

Deployment Implementation

Monitoring, Scaling, Load Testing

Name: Lucas Jacobs

Class: S-A-RB06

PCN: 490692

Student number: 4607368

Technical teachers: Felipe Ebert, Bartosz Paszkowski

Semester coach: Gerard Elbers

Table of Contents

Introduction.....	1
Deployment: Monitoring with load testing	2
Load Testing with Automatic Scaling.....	2
Automatic Scaling During Peak Moments	5
Load Testing With Grafana Dashboard	7
Conclusion	13
Database monitoring	14
Azure Grafana monitoring	14
Conclusion.....	18

Introduction

This document will show the evidence of deployment, automatic scaling, monitoring, and load testing.

Deployment: Monitoring with load testing

Load Testing with Automatic Scaling

For load testing, I used two tools: JMeter and Postman load testing.

With load testing, I want to see if my application can handle the requests that are incoming and that the pods will increase as the load increases. The same goes for that it will auto-scale down when there is less traffic. I will use Metric Server to collect and expose resource usage metrics from the Kubernetes nodes.

Furthermore, in the Kubernetes YAML file, I will have a HorizontalPodAutoscaler (HPA), which uses the metrics provided by the Metric Server to make scaling decisions. It will adjust the number of pod replicas to the utilization standard. I will use the measurements of the CPU utilization since it is a direct indicator of the workload the service has to handle, with also being a predictable measurement because it can easily be measured and it is a fundamental resource for many services.

See the following picture of the code to set up the HPA.

```
87  apiVersion: autoscaling/v2
88  kind: HorizontalPodAutoscaler
89  metadata:
90    name: hoa-userapi
91  spec:
92    scaleTargetRef:
93      apiVersion: apps/v1
94      kind: Deployment
95      name: userapi-deployment
96    minReplicas: 1
97    maxReplicas: 10
98    metrics:
99      - type: Resource
100      resource:
101        name: cpu
102        target:
103          type: Utilization
104          averageUtilization: 50
```

Figure 1: HPA specification

I specified 50% usage of the CPU of the requested CPU resources because it makes sure that it isn't overutilized or underutilized, and it gives enough capacity to handle spike traffic without immediately requiring new pods. It will also prevent unneeded costs since when you specify it lower it can happen that there are a lot of idle resources. Moreover, avoids costs of performance issues and potential downtime when the utilization is set too high. Finally, it also comes how it performs in real-world scenarios, therefore it can also be adjusted throughout the testing.

With the performed test on getting users, I first did a postman test.

Usage: 100 virtual users.

Method: Ramp up.

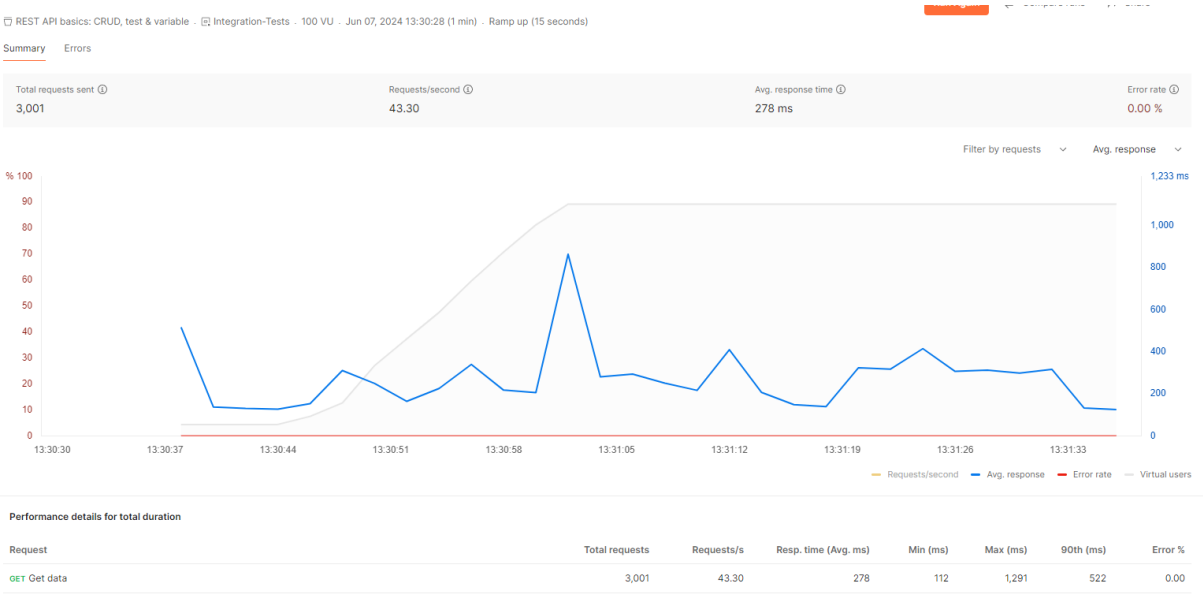


Figure 2: Metrics Test Postman

With the performing of this test, I got that the user API was scaled to three replicas.

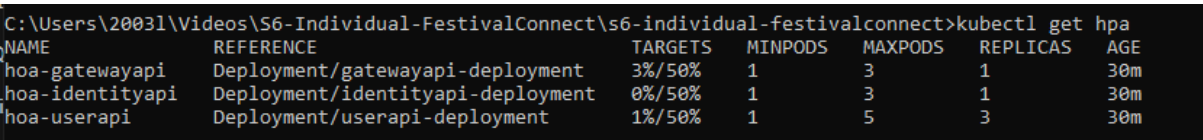


Figure 3: Scaling of pods user

After performing load testing it will scale to the needed pods. After some time when the traffic has gone down it will auto-scale down to 1

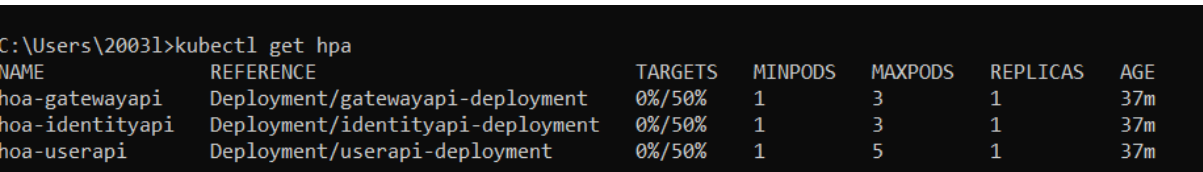


Figure 4: Scaling Down

When a test is being done when there is a spike the following will happen.

Virtual Users: 100.

Method: Spike.

