

Master's Degree in Agile Software Development for the Web

Final Work Report:
CLOUD COMPUTING AND CONTAINERS

Debit Card Withdrawal Tool

Tomasz Sojka

1.Introduction

The Final's Work aim was to create an architecture in a Kubernetes cluster with management through Istio, where different types of images are deployed. The created images had to use at least one cache system and a database. The application should be divided into at least 4 separate services. One access service for validating who is sending a request. If validation passed well, the request would be passed to reception service, which prepares and gathers all the needed data to make a request. If everything went well, the request is passed to the processing service and eventually the response is sent to the result service. Each service had to be replicated with a minimum of 3 instances, and must make use of the aforementioned services. All services should have 2 different versions. The domain design of the application was at discretion. The topic chosen was a withdrawal tool / bank app.

2.Application

First step to be carried out was to design and implement a complete application. The topic chosen was a withdrawal tool / bank app.

The operation of the application is simple. First, the user needs to log in to get access to all the endpoints. After logging in, credentials are stored in cache. The user is logged in for up to 10 minutes. At this time, or until he logs out, he can check the account balance or he can withdraw money from the account (It is checked if the account has enough money to make a withdrawal).

2.1 Division of the application into separate services

As required, application is divided into 4 separate services:

- bank-access-service – Service which takes all the requests (excluding login and logout). It checks who is trying to access the application.
- bank-reception-service – Service is accessed only if the user was validated by access service. It checks if all the circumstances are met to make what was requested.
- bank-request-processing-service – Service is accessed only if the user was first validated by access service and then all the circumstances were met in reception service. Here the “real work” is done.
- bank-result-service – Service to send responses to the user. It is used by all of the other services.

Additionally, the application has implemented 2 services that use Redis cache system and MYSQL database:

- bank-mysql-server – Service initiates MYSQL database and stores debit card details.
- bank-redis-server – Service is used to store session data.

What is more, there is one more service implemented that enables logging into and logging out of the application:

- bank-authentication-service – Service allows login and logout from the application.

The services are communicating by sending the data in JSON format. Requests from the client are also accepted in a JSON format. The system allows a traffic packet to have a specific X-type field in the HTTP header. In inbound traffic the “x-card-id” field is used to send card id between services.

2.2 Database

```
CREATE DATABASE bankapp;
USE bankapp;

CREATE TABLE `cards` (
  `card_id` INT NOT NULL AUTO_INCREMENT,
  `card_number` varchar(255),
  `card_pin` INT NOT NULL,
  `card_money` varchar(255),
  PRIMARY KEY (`card_id`)
);

INSERT INTO cards (card_number, card_pin, card_money) VALUES ('1234123412341234', 1234, '1000');
INSERT INTO cards (card_number, card_pin, card_money) VALUES ('2345234523452345', 2345, '20');
INSERT INTO cards (card_number, card_pin, card_money) VALUES ('3456345634563456', 3456, '5000');
INSERT INTO cards (card_number, card_pin, card_money) VALUES ('4567456745674567', 4567, '300');
INSERT INTO cards (card_number, card_pin, card_money) VALUES ('5678567856785678', 5678, '50');
INSERT INTO cards (card_number, card_pin, card_money) VALUES ('6789678967896789', 6789, '2');
```

A sql script used in bank-mysql-server service to create a database and to insert example debit card details.

2.3 Possible requests

- login – Request first arrives at the auth service, which checks if debit card credentials (card number and pin) exist in the database. If the card was found in the database, the user is being logged in, the session data is set and the response is sent to the result service. If login data is not valid, the result with an error message is sent to the result service.
- logout – Request first arrives at the auth service. If the user is logged in, session data is destroyed and a request is passed to the result service. If the user is not logged in, an empty response is passed to the result service with status 401.
- check account balance – Request first arrives at the access service, which checks if the user was logged in to an existing account. If the user is logged in, the request is passed further to reception service. If the user was not logged in, an empty response with status 401 is passed to the result service. Reception service, which is used only if the user was logged in, finds the card in the database and passes the balance of the account or error message in case the user is missing to the result service.
- withdraw – Request first arrives at the access service, which checks if the user was logged in to an existing account. If the user is logged in, the request is passed further

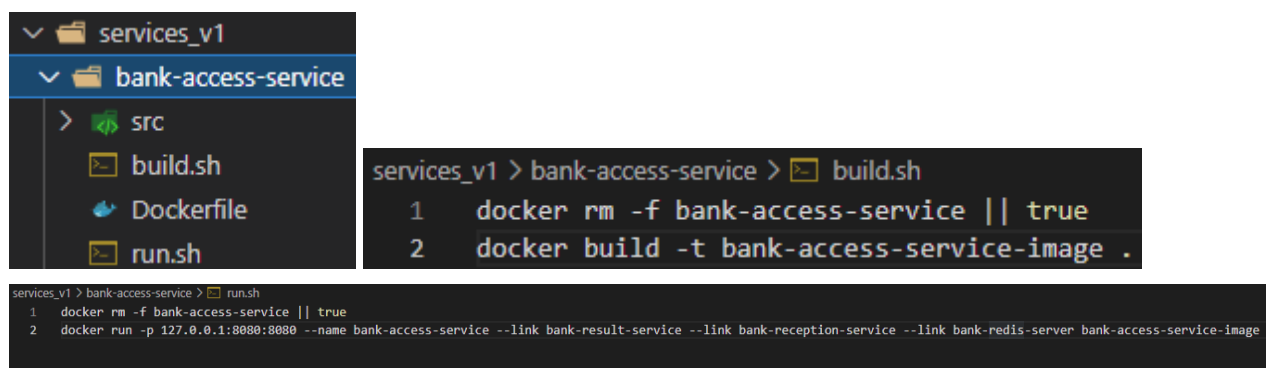
to reception service. If the user was not logged in, an empty response with status 401 is passed to the result service. Reception service, which is used only if the user was logged in, finds if the card exists in the database and if there is enough money on the account to make a withdrawal. If both aforementioned requirements are met, the request is passed with all the needed data to request processing service, where the amount of money to withdraw is subtracted from the balance and the new account balance is passed to the result service. If any of the requirements are not met, the request is passed straight to the result service with the information about what happened – the error message that the card was not found in the database or message that there is not enough money on the account to make withdrawal.

