NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Project Assignment #2

SCC 2024/2025

David Castro 60973

Yaroslav Hayduk 60739

10th December

Implementation

Solution used

For our implementation we used the initial *Tukano* base code provided by the professors as we felt it would be a simpler and more straightforward solution to implement. The blobs authentication was done using the same blueprint and convention as the other endpoints, consisting of the following classes:

- *Auth*
- *RestAuth*
- *JavaAuth*
- *RestAuthResource*

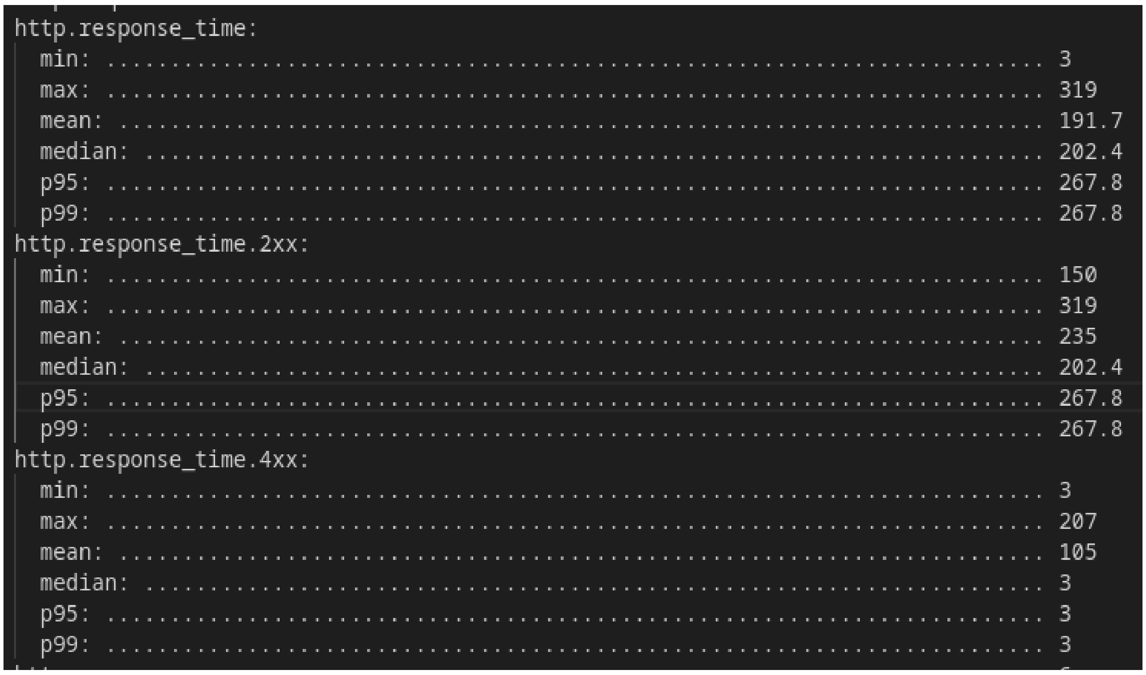
In addition there are also the *RequestCookies* classes given in the *lab9* in the *auth* folder under *impl*.

Kubernetes deployment

Our kubernetes deployment files are under the *k8s* folder in the main directory. Our implementation consists of the *Tukano* web app, *PostgreSQL* database, *Redis* cache and *MinIO* for the blob storage management, having used each corresponding latest image for their containers. We opted to go with the simplest possible solution, so our architecture doesn't become overly complicated, starting with the *Tukano*, we have a Deployment where we define all the environment variables we use in our application, the image we use, pulled from our docker repository, and the Service with the LoadBalancer/NodePort. For our cache, we have an even simpler setup, with the Deployment that uses the Redis image and defines the desired resources, and the Service of type ClusterIP. For the PostgreSQL database we use a PersistentVolumeClaim to ensure persistence of the data we store, our Deployment that identifies the image to use, `postgres:latest`, defines the necessary environment variables likes user, password and database name, and defines the persistent volume that's going to communicate with, and the Service, also of type ClusterIP. Finally we have the MinIO Deployment which we use for managing the blob storage. Here we define the image to be used and some args and environment variables necessary for the correct execution of the container. Obviously we also have a defined PersistentVolumeClaim and the Service of type NodePort with all the port configurations.