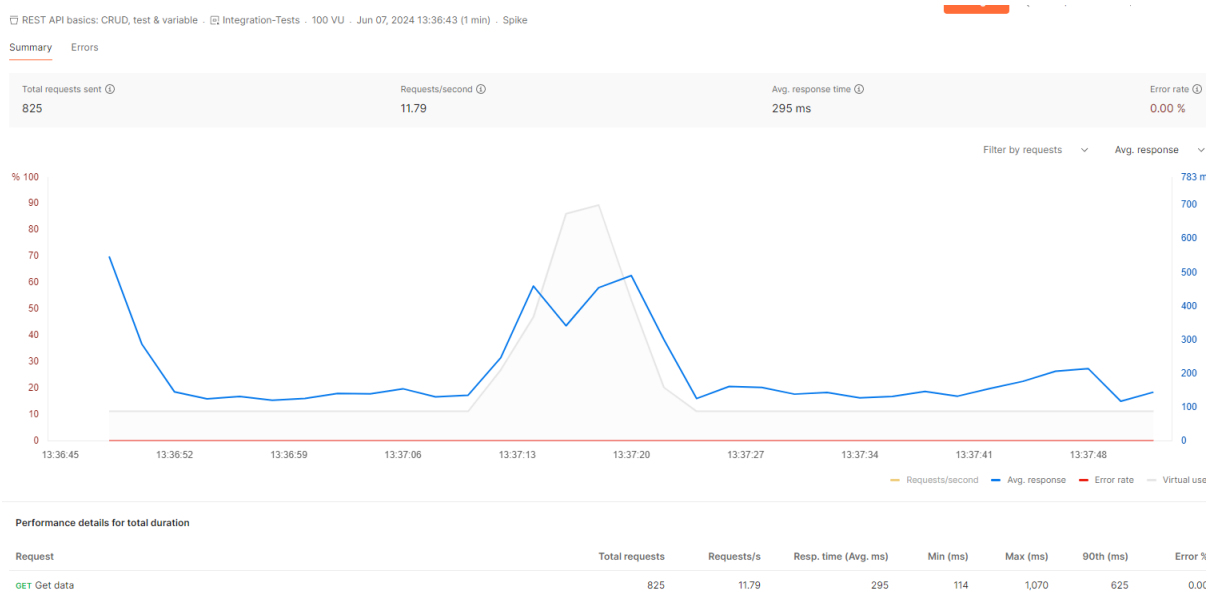


Figure 5: Spike test result

The responses instantly increase but maintain the performance requirements of being still between 0.1 seconds and 1 second, meaning that the user can notice a small delay but this would be minimal.

These tests show that it is possible that the application can be automatically scaled. Note, that the Azure account has a student subscription, I was limited to the sources I had, therefore, the low limit and the maximum pods were set.

Automatic Scaling During Peak Moments

Festivals happen mostly during the weekends. This is the time where people are more likely to be active on the application. Therefore, I will use CronJob to perform an action that during the weekends the application will scale and after the weekend it will scale down. To give an example see the following picture.

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: user-scale-up-for-weekend
spec:
  schedule: "0 10 * * 5" # 10:00 AM on Fridays
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: scale
              image: bitnami/kubectl:latest
              command:
                - /bin/sh
                - -c
                - kubectl scale deployment userapi-deployment --replicas=10
          restartPolicy: OnFailure