3. Operations

The operations carried out for the final work.

3.1 Application implementation.

First operation to be carried out was to create a complete application. Images were created for each service. In the beginning each image was built separately in the terminal using the “docker build” command and Dockerfiles. Same for running containers, each was run separately in the terminal using the “docker run” command with “--link” option to make services see each other by name. Afterwards, the process was partly automated by creating “build.sh” and “run.sh” files for all services, in each folder. To run “*.sh” files command “chmod +x <file name>” had to be used first, even though it significantly shortened the time needed to build the images and start the containers.



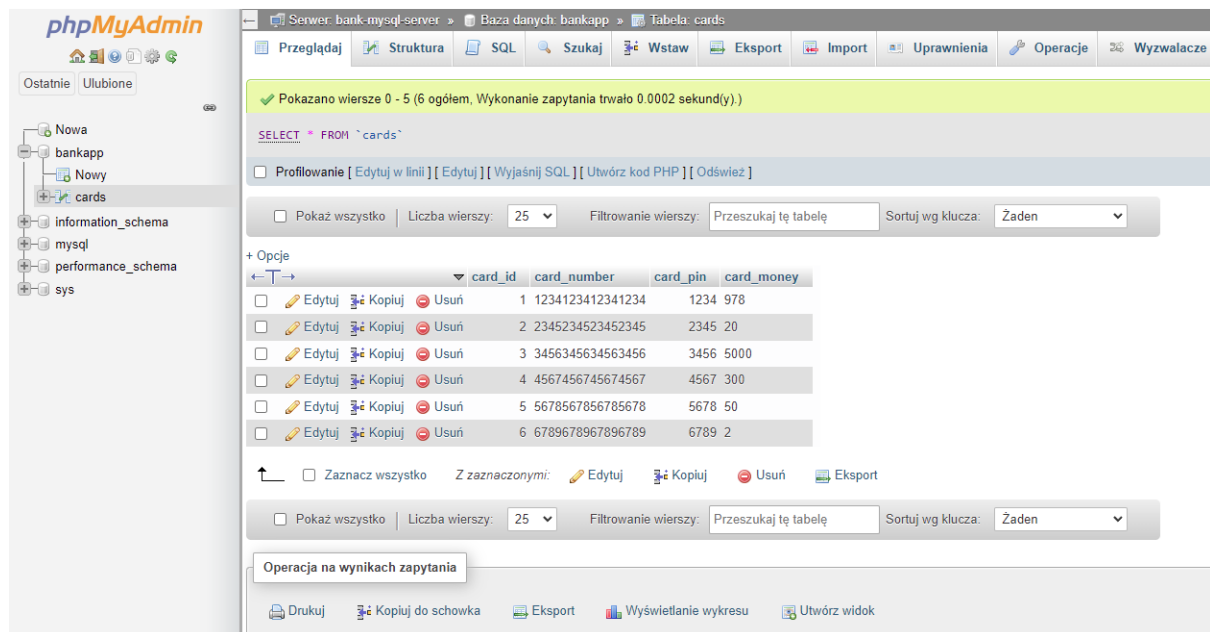
The image shows a file explorer on the left and a terminal window on the right. The file explorer displays the directory structure for 'services_v1' > 'bank-access-service', including a 'src' folder, 'build.sh', 'Dockerfile', and 'run.sh' files. The terminal window shows the execution of 'build.sh' and 'run.sh' scripts. The 'build.sh' script contains two commands: 'docker rm -f bank-access-service || true' and 'docker build -t bank-access-service-image .'. The 'run.sh' script contains two commands: 'docker rm -f bank-access-service || true' and 'docker run -p 127.0.0.1:8080:8080 --name bank-access-service --link bank-result-service --link bank-reception-service --link bank-redis-server bank-access-service-image'.

```
services_v1 > bank-access-service > build.sh
1  docker rm -f bank-access-service || true
2  docker build -t bank-access-service-image .

services_v1 > bank-access-service > run.sh
1  docker rm -f bank-access-service || true
2  docker run -p 127.0.0.1:8080:8080 --name bank-access-service --link bank-result-service --link bank-reception-service --link bank-redis-server bank-access-service-image
```

