SCC Report 2

Azure Tukano with Kubernetes – 2024/2025

Project and Report done by:

João Custódio - Nº 71736

Henrique Oliveira – Nº 70242

Introduction

In this project, we ported the previous project solution from the base tukano implementation to Kubernetes and added access control for accessing the blobs storage to logged in users. To do so, we used Kubernetes containers to run each of the 4 pods, where each pod contains a different service.

In terms of requirements for this work, the project meets all of the requirements established for the mandatory fields, plus the redis cache implemented in kubernetes present in the optional requirements.

Just like the previous project, it is possible to disable and enable the use of caching in the “utils.DB.java” boolean switch present in this class. For this application to work, maintain the boolean switch “usePostegre” ‘true’, as this project does not support the use of a noSQL implementation. Finally, in the JavaBlobs class, it is possible to find a boolean switch called “useAuth” that should remain ‘true’ for the correct use of cookies and authentication mechanisms.

The file “kubernetes_file.yaml” represents the file used to create the kubernetes cluster. Please change some of the environmental variables present in this file so that the correct DNS url is given to the system (example: in BLOBS_SERVICE_URL in the users-shorts deployment, the correct URL/IP to the blobs pod should be given, and should always end in “/tukano-1/rest”)

Kubernetes

To port this project to Kubernetes we used the help of 4 different pods, these being the Users & Shorts Pod (responsible for all the operations that the /users and /shorts have, with an externally accessible IP), the PostgreDB pod (with an internal ip), the Blobs Pod (responsible for all the operations that the /blobs have and with user access control, with an externally accessible IP) and finally, a Redis Cache Pod (with an internal ip).

Users & Shorts Pod

The Users & Shorts pod is responsible for all the operations inside the /users & /shorts. For these operations to be successfully executed, this pod communicates internally with the PostgreDB pod to be able to read and write data to that database. In addition to that, this pod also communicates internally with the Redis Cache Pod to be able to use the cache services. Finally, this pod also uses HTTP requests to

the Blobs Pod to be able to execute some operations, being that the reason that is crucial is to give the correct DNS/Ip Address of the Blobs Pod to the environmental variables of this pod.

Postgre Database Pod

This pod uses a simple postgre image container to execute a database. It is used to store the data used in this application. It is only internally accessible.

Blobs Pod

This pod is responsible for all the operations inside the /blobs and for access control to the same methods. It uses persistent file system storage and also uses HTTP requests to be able to communicate with the Users & Shorts pod being crucial to give the correct addresses in this pod's environmental variables in the kubernetes file.

Redis Pod

This pod uses a simple redis image container to execute a redis cache. It is only internally accessible.

Access Control

Finally, in this project we implemented access control with the use of cookies to access some methods of the blobs pod. To do so, only logged in users can access the 'upload' and 'download' methods, and only Admin users can access the 'delete' method.

To do so, a user needs to use the operation 'login' in the blobs pod to be able to be given the cookie. For that, the user needs to insert their username and password, that will later be checked to see if it matches with a user that exists in the system (using the Users & Shorts pod 'getUser' method), and if so, the user receives their

cookie and the cookie will be stored internally. After that, the user can do the other 2 operations regularly as the system knows if the user has the cookie with them or not.

Finally, if the user wants to delete a blob, they need to have an admin cookie. To do so, they need to login as username ‘admin’ and password ‘admin’ and they will have access to the delete blob method.

Performance Analyses

Tukano Base

The following images represent some prints taken from the Base Tukano application that were locally tested so that we can later compare them to other tests of the full application:

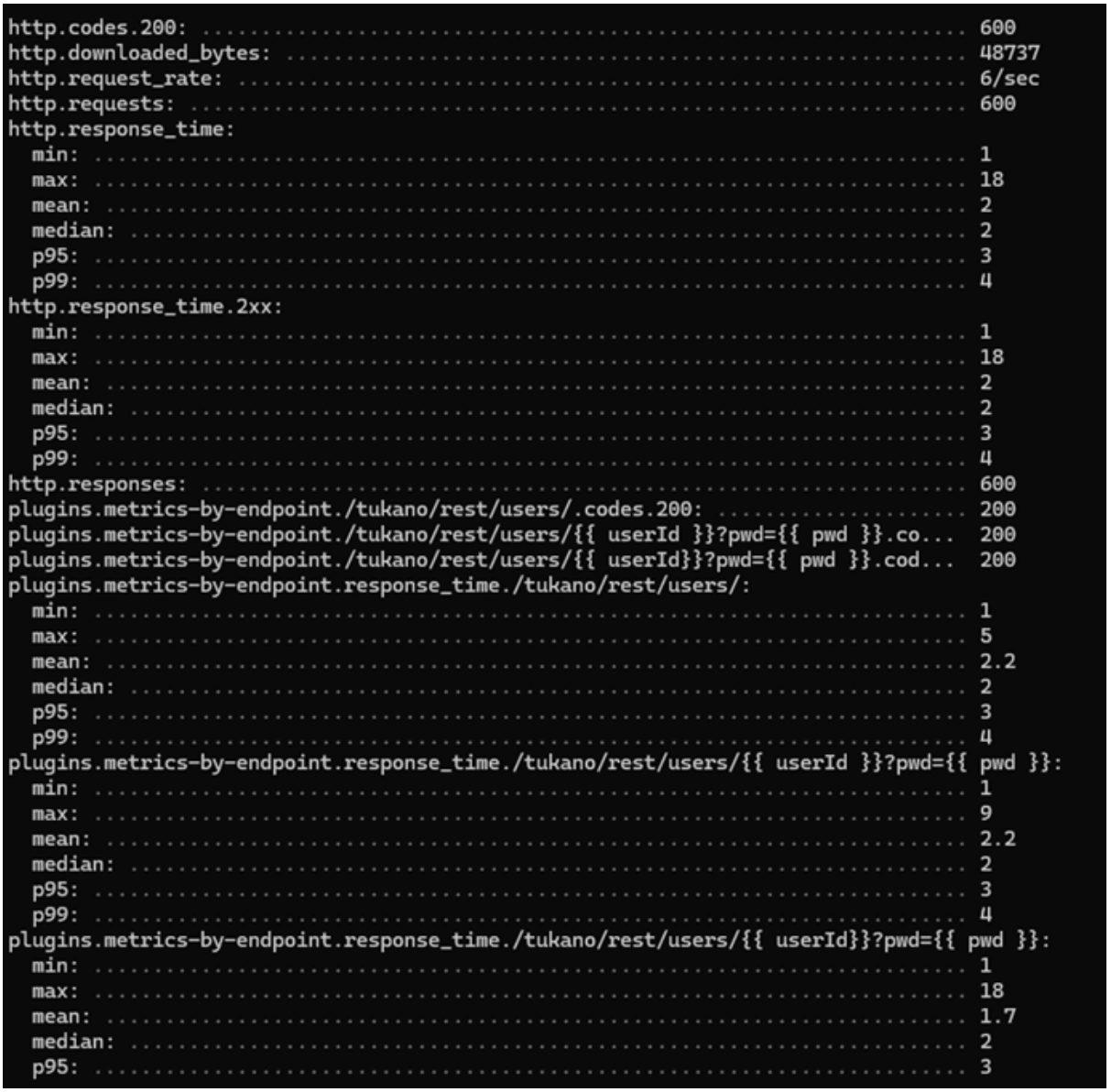

Fig 1. Artillery test ‘user_register.yaml’ run with the base tukano app

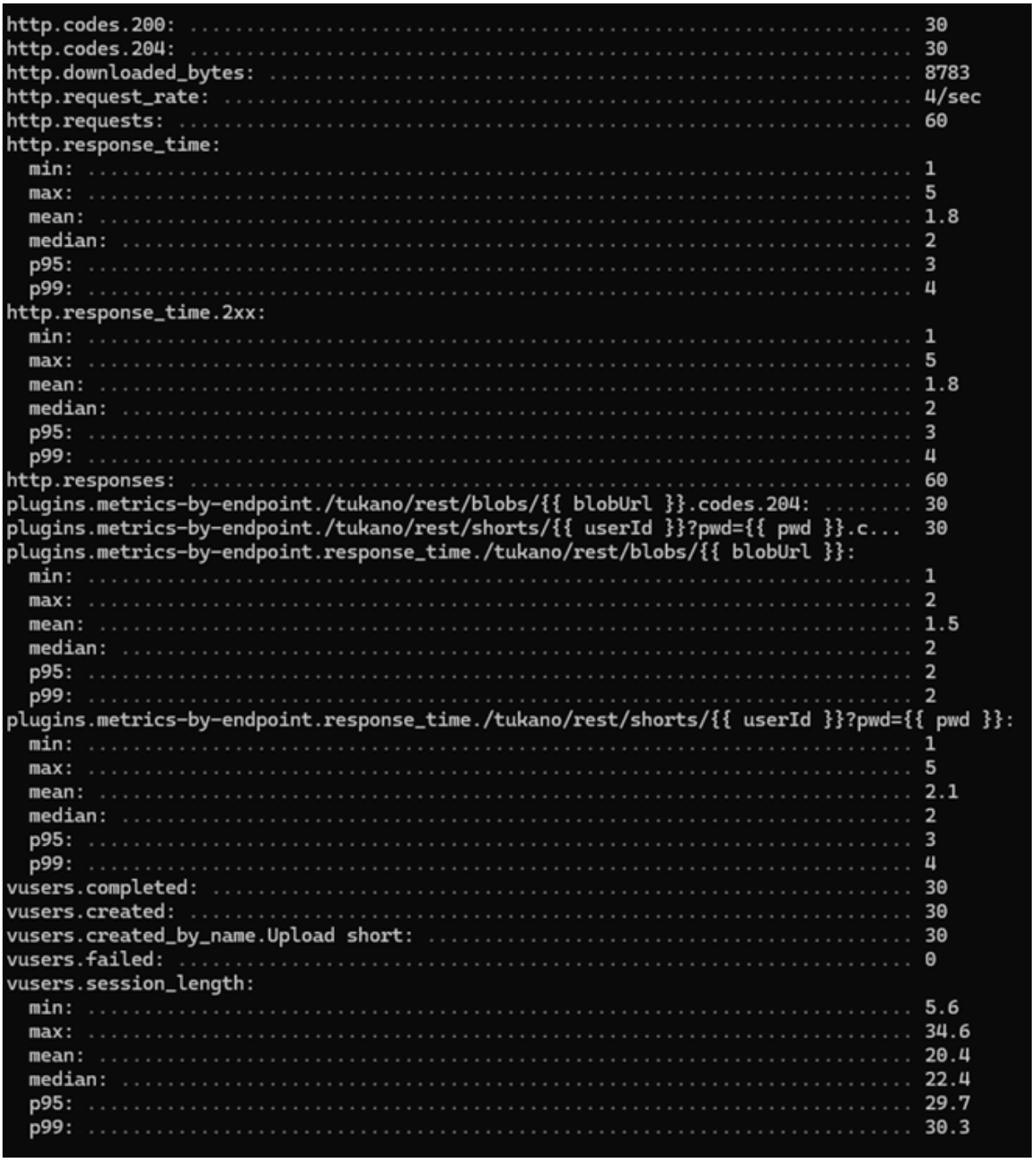

Fig 2. Artillery test ‘upload_shorts.yaml’ for Base Tukano

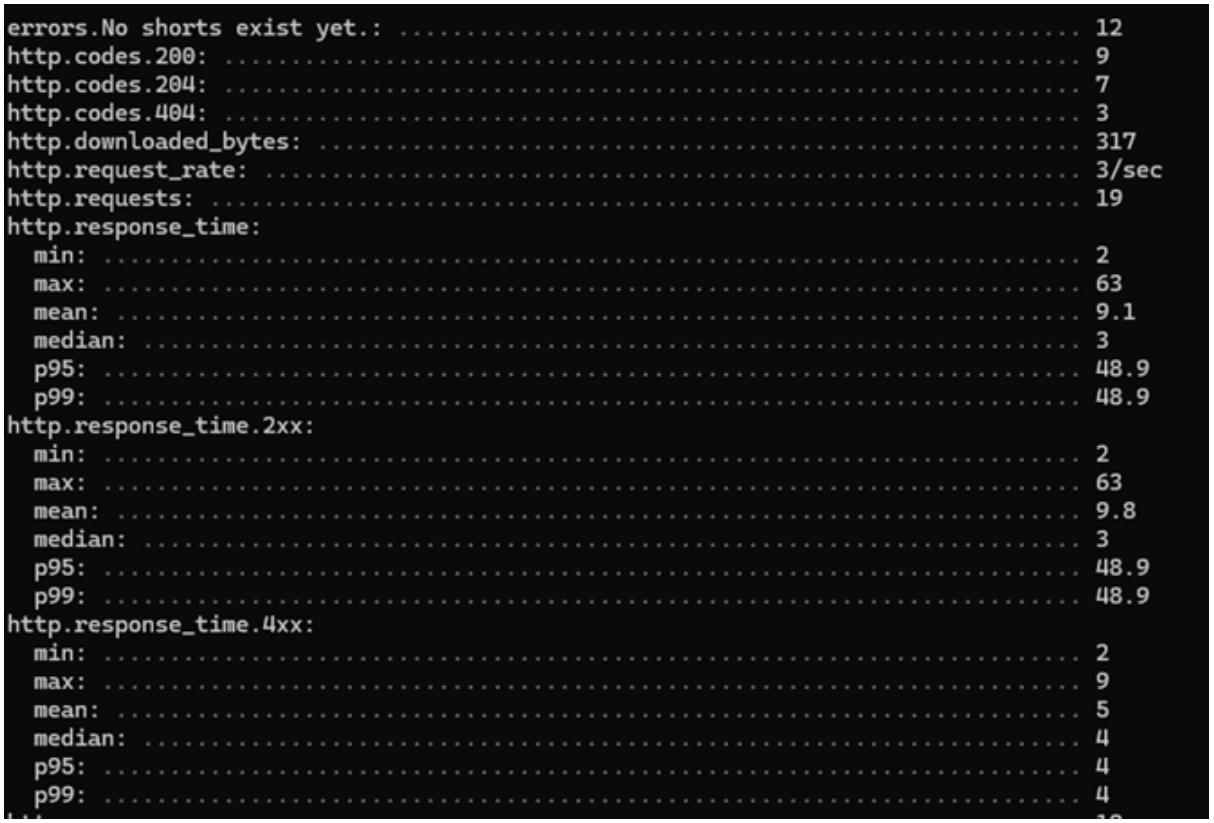