---

apiVersion: batch/v1
kind: CronJob
metadata:
  name: user-scale-down-after-weekend
spec:
  schedule: "0 9 * * 1" # 9:00 AM on Mondays
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: scale
              image: bitnami/kubectl:latest
              command:
                - /bin/sh
                - -c
                - kubectl scale deployment userapi-deployment --replicas=1
          restartPolicy: OnFailure
```

Figure 6: CronJob For Scaling During weekends

In this I specify that deployment will increase in terms of services, having it scaled up at 10 AM on Fridays, and scaling it down at 9 AM on Mondays.

So during weekends, it will scale to 10 replicas and during weekdays it will scale between 1 and 5 based on the CPU utilization. The jobs are defined and can be found in the Kubernetes cluster under ‘Cron jobs’.

Deployments Pods Replica sets Stateful sets Daemon sets Jobs Cron jobs

Filter by cron job name

Filter by namespace

Enter the full cron job name

All namespaces

Add label filter

<input type="checkbox"/>	Name	Namespace	Schedule	Suspend	Active jobs	Last schedule time	Age ↓
<input type="checkbox"/>	user-scale-down-after-weekend	default	0 9 * * 1	False	0	-	19 minutes
<input type="checkbox"/>	identity-scale-down-after-weekend	default	0 9 * * 1	False	0	-	19 minutes
<input type="checkbox"/>	identity-scale-up-for-weekend	default	0 10 * * 5	False	0	-	16 minutes
<input type="checkbox"/>	user-scale-up-for-weekend	default	0 10 * * 5	False	0	-	16 minutes

Figure 7: Cronjob in Azure

Load Testing With Grafana Dashboard

I also made a test in Jmeter, where I could exceed the limit of having more than 100 virtual users. but resources were very limited, and with the response time graph being unclear in Jmeter, I for now went with Postman. For creating a user, the test can become a little hard to set up when you need unique emails all the time. Therefore, I went with a post method on the 'post-service' to create a post that belongs to the community.

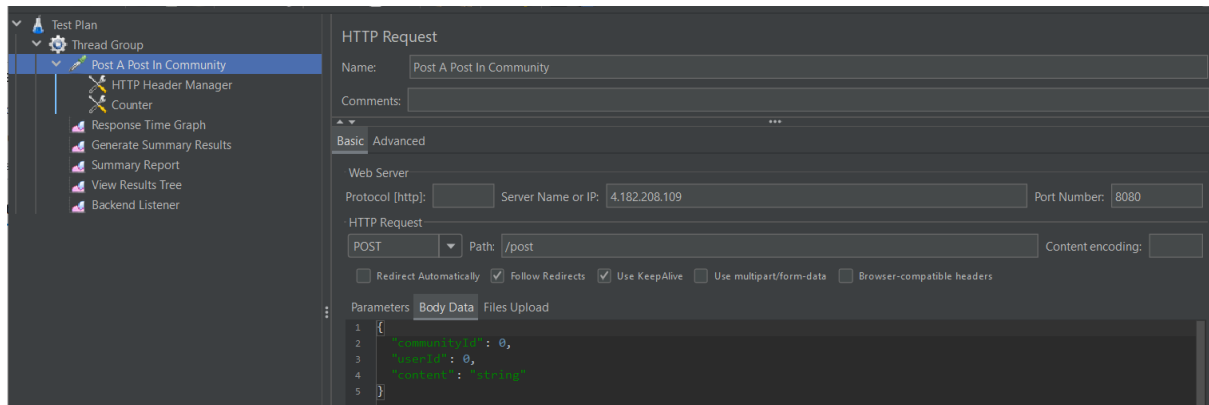


Figure 8: Jmeter test

It needed some authorization which is specified in the HTTP Header Manager.

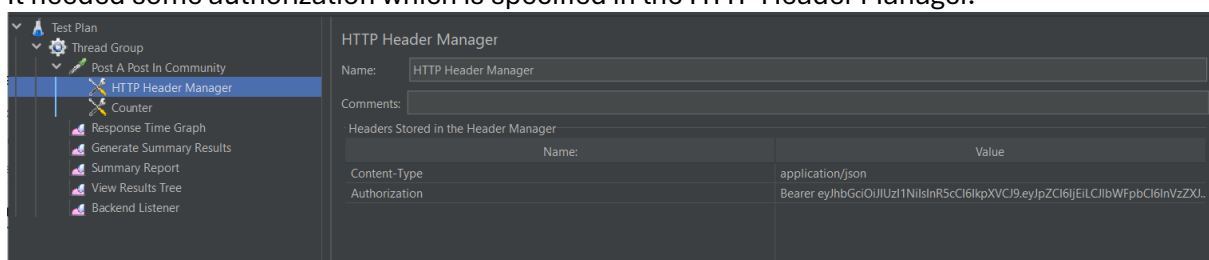


Figure 9: Header HTTP

For displaying the results, I used a Grafana Dashboard that retrieves data from a influx database, who listens to the test. The Influx database listens with a backend listener to the JMeter test.

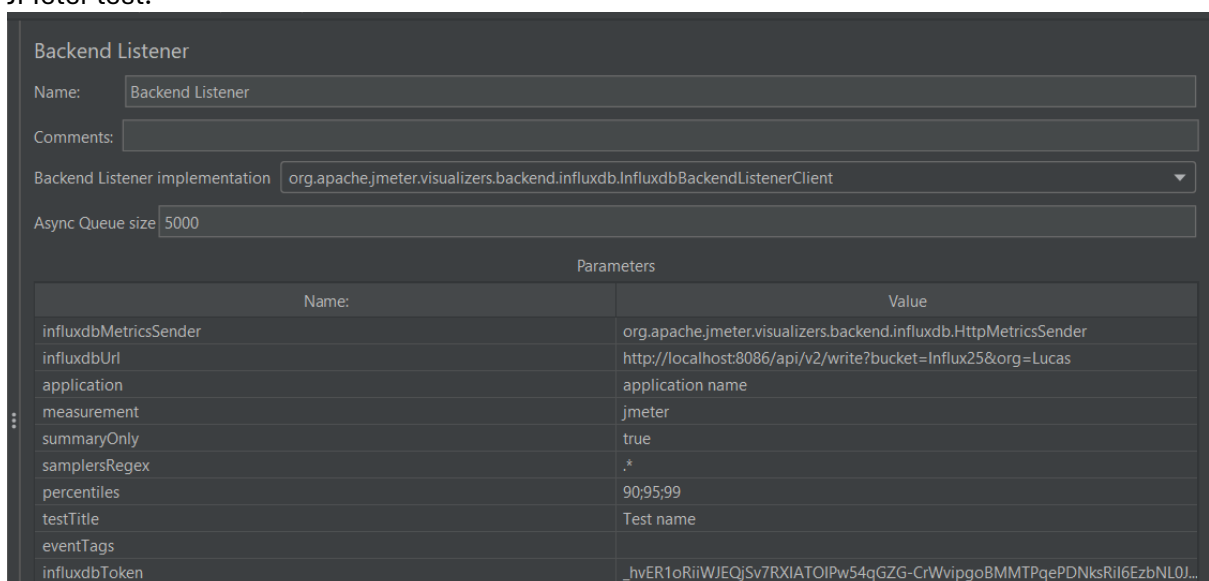
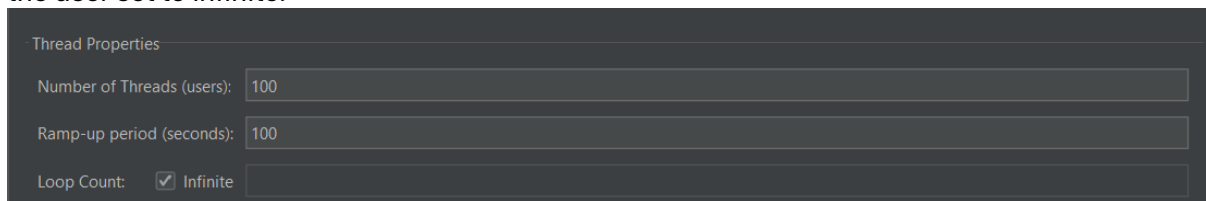


Figure 10: Influx Backend Listener

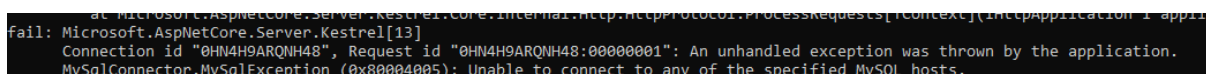
First I did a basic test, which ramps up with one user per second and having the test amount on the user set to infinite.



The image shows the 'Thread Properties' dialog box in JMeter. It has three input fields: 'Number of Threads (users)' set to 100, 'Ramp-up period (seconds)' set to 100, and 'Loop Count' with a checked 'Infinite' checkbox.

Figure 11: Test settings JMeter first try

After a minute of load, the Azure MySQL Database Server can not keep up with the incoming load. Since I am using a student account, I cannot afford a high-performing database server, therefore keeping it limited to performing high-load tests. See the following picture of the error.

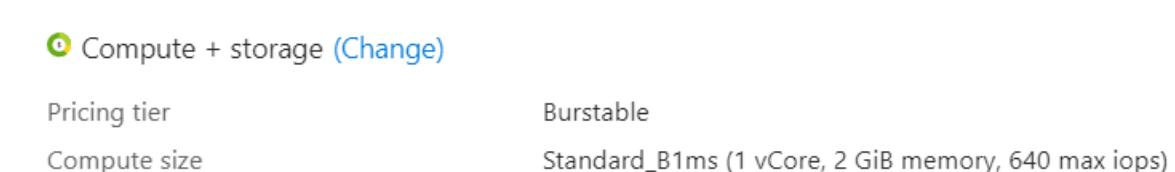


The image shows a snippet of a log file with the following text:


```
fail: Microsoft.AspNetCore.Server.Kestrel.Core.Internal.Http.HttpProtocol.ProcessRequests[context](HttpContext) application
fail: Microsoft.AspNetCore.Server.Kestrel[13]
Connection id "0HN4H9ARQNH48", Request id "0HN4H9ARQNH48:00000001": An unhandled exception was thrown by the application.
MySQLConnector.MySqlException (0x80004005): Unable to connect to any of the specified MySQL hosts.
```

Figure 12: MySQL Error

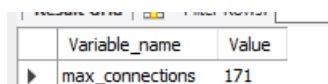
I only have one CPU Core and two gigabytes of RAM for the database server. It can also lay around handling a max of 640 input/output operations per second to and from its storage (IOPS). This can occur since there can be multiple operations during a post (insert the data, background tasks, commit a transaction, etc).



The image shows the 'Compute + storage' specifications for an Azure database server. It includes a 'Pricing tier' of 'Burstable' and a 'Compute size' of 'Standard_B1ms (1 vCore, 2 GiB memory, 640 max iops)'. There is a '(Change)' link next to the pricing tier.

Figure 13: Compute Specifications Database Server

This error can occur due to various things. During an observation that I did while doing a load test on my local application using the Azure database server, it can easily handle all the requests meaning it has sufficient capacity. Therefore it can handle incoming requests and it does not exceed the concurrent maximum connection setting which is the following.



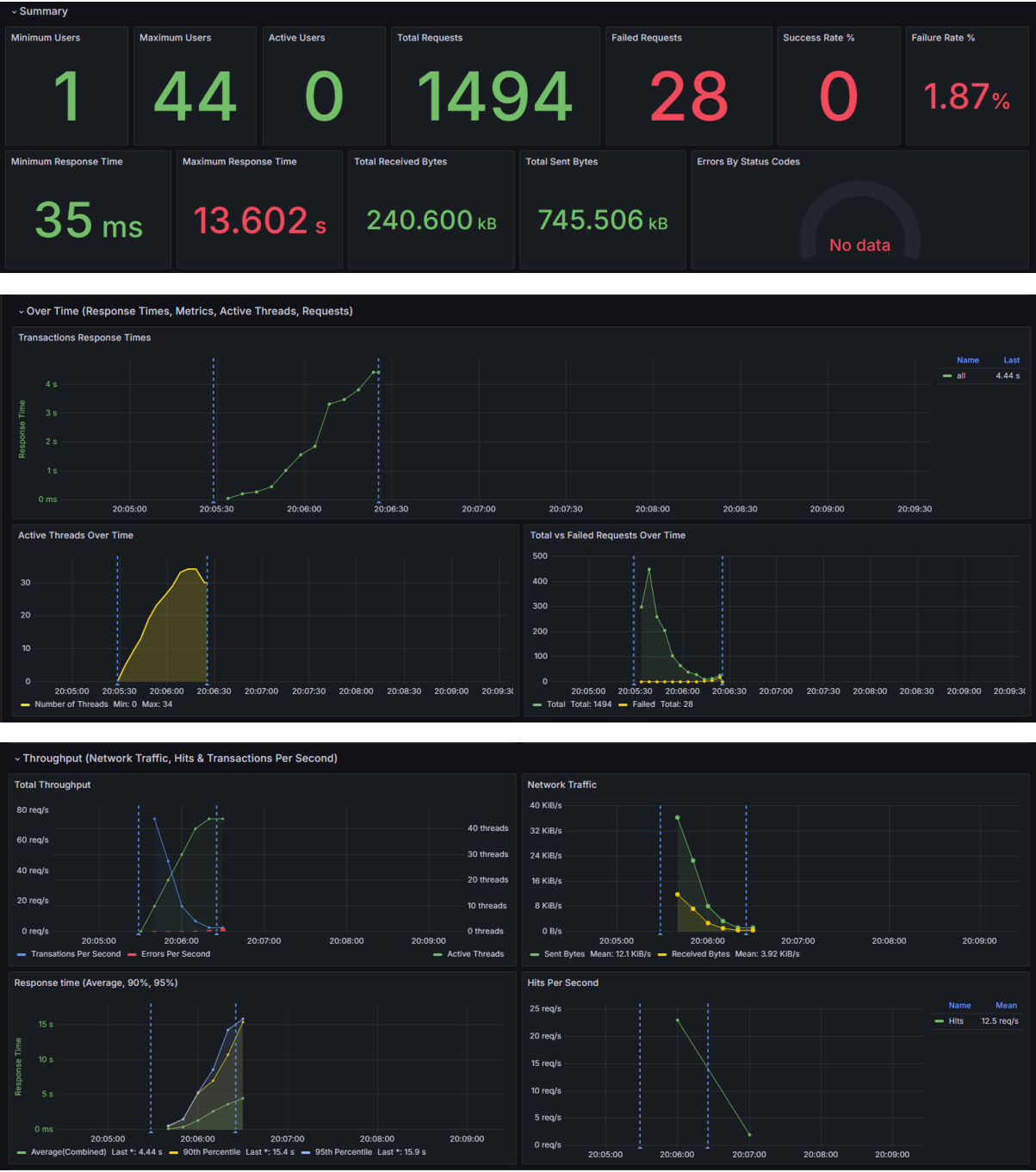
Variable_name	Value
max_connections	171

Figure 14: Max connections Database

So from the Kubernetes deployment high load test, the connection errors were frequently occurring. This suggests that it does not relate to the Azure MySQL Database Server itself, but to the way how the Kubernetes server is managing and routing the incoming requests to the database.