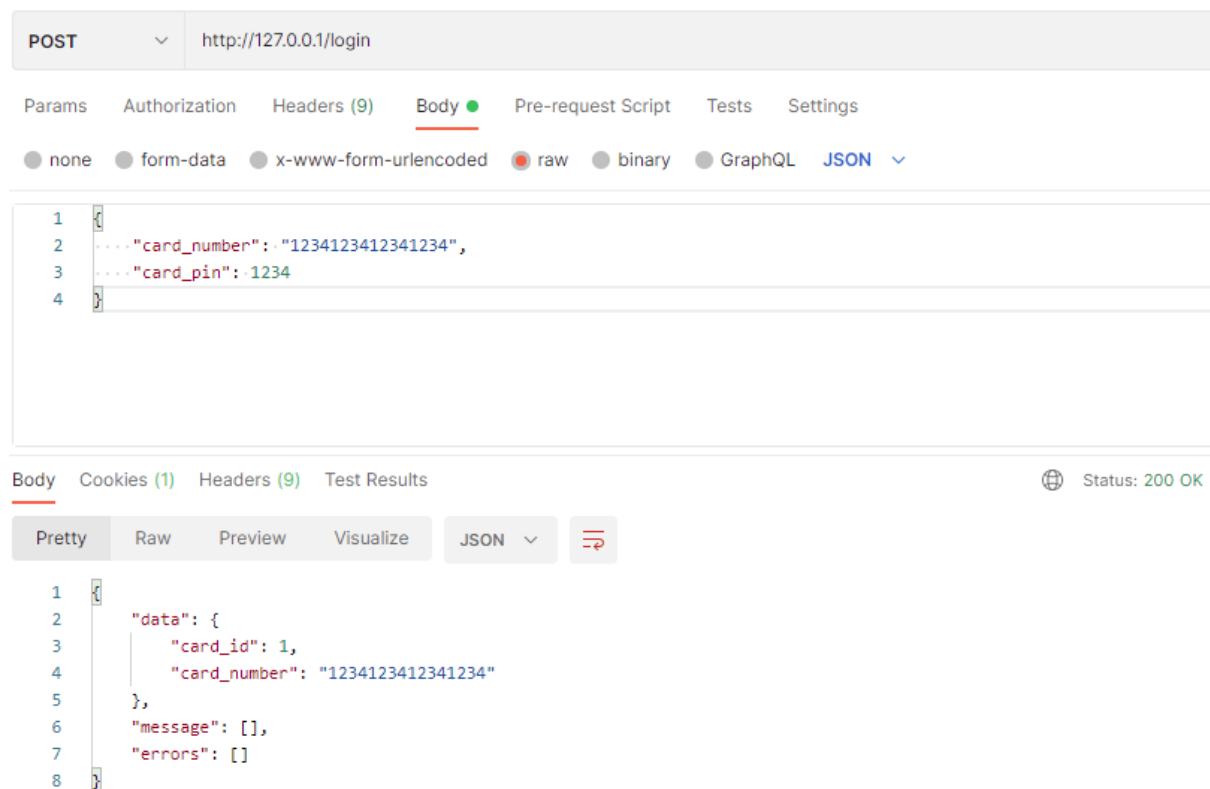
Example “build.sh” and “run.sh” files used during implementation of the application – here for bank-access-service.

To test if the MySQL database works correctly, the phpmyadmin image was used in the development process.



The phpmyadmin service started on localhost.

The endpoints were tested by the Postman application.



Successful login

POST http://127.0.0.1/login

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** ▼

```
1 {
2   ... "card_number": "1234123412341234",
3   ... "card_pin": 1235
4 }
```

Body Cookies Headers (8) Test Results 🌐 Status: 400 Bad Request

Pretty Raw Preview Visualize **JSON** ▼

```
1 {
2   "data": {},
3   "message": [],
4   "errors": [
5     "Card number or PIN not correct"
6   ]
7 }
```

Login – wrong credentials.

GET http://127.0.0.1/logout

Params Authorization Headers (7) **Body** Pre-request Script Tests Settings

Query Params

| | KEY | VALUE | DESCRIPTIC |
|--|-----|-------|-------------|
| | Key | Value | Description |

Body Cookies (1) Headers (8) Test Results 🌐 Status: 200 OK

Pretty Raw Preview Visualize **JSON** ▼

```
1 {
2   "data": {},
3   "message": [],
4   "errors": []
5 }
```

Successful logout

GET

▼

http://127.0.0.1/logout

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Query Params

| | KEY | VALUE | DESCRIPTIC |
|--|-----|-------|-------------|
| | Key | Value | Description |

Body Cookies (1) Headers (8) Test Results 🌐 Status: 401 Unauthorized

Pretty Raw Preview Visualize JSON ▼

```

1 {
2   "data": {},
3   "message": [],
4   "errors": []
5 }
```

Logout, when no user was logged in.

POST ▼ http://127.0.0.1/check-account

Params Authorization Headers (9) **Body** ● Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **Text** ▼

```

1 {}
```

Body Cookies (1) Headers (8) Test Results 🌐 Status: 200 OK

Pretty Raw Preview Visualize JSON ▼

```

1 {
2   "data": "986",
3   "message": [],
4   "errors": []
5 }
```

Successful check of the account balance.

POST http://127.0.0.1/check-account

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL Text

```
1 {}
```

Body Cookies (1) Headers (8) Test Results Status: 401 Unauthorized

Pretty Raw Preview Visualize JSON

```
1 {
2   "data": {},
3   "message": [],
4   "errors": []
5 }
```

Check the account balance - user not logged in.

POST http://127.0.0.1/withdraw

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {"withdraw_am": 20}
```

Body Cookies (1) Headers (8) Test Results Status: 400 Bad Request

Pretty Raw Preview Visualize JSON

```
1 {
2   "data": {},
3   "message": [],
4   "errors": [
5     "Not enough money to withdraw"
6   ]
7 }
```

Withdrawal – user has not enough money on his card.

POST http://127.0.0.1/withdraw

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1 {"withdraw_am": 2}
```

Body Cookies (1) Headers (8) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON

```
1 {
2   "data": 984,
3   "message": [],
4   "errors": []
5 }
```

Successful withdrawal.