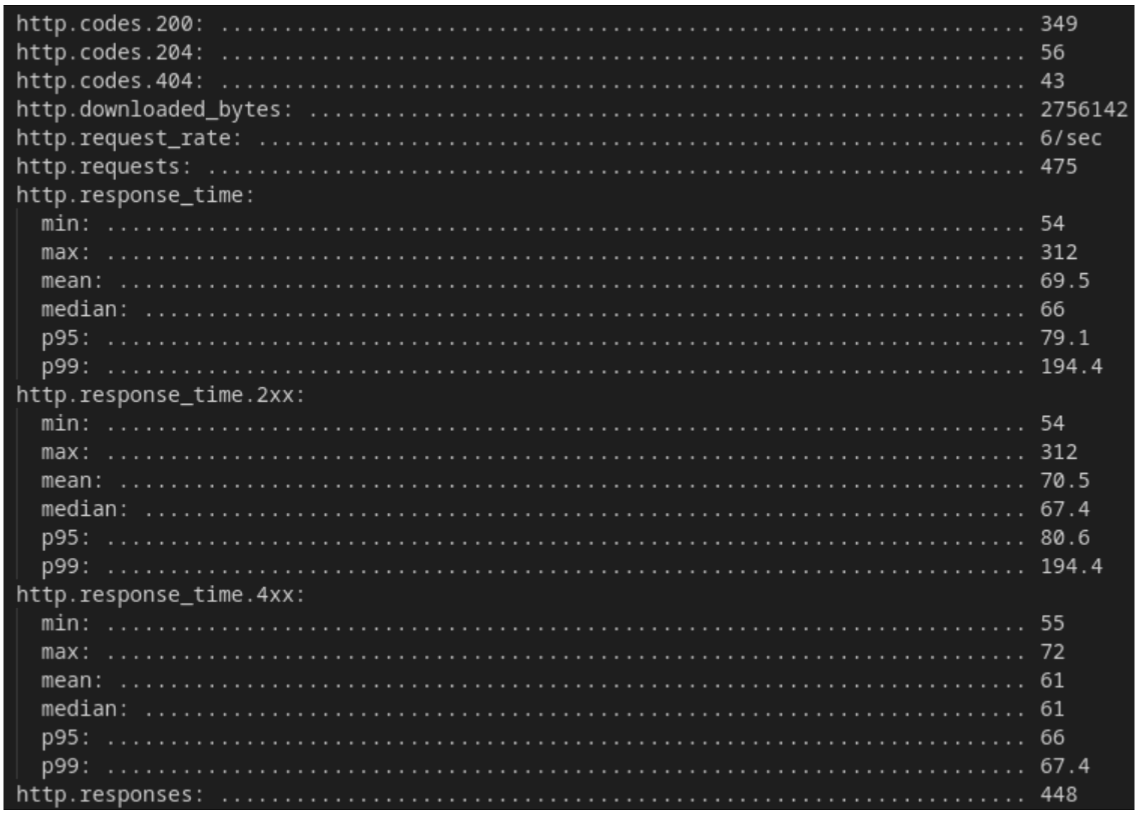
Performance evaluation

In our previous project we opted to use the professors' provided tests so, to maintain some consistency and more easily compare both results, we will be reusing them. Previously, running our realistic flow test we got the following result:



```
http.response_time:
  min: ...................................................................... 3
  max: ...................................................................... 319
  mean: ..................................................................... 191.7
  median: ................................................................... 202.4
  p95: ...................................................................... 267.8
  p99: ...................................................................... 267.8
http.response_time.2xx:
  min: ...................................................................... 150
  max: ...................................................................... 319
  mean: ..................................................................... 235
  median: ................................................................... 202.4
  p95: ...................................................................... 267.8
  p99: ...................................................................... 267.8
http.response_time.4xx:
  min: ...................................................................... 3
  max: ...................................................................... 207
  mean: ..................................................................... 105
  median: ................................................................... 3
  p95: ...................................................................... 3
  p99: ...................................................................... 3
```

Running it now with our current setup the results are as shown below:



```
http.codes.200: ................................................................ 349
http.codes.204: ................................................................ 56
http.codes.404: ................................................................ 43
http.downloaded_bytes: ......................................................... 2756142
http.request_rate: ............................................................. 6/sec
http.requests: ................................................................. 475
http.response_time:
  min: ......................................................................... 54
  max: ......................................................................... 312
  mean: ........................................................................ 69.5
  median: ...................................................................... 66
  p95: ......................................................................... 79.1
  p99: ......................................................................... 194.4
http.response_time.2xx:
  min: ......................................................................... 54
  max: ......................................................................... 312
  mean: ........................................................................ 70.5
  median: ...................................................................... 67.4
  p95: ......................................................................... 80.6
  p99: ......................................................................... 194.4
http.response_time.4xx:
  min: ......................................................................... 55
  max: ......................................................................... 72
  mean: ........................................................................ 61
  median: ...................................................................... 61
  p95: ......................................................................... 66
  p99: ......................................................................... 67.4
http.responses: ................................................................ 448
```

Conclusion

Through the examination of the results and comparison against the results obtained during the first assignment, we can conclude that the current deployment of the application is much more efficient and faster than the last one. Even though the values for min and max response times seem similar, we can observe a drastic difference in the mean and median times, they got reduced by half or more.
This observation leads to two possible conclusions:
1- We didn't properly set up the Redis Cache in the first assignment (which we already suspected).
2- We allocated more resources for the kubernetes deployment then we did in the previous assignment, namely more CPU and RAM for the containers that we probably needed.

Notes:

- We choose to use the minIO as a storage solution because in the assignment page it was written that “The solution can leverage existing docker images, already found in Docker Hub, or customized versions of those.”. It's pretty much a regular filesystem but just optimized for cloud architectures, like kubernetes and uses the S3 (Amazon) API which makes it accessible via REST request.
- The use of AI was mostly related to the S3 storage class and debugging the postegres Pod, given that it would randomly crash on deployment, giving the *CrashLoopBackOff* error.