Therefore, this error can occur from several things such as connection pool, not enough resources, network latency, and misconfigurations. By trying to eliminate the potential bottlenecks, the thing I concluded that it probably has to do something with either misconfiguration or network latency.

See the following results of the dashboard.



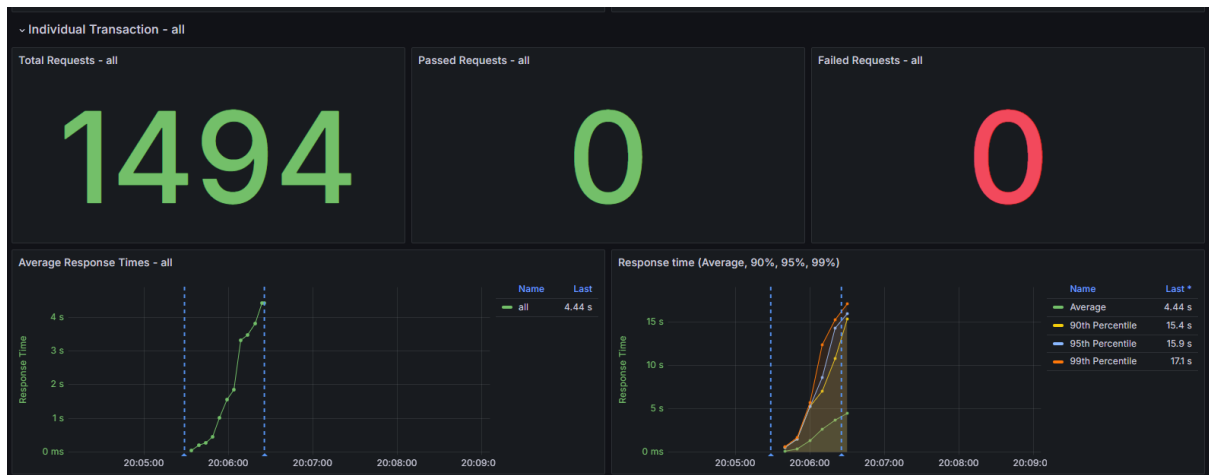


Figure 15: Dashboard Monitoring Grafana

This dashboard covers the important key performance indicators (KPIs) such as:

- Application Response Time.
- Peak Response Time.
- Error Rate.
- Concurrent Users.
- Requests per Second.

To perform more load on the post service, I did an Azure load test. With this Azure load test, I could upload the test that I made in JMeter and execute it using the engines provided by Azure.

Furthermore, I specified an Auto-stop test for when it reaches a specific error percentage to prevent extra costs when something was incorrectly configured.

Auto-stop test

The test will automatically stop if the error percentage is high for the specified time window. This prevents incurring costs for an incorrectly configured test. You can disable this or customize the error percentage and time window. [Learn more](#)

Auto-stop test * ⓘ

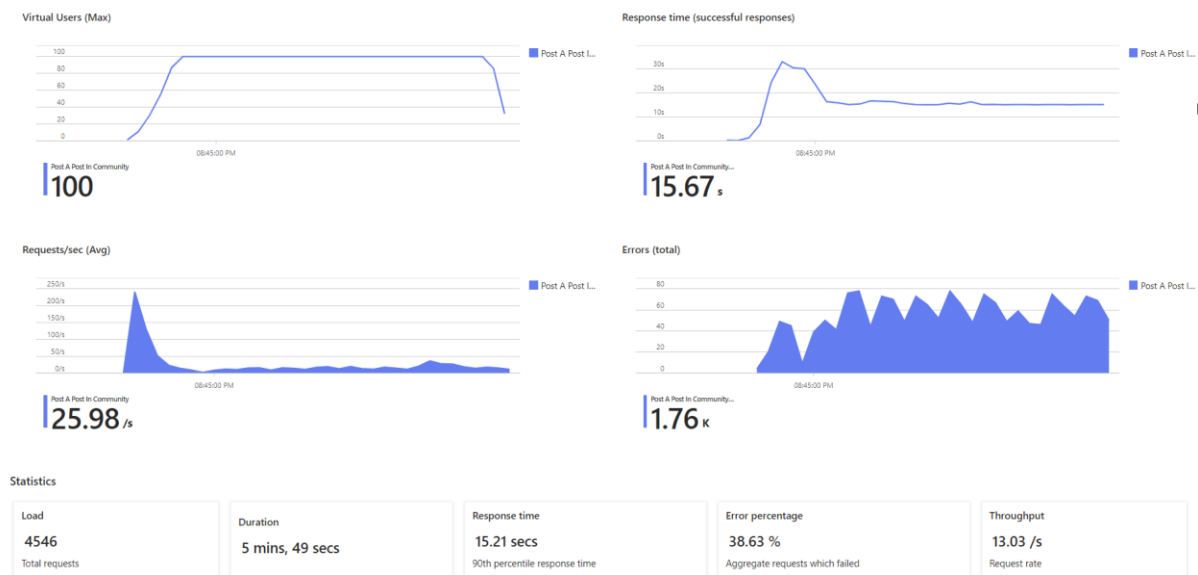
☒ Enable
☐ Disable

Error percentage * ⓘ

Time window (seconds) * ⓘ

Figure 16: Auto Stop Test

When performing the I get the following results



So it will start performing a lot of requests, but eventually, it will produce a lot of errors, resulting in fewer requests per second and handling a lot of errors.

When performing another test on login request it eventually also fails.

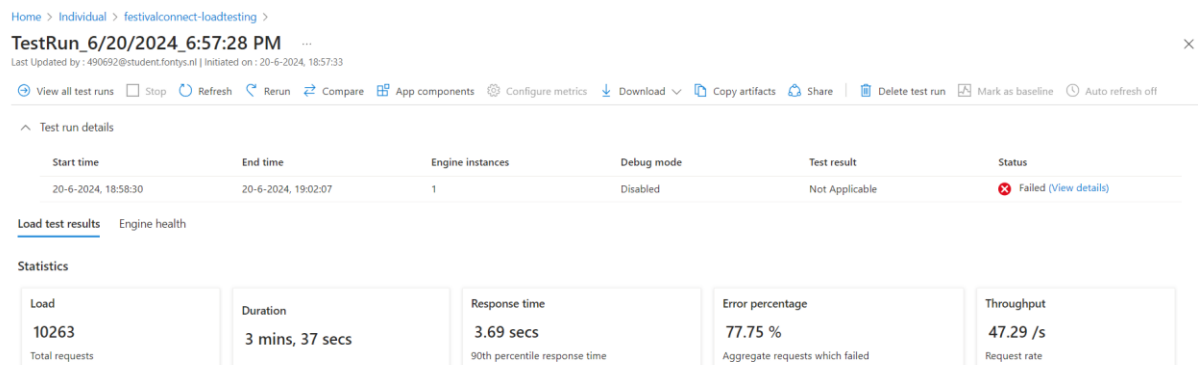


Figure 17: Test Results Login

With also having this kind of error.