POST http://127.0.0.1/withdraw

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```
1 {"withdraw_am": 2}
```

Body Cookies (1) Headers (8) Test Results Status: 401 Unauthorized

Pretty Raw Preview Visualize JSON

```
1 {
2   "data": {},
3   "message": [],
4   "errors": []
5 }
```

Withdrawal - user not logged in.

Later, in the following operations, the services were prepared in the 2nd version. In the 2nd version each returned response was slightly different, containing more information.

```

{
  "data": "180",
  "message": [
    "User validated by access service, account balance can be checked.",
    "Account balance on card 1234123412341234 for the day 2022-02-23T15:58:43.320Z is 180.",
    "Good results!"
  ],
  "errors": []
}

```

<----- access service - v2
 <----- reception service - v2
 <----- result service - v2

The response from successful check the account balance request, where the access, reception and result services were used in the 2nd version.

```

{
  "data": 94,
  "message": [
    "User validated by access service, withdrawal process can start.",
    "You have enough money to withdraw",
    "Confirmation: Withdrawal of 2euro on card 1234123412341234 on the day 2022-02-23T16:10:34.013Z",
    "Good results!"
  ],
  "errors": []
}

```

<----- access service - v2
 <----- reception service - v2
 <----- request processing service - v2
 <----- result service - v2

The response from successful withdrawal request, where the access, reception, request processing and result services were used in the 2nd version.

3.2 Deployment and routing

After preparation of the complete application in the first version, the next step was to deploy it using kubernetes and istio. Gateway and YAML files for each service were prepared. What is more, a “build-all.sh” and “run.sh” files were prepared, first file to build all the images at once and the second one to deploy all the YAML files at once. It automated the process.

```

services_v1 > build-all.sh
1  #!/bin/bash
2  #
3
4  cd bank-access-service
5  docker build -t bank-access-service-image .
6  cd ../bank-auth-service
7  docker build -t bank-auth-service-image .
8  cd ../bank-reception-service
9  docker build -t bank-reception-service-image .
10 cd ../bank-request-processing-service
11 docker build -t bank-request-processing-service-image .
12 cd ../bank-result-service
13 docker build -t bank-result-service-image .
14 cd ../bank-mysql-server
15 docker build -t bank-mysql-server-image .
16 cd ../bank-redis-server
17 docker build -t bank-redis-server-image .

```

“build-all.sh” file

```

v1 > run.sh
1  kubectl apply -f bank-access-service.yaml
2  kubectl apply -f bank-auth-service.yaml
3  kubectl apply -f bank-mysql-server.yaml
4  kubectl apply -f bank-reception-service.yaml
5  kubectl apply -f bank-redis-server.yaml
6  kubectl apply -f bank-request-processing-service.yaml
7  kubectl apply -f bank-result-service.yaml

```

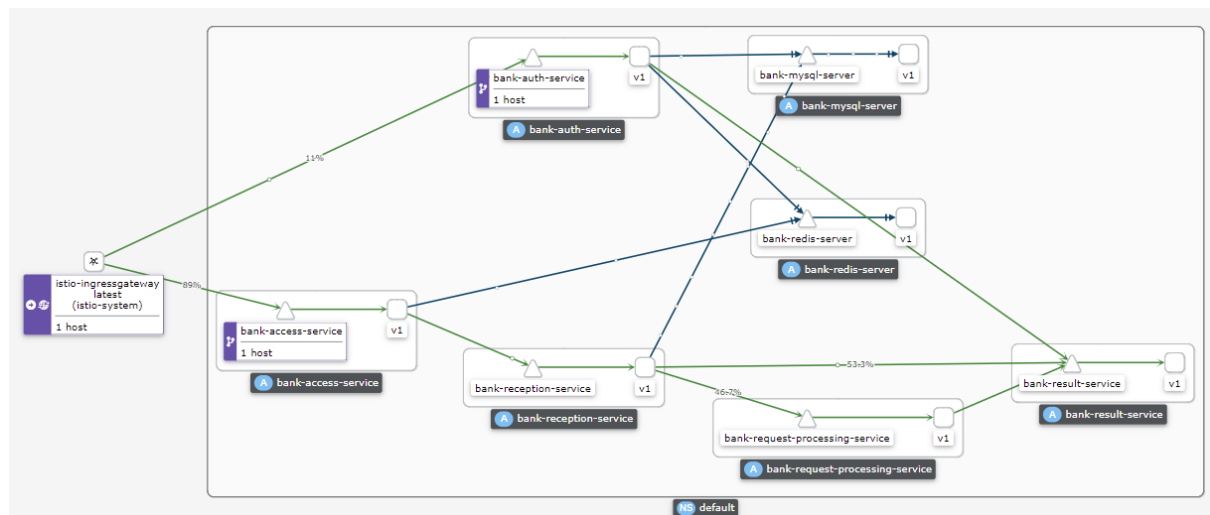
“run.sh” file

Then istio was downloaded and the first approach was to use minikube to set up a kubernetes cluster. It was because of no knowledge about the option to enable kubernetes in Docker Desktop applications. It took a lot of time because of some errors between the minikube tunnel and WSL2. Afterwards, after getting to know about Docker Desktop options, the approach was changed to use it instead. All the services were deployed successfully and istio dashboards were started to track the traffic.

