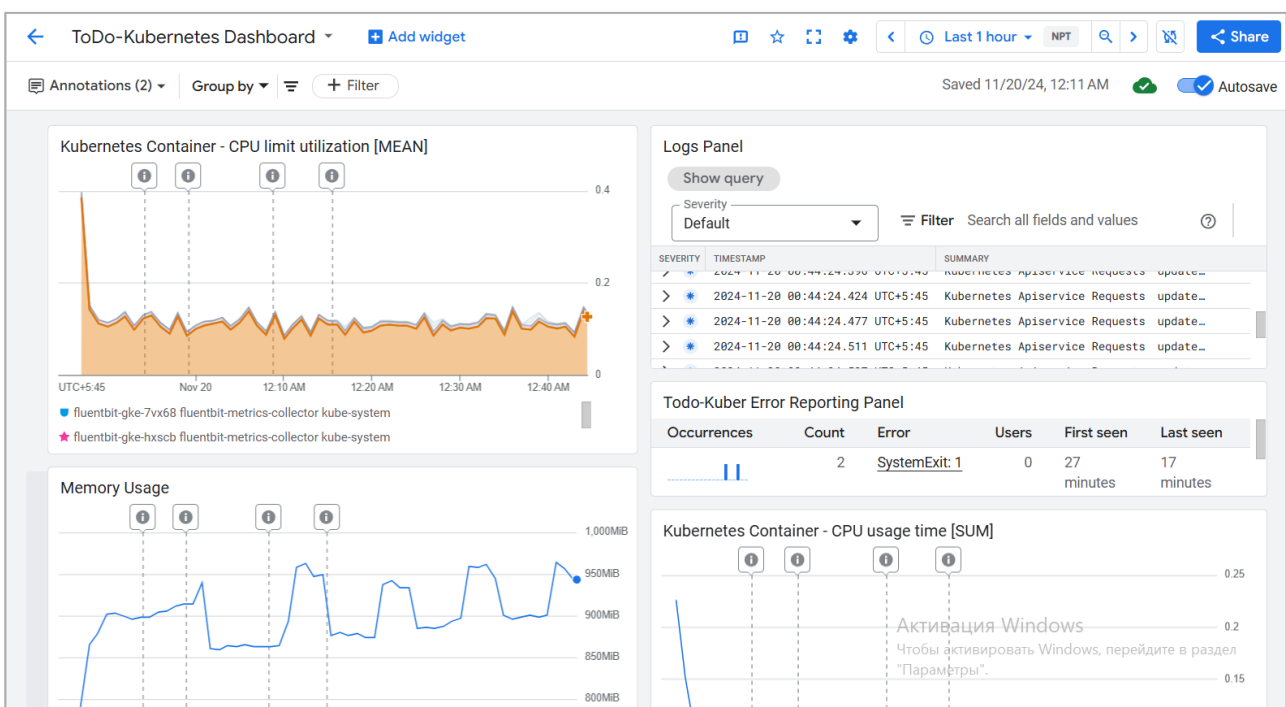
- When it just started to work and CPU is less than 50%, it uses minimum replicas number.
- When CPU is more than 50% and the load on application is increasing, the number of replicas is also starting to grow according to the load.
- When I lowered the load on application and it starts consuming less and less CPU, the number of replicas also starts to decrease accordingly.

So, this is automatic scaling, which is very convenient when you want to improve the performance of your application and ensure its constant availability.

6) Performance Monitoring

- Metrics tracked and dashboards created.

So, I use monitoring tools to check my application work. In my project I use monitoring from Google Cloud. I created special dashboard for my Kubernetes Cluster to check it work, logs, errors and performance.



How you can on this image, my dashboard name is "ToDo-Kubernetes Dashboard". I added following metrics to check my app:

1. CPU Usage, useful to check how is the CPU used, what is the load on it.
2. CPU Usage Time, to see when CPU used (SUM of usage time).
3. Memory Usage, to check how is the memory used.
4. Logs Panel, to see how Kubernetes working. It useful, because if some error will occur, we can easily find the reason and fix it.
5. Error Reporting Panel, one more tool that provides more information about occurred errors with for example pods, load balancer, etc.

7) Cost Optimization Strategies

1. Set the exact values of requests and limits for memory and processor in your Pods. This will prevent resource overruns and allow for more efficient use of cluster capacity.
2. Minimizing unused resources Check the cluster regularly for unused resources, such as unnecessary PersistentVolume, inactive services, or pods.

3. Set up automatic scaling of nodes in the kubernetes cluster, even if our application needed more power, GCP started another node to maintain the required amount of resources. Or vice versa, if the load was too low, it led to the destruction of one of the nodes.
4. If several services use external load balancers, combine them through a single Ingress controller such as Nginx. This will reduce the number of paid external IP addresses.
5. Implement caching for frequently used data (for example, Redis or Memcached) to reduce the load on the backend and reduce the amount of computing required to process requests.

6. Conclusion

In conclusion, I would like to say that cloud applications have a huge number of advantages, but they all lose their meaning if security and scalability measures are not taken into account.

Role assignments for data access and data encryption prevent unauthorized access to resources and ensure data integrity and protection. Network load balancer, HTTPS protocol with SSL certificate ensure stability and security in the network. Using monitoring and notification tools allows you to monitor the operation of the application and track errors. In addition, even if you get into an unpleasant situation, following the incident response plan, you will be able to respond quickly and solve all problems.

GKE provides us with the scalability of our application. By applying horizontal and vertical scaling strategies, we can control how our application will distribute the load and work. And automatic scaling allows us to optimally distribute power and manage costs without our intervention. Cost optimization strategies allow us to improve our financial position.

Nowadays, when IT space is developing rapidly, security and scalability are among the key factors for building not only cloud but also many other applications.

7. Recommendations

Having fully completed this assignment, I can recommend the following:

1. Role allocation is a very important aspect for the security of cloud applications. The implementation of the principle of minimum access allows you to cross unauthorized access.
2. Data encryption, storage using key management to secure your cloud application.
3. It is important to use a secure HTTPS protocol with an SSL certificate for the security of the application on the network.
4. Use Load Balancer, it is a very useful tool that will distribute the load on your application.
5. The implementation of automatic scaling in the kubernetes cluster will allow your application to automatically distribute the load without any manual intervention.
6. The implementation of monitoring and notification tools will allow you to respond to errors in a timely manner.

8. References

[1] Principle of least privilege

[Principle of least privilege - Wikipedia](#)

[2] Docker Documentation

<https://www.docker.com/>

[3] Google Kubernetes Engine (GKE) Documentation

<https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview>

[4] Cloud Load Balancing documentation

[Cloud Load Balancing documentation](#) | [Google Cloud](#)

[5] Article about Health Check

[Health checks overview](#) | [Load Balancing](#) | [Google Cloud](#)

[6] About Auto Scaling and horizontal/vertical in general | Kubernetes

[Autoscaling Workloads](#) | [Kubernetes](#)

[7] Horizontal Pod Autoscaling documentation

[Horizontal Pod Autoscaling](#) | [Kubernetes](#)

9. Appendices

I have described my every step in great detail for each of the sections, so there is no need for this.

But I want to provide github with project code:

[Cloud-AppDevelopment-2024/Assignment 4 at master · azikkw/Cloud-AppDevelopment-2024](#)