```
fail: Microsoft.AspNetCore.Server.Kestrel.Core.Internal.Http.HttpProtocol.ProcessRequests[1] (HttpContext) application
fail: Microsoft.AspNetCore.Server.Kestrel[13]
Connection id "0HN4H9ARQNH48", Request id "0HN4H9ARQNH48:00000001": An unhandled exception was thrown by the application.
MySqlConnection.MySqlConnection (0x80004005): Unable to connect to any of the specified MySQL hosts.
```

Figure 18: Error Login Load Test

When performing this test the identity services scaled in size.

Pod Name	Namespace	Ready	Running	Restarts	Age	IP Address	Node Name
csi-azurefile-node-j8p5	kube-system	3/3	Running	0	45 minutes	10.224.0.4	aks-agentpool-1945846...
kube-proxy-xwnx5	kube-system	1/1	Running	0	45 minutes	10.224.0.4	aks-agentpool-1945846...
coredns-6695469449-kq...	kube-system	1/1	Running	0	44 minutes	10.224.0.36	aks-agentpool-1945846...
metrics-server-5cd44496f...	kube-system	2/2	Running	0	44 minutes	10.224.0.57	aks-agentpool-1945846...
metrics-server-5cd44496f...	kube-system	2/2	Running	0	44 minutes	10.224.0.13	aks-agentpool-1945846...
usercache-777dbb876c-8...	default	1/1	Running	0	33 minutes	10.224.0.19	aks-agentpool-1945846...
rabbitmq-6c88bb475-p...	default	1/1	Running	0	33 minutes	10.224.0.107	aks-agentpool-1945846...
identityapi-79758bc6f5-g...	default	1/1	Running	0	33 minutes	10.224.0.48	aks-agentpool-1945846...
userapi-deployment-869...	default	1/1	Running	0	33 minutes	10.224.0.73	aks-agentpool-1945846...
gatewayapi-deployment-...	default	1/1	Running	0	33 minutes	10.224.0.98	aks-agentpool-1945846...
connectivity-agent-6bb7...	kube-system	1/1	Running	0	13 minutes	10.224.0.69	aks-agentpool-1945846...
connectivity-agent-6bb7...	kube-system	1/1	Running	0	13 minutes	10.224.0.11	aks-agentpool-1945846...
identityapi-79758bc6f5-t...	default	1/1	Running	0	10 minutes	10.224.0.40	aks-agentpool-1945846...
identityapi-79758bc6f5-l...	default	1/1	Running	0	9 minutes	10.224.0.27	aks-agentpool-1945846...
identityapi-79758bc6f5-n...	default	1/1	Running	0	9 minutes	10.224.0.28	aks-agentpool-1945846...
identityapi-79758bc6f5-g...	default	1/1	Running	0	8 minutes	10.224.0.17	aks-agentpool-1945846...

Figure 19: Identity Service Scaled

After performing these tests, the costs increased significantly after only performing three quick tests. I estimated the costs beforehand, to prevent extra costs.

Azure Load Testing

Region: East US Estimation Method: Load test details

Load Testing Resource

1 **x** \$10.00 **=** \$10.00
Resource Per resource

Load Test Details

Each Azure Load Testing resource includes 50 virtual user hours per month.

100 **x** 5 **x** 3
Virtual users per test Test Duration (minutes) Test runs per month

+ ADD ANOTHER TEST

Overage

0 **=** \$0.00
Total virtual usage hours over included hours

Figure 20: Cost Estimation

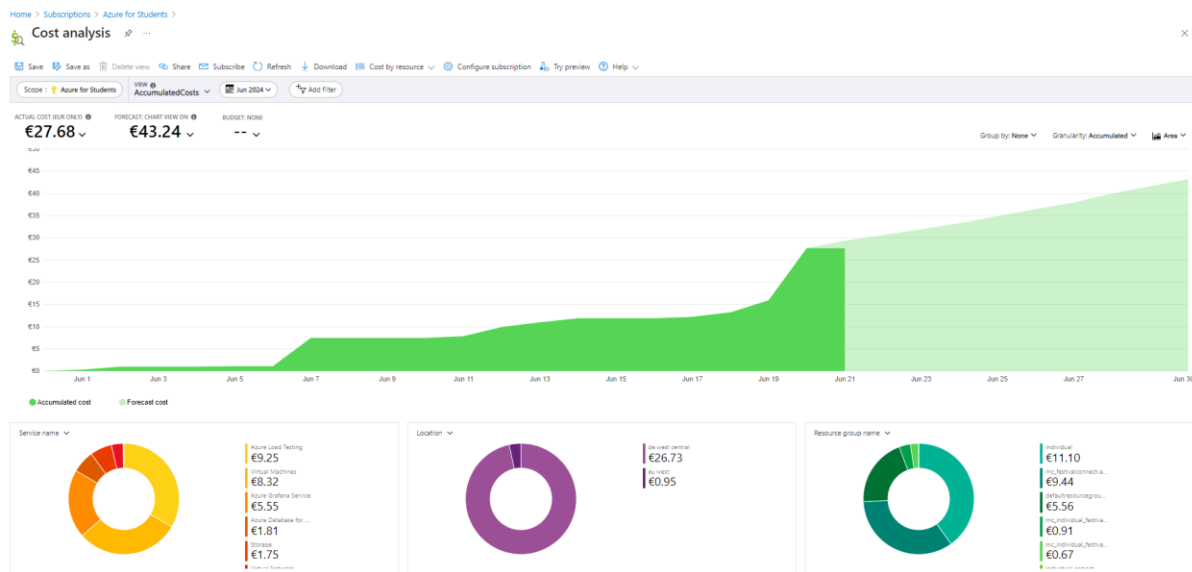


Figure 21: Realized Costs After Tests

When performing more heavy tests, you exceed the free usage of 50 Virtual Hour Usage, which will then drastically increase the cost, which I could not afford with the student account.

Conclusion

It came down to having a proper way of showing the result properly and showing the results that the service can automatically scale up and down. By looking at Azure performance testing where I performed some tests in, I could not make more tests on it because it became too expensive. Also with the limited resources I had, I could not show a proper load testing with actual thousands of requests per second. I showed various tests that the application can be automatically scalable. Also, I used a great dashboard to display all my metrics that gave some KPI information about the application.

These tests came down to improving my insides on what the bottlenecks are and how the non-functional requirements can improve on this:

- **Performance:** Improving response time to be between 0.0 and 1.0 seconds for Post methods.
- **Scalability:** Making use of Metric Server, HPA, and CronJobs to auto-scale to a specific amount of pods during the weekends
- **Reliability:** available when the application spikes without crashing. This was only with getting requests, for the post, it eventually gave constant errors, having most likely not enough resources.
- **Cost-efficiency:** CPU utilization.
- **Maintainability:** Usage of HPA and CronJobs to automate scaling, preventing manual operations.

Database monitoring

For the database, I am using the database services of MySQL in Azure. See the following metrics that can be monitored for the database when performing load testing.

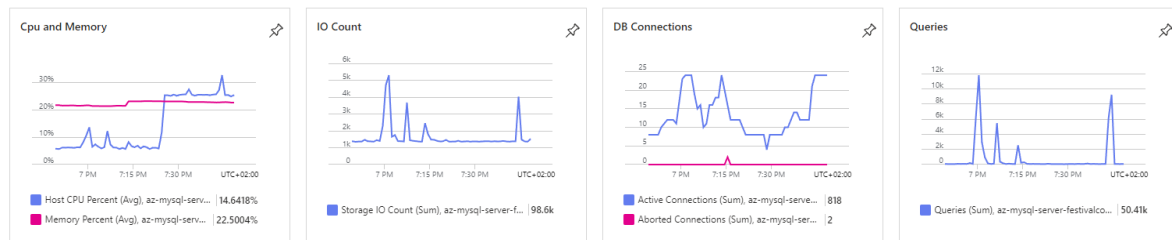


Figure 22: Database Monitoring

Azure Grafana monitoring

In the Jmeter testing, I already used Grafana. I also used Grafana in Azure so that I can monitor my application metrics for CPU usage etc. This can be further analyzed by viewing metrics per pod.

The usage of Grafana is quite expensive, therefore I used it for looking at some small metrics, trying to limit the resources that I will use. With Grafana, I could monitor the overall cluster, or each node individually to monitor several metrics.

The following pictures will show the up-and-running services that are deployed.