```
tomasz@DESKTOP-HU4RFJM:~/app/v2$ kubectl get pods
```

| NAME | READY | STATUS | RESTARTS | AGE |
|---|-------|---------|-------------|-----|
| bank-access-service-v1-55499ddb8-7gctk | 2/2 | Running | 0 | 36m |
| bank-access-service-v1-55499ddb8-hh874 | 2/2 | Running | 0 | 36m |
| bank-access-service-v1-55499ddb8-kgdtm | 2/2 | Running | 0 | 36m |
| bank-auth-service-v1-786877df7f-5qzqq | 2/2 | Running | 3 (36m ago) | 36m |
| bank-auth-service-v1-786877df7f-6h8m8 | 2/2 | Running | 2 (36m ago) | 36m |
| bank-auth-service-v1-786877df7f-trhr1 | 2/2 | Running | 3 (36m ago) | 36m |
| bank-mysql-server-v1-99d869f77-5g4tk | 2/2 | Running | 0 | 36m |
| bank-reception-service-v1-59cbb98454-cd5kk | 2/2 | Running | 3 (36m ago) | 36m |
| bank-reception-service-v1-59cbb98454-d24xh | 2/2 | Running | 3 (36m ago) | 36m |
| bank-reception-service-v1-59cbb98454-gnhdh | 2/2 | Running | 2 (36m ago) | 36m |
| bank-redis-server-v1-668c58c5b4-5z1kr | 2/2 | Running | 0 | 36m |
| bank-request-processing-service-v1-7ffd689655-44899 | 2/2 | Running | 2 (36m ago) | 36m |
| bank-request-processing-service-v1-7ffd689655-w725d | 2/2 | Running | 2 (36m ago) | 36m |
| bank-request-processing-service-v1-7ffd689655-zsrq2 | 2/2 | Running | 2 (36m ago) | 36m |
| bank-result-service-v1-74c5cf4468-d7cl5 | 2/2 | Running | 0 | 36m |
| bank-result-service-v1-74c5cf4468-mppmn | 2/2 | Running | 0 | 36m |
| bank-result-service-v1-74c5cf4468-q6k6w | 2/2 | Running | 0 | 36m |

All the pods with deployed services of version one. As it was requested, all of them are having at least 3 instances (excluding database and cache system).

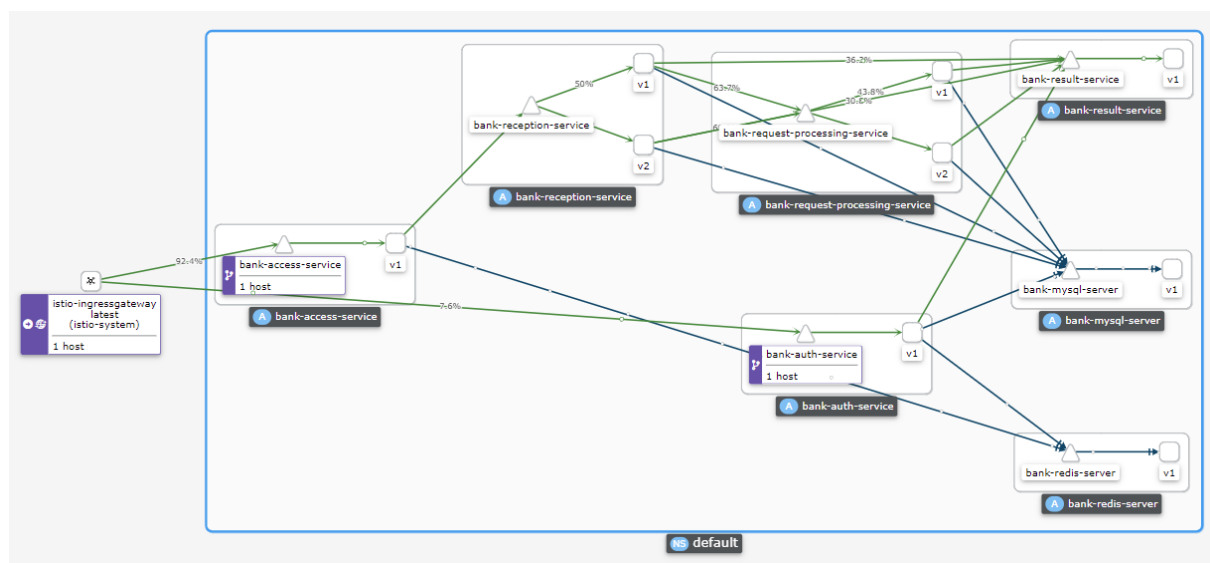


Visualization of the traffic between services in the first version. The visualization includes all the successful requests (it excludes not authorized and bad requests to make it more clear – all the next visualizations are shown in the same way).

Next step was to implement and deploy 2 services in version 2, with slightly different functions. The services bank-reception-service and bank-request-processing-service were duplicated and changed, so they return additional information about requests. The same gateway was used.

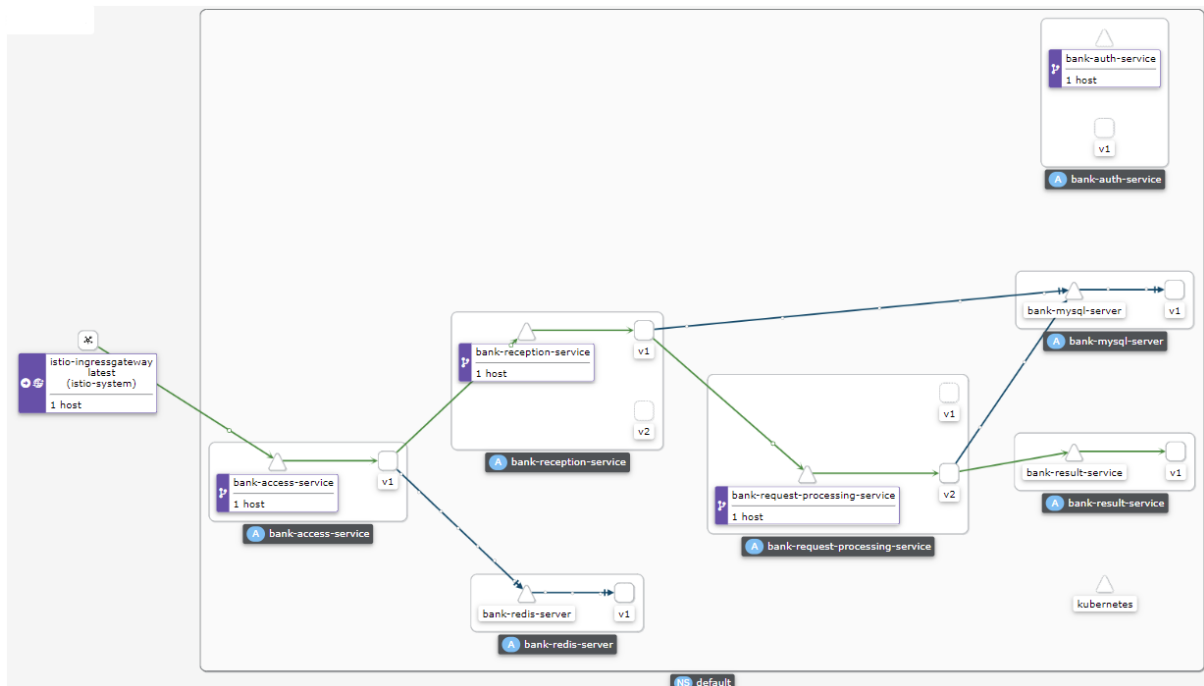
```
tomasz@DESKTOP-HU4RFJM:~/app/v2$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
bank-access-service-v1-55499ddbd8-7gctk    2/2     Running   0           57s
bank-access-service-v1-55499ddbd8-hh874    2/2     Running   0           57s
bank-access-service-v1-55499ddbd8-kgdtm    2/2     Running   0           57s
bank-auth-service-v1-786877df7f-5qzqq     2/2     Running   3 (34s ago) 57s
bank-auth-service-v1-786877df7f-6h8m8     2/2     Running   2 (45s ago) 57s
bank-auth-service-v1-786877df7f-trhrl     2/2     Running   3 (33s ago) 57s
bank-mysql-server-v1-99d869f77-5g4tk      2/2     Running   0           57s
bank-reception-service-v1-59cbb98454-cd5kk  2/2     Running   3 (34s ago) 57s
bank-reception-service-v1-59cbb98454-d24xh  2/2     Running   3 (33s ago) 57s
bank-reception-service-v1-59cbb98454-gnhdh  2/2     Running   2 (46s ago) 57s
bank-reception-service-v2-85967d9478-mjps5  2/2     Running   2 (36s ago) 43s
bank-reception-service-v2-85967d9478-mlbk2  2/2     Running   2 (37s ago) 43s
bank-reception-service-v2-85967d9478-pnn59  2/2     Running   2 (37s ago) 43s
bank-redis-server-v1-668c58c5b4-5zllkr    2/2     Running   0           56s
bank-request-processing-service-v1-7ffd689655-44899  2/2     Running   2 (46s ago) 56s
bank-request-processing-service-v1-7ffd689655-w725d  2/2     Running   2 (45s ago) 56s
bank-request-processing-service-v1-7ffd689655-zsrq2  2/2     Running   2 (45s ago) 56s
bank-request-processing-service-v2-55c7c465b5-72gzc  2/2     Running   2 (36s ago) 43s
bank-request-processing-service-v2-55c7c465b5-7skdc  2/2     Running   2 (36s ago) 43s
bank-request-processing-service-v2-55c7c465b5-x4jfx  2/2     Running   2 (36s ago) 43s
bank-result-service-v1-74c5cf4468-d7cl5    2/2     Running   0           56s
bank-result-service-v1-74c5cf4468-mppmn    2/2     Running   0           56s
bank-result-service-v1-74c5cf4468-q6k6w    2/2     Running   0           56s
```

All the pods with deployed services of version one and 2 services of version 2.



Visualization of the traffic between services in the first version and 2 services of version 2.

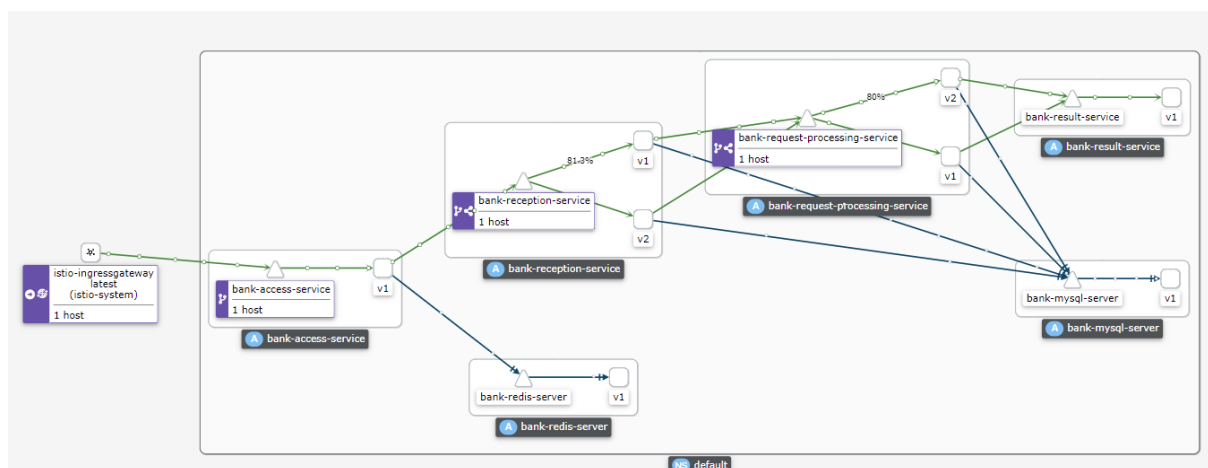
Next step was to prepare routing to specific nodes of each service. To do so, first a destination rule file was prepared to define the available versions, called subsets. Then, a virtual service file was prepared. For now only 2 services are deployed in 2 different versions, which are bank-reception-service and bank-request-processing-service. The prepared virtual service file is routing all the traffic to version 1 of reception service and to version 2 of request processing service.



Visualization of the traffic between services after applying virtual service, which routes the traffic to version 1 of reception service and to version 2 of request processing service.

All of the traffic is routed properly.

Next operation to be carried out was to direct a particular percentage of traffic to a new version of a service as part of A/B testing. In the last step 100% of the traffic was routed to version 1 of reception service and to version 2 of request processing service. This time, 20% of that traffic is redirected to another version of the service. A Virtual service file was prepared, which uses weights to do the task.



Visualization of the traffic between services after applying virtual service, which routes 80% of the traffic to version 1 of reception service and 20% of the traffic to version 2. For request processing service, 80% is routed to version 2 and 20% to version 1.

As can be seen on the visualization, the routing works well, with small error for bank-reception-service, where 81.3% of the traffic is passed to version 1. It can be because of not enough requests being sent.

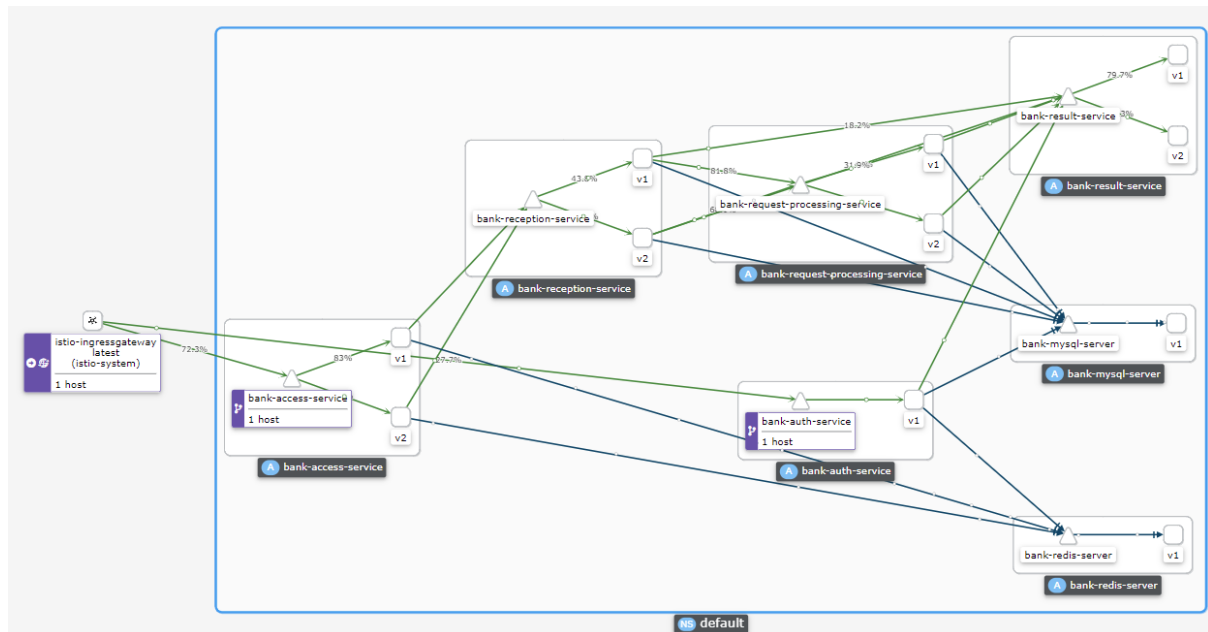
Next step to be carried out was to prepare routing by version. Two virtual service files were prepared. In the first one all the services are set to version 1, in another one all the services are set to version 2. The results were similar to the results of the operations carried out before. After applying the first file, all the services worked in the 1st version. Because only bank-reception-service and bank-request-processing-service were prepared in 2 versions at that point, in the 2nd file only these services were set to work in the 2nd version.

Next step was a deployment of version 2 of the remaining services (excluding auth service, which was not requested). The services bank-access-service and bank-result-service were duplicated and changed, so they return additional information about requests.