```
C:\Users\20031\Videos\S6-Individual-FestivalConnect\s6-individual-festivalconnect\k8s>kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
frontend	LoadBalancer	10.0.29.29	4.182.169.114	80:31360/TCP	94s
gatewayapi	LoadBalancer	10.0.10.231	4.182.208.109	8080:31067/TCP	92m
identityapi	ClusterIP	10.0.193.101	<none>	8080/TCP	90m
kubernetes	ClusterIP	10.0.0.1	<none>	443/TCP	95m
postapi	ClusterIP	10.0.50.216	<none>	8080/TCP	91m
rabbitmq	NodePort	10.0.97.18	<none>	5672:31824/TCP,15672:30258/TCP	91m
userapi	ClusterIP	10.0.187.14	<none>	8080/TCP	90m
usercache	ClusterIP	10.0.83.126	<none>	6379/TCP	91m

Figure 23: Services Over

Name ↑	Type ↑	Location ↑	
az-mysql-server-festivalconnect	Azure Database for MySQL flexible server	Germany West Central	...
festivalconnect-aks	Kubernetes service	Germany West Central	...
individualkeyvault	Key vault	West Europe	...

festivalconnect-loadtesting	Azure Load Testing	Germany West Central	...
-----------------------------	--------------------	----------------------	-----

Figure 24: Azure Services Used

Name	Namespace	Ready	Up-to-date	Available	Age ↓
coredns	kube-system	2/2	2	2	21 minutes
coredns-autoscaler	kube-system	1/1	1	1	21 minutes
connectivity-agent	kube-system	2/2	2	2	21 minutes
metrics-server	kube-system	2/2	2	2	21 minutes
rabbitmq	default	1/1	1	1	18 minutes
gatewayapi-deployment	default	1/1	1	1	18 minutes
identityapi-deployment	default	1/1	1	1	18 minutes
userapi-deployment	default	1/1	1	1	17 minutes

Figure 25: the running pods in k8s cluster

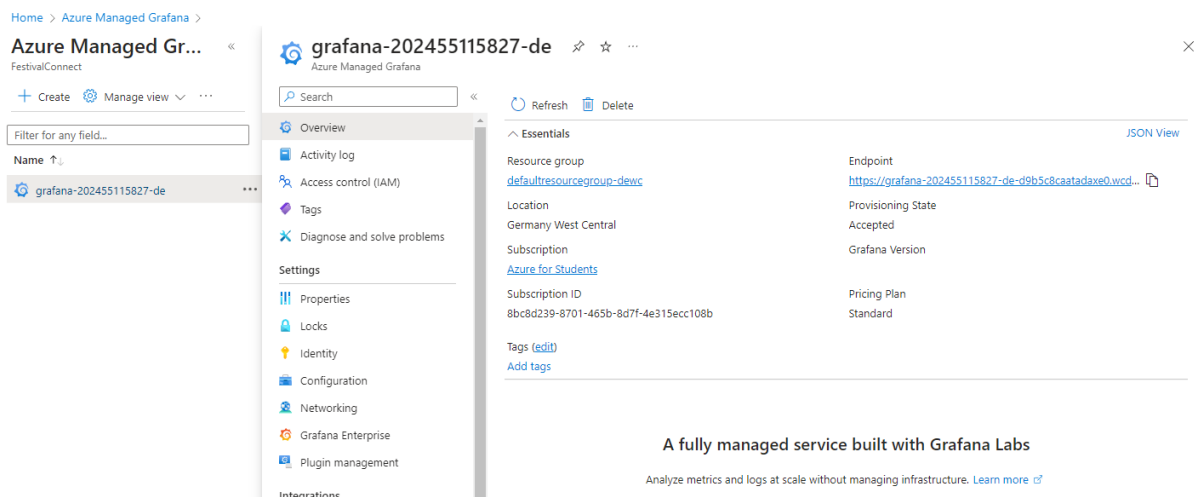


Figure 26: Grafana service in Azure

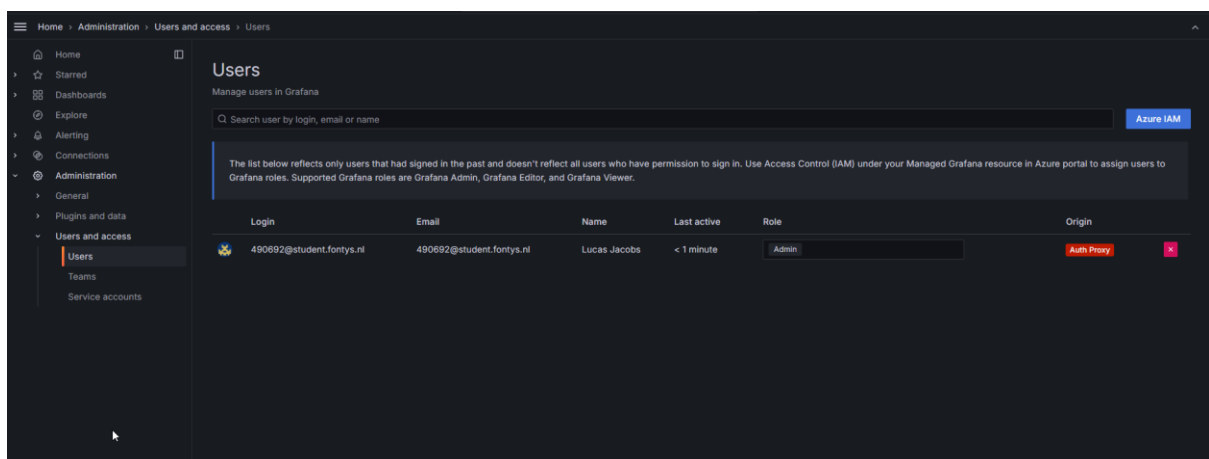


Figure 27: User that has access to Grafana

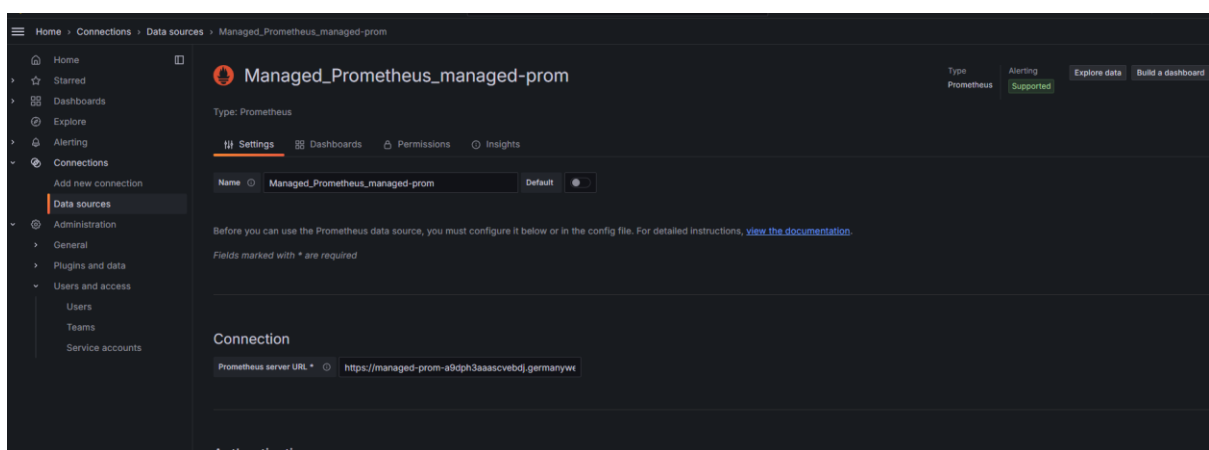


Figure 28: Dataconnection

In the dashboard section, azure creates multiple monitor options to either monitor the cluster or a specific pod. See the following of monitoring a single pod.