```
tomasz@DESKTOP-HU4RFJM:~/app/v2$ kubectl get pods
```

| NAME | READY | STATUS | RESTARTS | AGE |
|---|-------|---------|---------------|-----|
| bank-access-service-v1-55499ddb8-7gctk | 2/2 | Running | 2 (4h30m ago) | 18h |
| bank-access-service-v1-55499ddb8-hh874 | 2/2 | Running | 2 (4h30m ago) | 18h |
| bank-access-service-v1-55499ddb8-kgdtm | 2/2 | Running | 2 (4h30m ago) | 18h |
| bank-access-service-v2-7b5d7cb4f4-mzjc2 | 2/2 | Running | 0 | 35s |
| bank-access-service-v2-7b5d7cb4f4-p2h46 | 2/2 | Running | 0 | 35s |
| bank-access-service-v2-7b5d7cb4f4-sxnpw | 2/2 | Running | 0 | 35s |
| bank-auth-service-v1-786877df7f-5qzqq | 2/2 | Running | 6 (4h29m ago) | 18h |
| bank-auth-service-v1-786877df7f-6h8m8 | 2/2 | Running | 5 (4h29m ago) | 18h |
| bank-auth-service-v1-786877df7f-trhrl | 2/2 | Running | 7 (4h29m ago) | 18h |
| bank-mysql-server-v1-99d869f77-5g4tk | 2/2 | Running | 2 (4h30m ago) | 18h |
| bank-reception-service-v1-59cbb98454-cd5kk | 2/2 | Running | 7 (4h29m ago) | 18h |
| bank-reception-service-v1-59cbb98454-d24xh | 2/2 | Running | 6 (4h29m ago) | 18h |
| bank-reception-service-v1-59cbb98454-gnhdh | 2/2 | Running | 6 (4h29m ago) | 18h |
| bank-reception-service-v2-85967d9478-489zp | 2/2 | Running | 4 (4h29m ago) | 17h |
| bank-reception-service-v2-85967d9478-6pw8v | 2/2 | Running | 5 (4h29m ago) | 17h |
| bank-reception-service-v2-85967d9478-zt24z | 2/2 | Running | 5 (4h29m ago) | 17h |
| bank-redis-server-v1-668c58c5b4-5z1kr | 2/2 | Running | 2 (4h30m ago) | 18h |
| bank-request-processing-service-v1-7ffd689655-44899 | 2/2 | Running | 5 (4h29m ago) | 18h |
| bank-request-processing-service-v1-7ffd689655-w725d | 2/2 | Running | 5 (4h29m ago) | 18h |
| bank-request-processing-service-v1-7ffd689655-zsrq2 | 2/2 | Running | 6 (4h29m ago) | 18h |
| bank-request-processing-service-v2-55c7c465b5-bdb6c | 2/2 | Running | 5 (4h29m ago) | 17h |
| bank-request-processing-service-v2-55c7c465b5-s9525 | 2/2 | Running | 5 (4h29m ago) | 17h |
| bank-request-processing-service-v2-55c7c465b5-wld6m | 2/2 | Running | 4 (4h29m ago) | 17h |
| bank-result-service-v1-74c5cf4468-d7cl5 | 2/2 | Running | 2 (4h30m ago) | 18h |
| bank-result-service-v1-74c5cf4468-mppmn | 2/2 | Running | 2 (4h30m ago) | 18h |
| bank-result-service-v1-74c5cf4468-q6k6w | 2/2 | Running | 2 (4h30m ago) | 18h |
| bank-result-service-v2-d997fd55b-crkmt | 2/2 | Running | 0 | 34s |
| bank-result-service-v2-d997fd55b-lxgp9 | 2/2 | Running | 0 | 34s |
| bank-result-service-v2-d997fd55b-pc9w4 | 2/2 | Running | 0 | 34s |

All the pods with deployed services of version 1 and version 2.



Visualization of the traffic between all of the services in both versions.

Last step was to rollback from version 2 to version 1 of the services affected by the previous step. To do it, the deployment files for bank-access-service and bank-result-service were deleted, using a prepared “.sh” file.

```
delete-2-acc-res.sh
kubectl delete -f bank-access-service-v2.yaml
kubectl delete -f bank-result-service-v2.yaml
```

File used to rollback from version 2 to version 1 for bank-access-service and bank-result-service.

Afterwards, because the version 2 of these services didn't make any external changes, there was no problem to rollback to version 1.

3. Traffic automation for testing

The process of testing the application was gradually automated during work.

First operation that was carried out during work was implementation of the application. During this process the testing was already partly automated, which is described in the beginning of the point about application implementation (**3.1 Application implementation**).

Next step of automation was done during the next operation – deployment of the version 1 of all services. It is described in the beginning of the deployment and routing point (**3.2 Deployment and routing**).

The files automating operations:

- /services_v1/build_all.sh – building all the images of the services in version 1

- /services_v2/build-2.sh – building first 2 images of the services in version 2 (bank-reception-service and bank-request-processing-service)
- /services_v2/build-all.sh – building all the images of the services in version 2
- /v1/run.sh – deploying all the services of version 1 in kubernetes cluster
- /v1/delete.sh – deleting all the services of version 1 from kubernetes cluster
- /v2/run-2-rec-req.sh deploying first 2 services of version 2 in kubernetes cluster (bank-reception-service and bank-request-processing-service)
- /v2/run.sh – deploying all the services of version 2 in kubernetes cluster
- /v2/delete-2-acc-res.sh deleting 2 services of version 2 from kubernetes cluster (bank-access-service and bank-result-service) – use to rollback for the last operation
- /v2/delete.sh – deleting all the services of version 2 from kubernetes cluster

Finally, “istio_deploy_v1.sh” file was prepared, which automates all the commands needed to deploy services in version one using kubernetes and istio. File includes:

- istio downloading and configuration,
- build of all of the services in version 1,
- deployment of gateway,
- deployment of the services in kubernetes,
- scan for validation issues by istioctl analyze
- set of IP and PORTS (works if an environment supports external load balancers)
- return of gateway url, where the application is available.

The file requires application files to be locally saved in the “app” directory. It should be started out of the “app” directory.

4.Weknesses of the work done

Because of lack of time the frontend application is not developed. However, the use of the services can be checked by Postman application, by testing endpoints. Even though the functionality of the application is sufficient to fulfill all the requirements, it is really simple and should be extended. The second versions of the services could be functioning in a more different way. Now the difference is only that the second versions are providing additional information for the user.