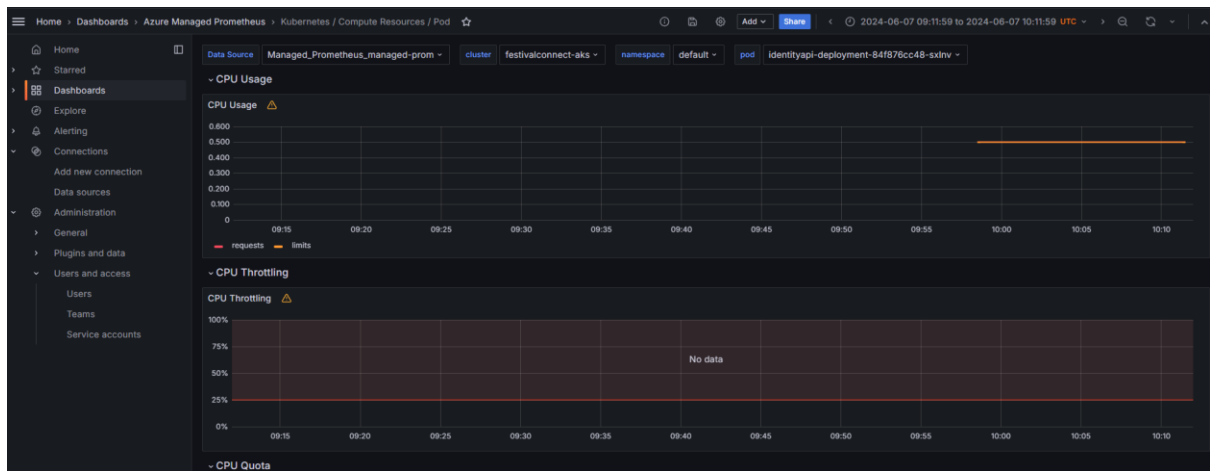


Figure 29: Monitoring Identity Api



Figure 30: Monitoring User Api

The following shows the metrics of the cluster itself and contains the metrics of the resource usage.

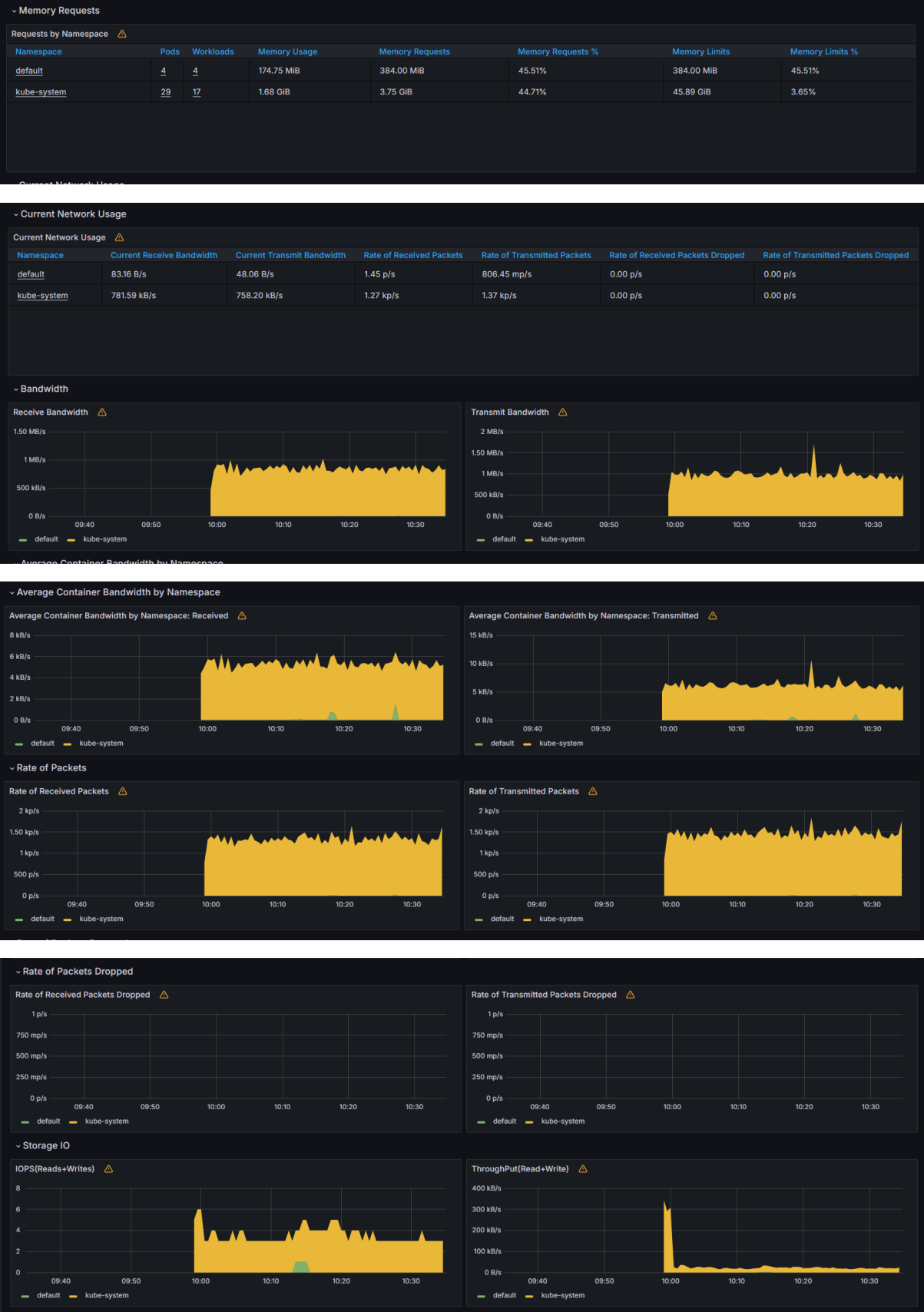


Figure 31: Monitoring Cluster

Conclusion

With monitoring, we can check for nonfunctional requirements, if they are being met, and based on the information optimize the application. It can identify several things such as

- **Performance:** tracking if response time and throughput remain responsive under different load conditions.
- **Scalability:** Helps to plan if you need more resources.
- **Security:** Can help in detecting unusual patterns that can indicate to an attack.
- **Reliability:** Monitoring can help to find bottlenecks and address them before any downtime.
- **Cost:** Making sure the resources are being used efficiently.