Fig 3. Artillery test ‘realistic_flow.yaml’ for Base Tukano Response Time

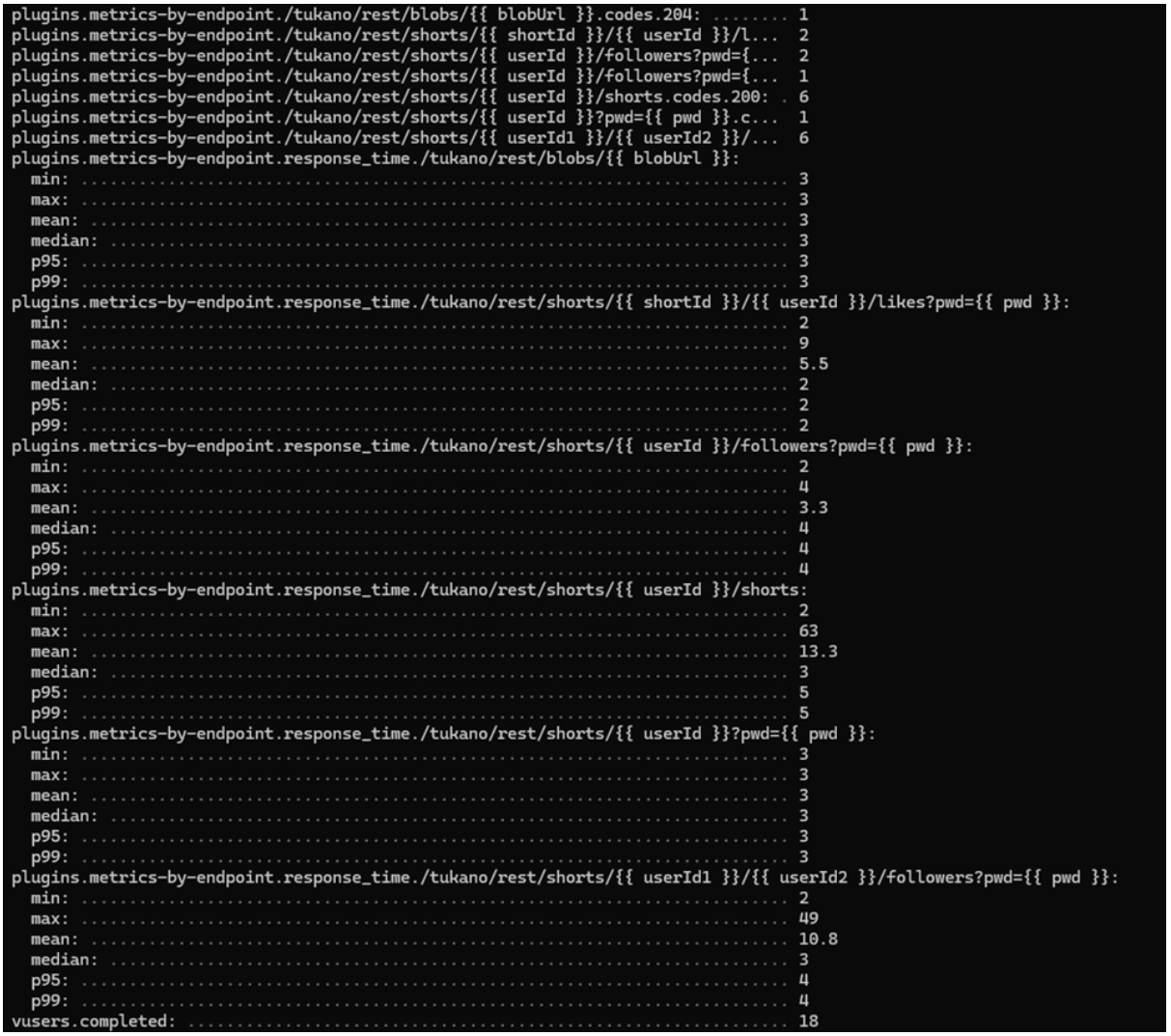

Fig 4. Artillery test ‘realistic_flow.yaml’ for Base Tukano Individual Request Time

Tukano Kubernetes without Cache

Compared to base Tukano, the first thing that is immediately noticeable is that the HTTP response times are longer, though this isn’t due to the Kubernetes implementation. This is simply due to the fact we tested base Tukano *locally*, and the Kubernetes version was deployed to the cloud.

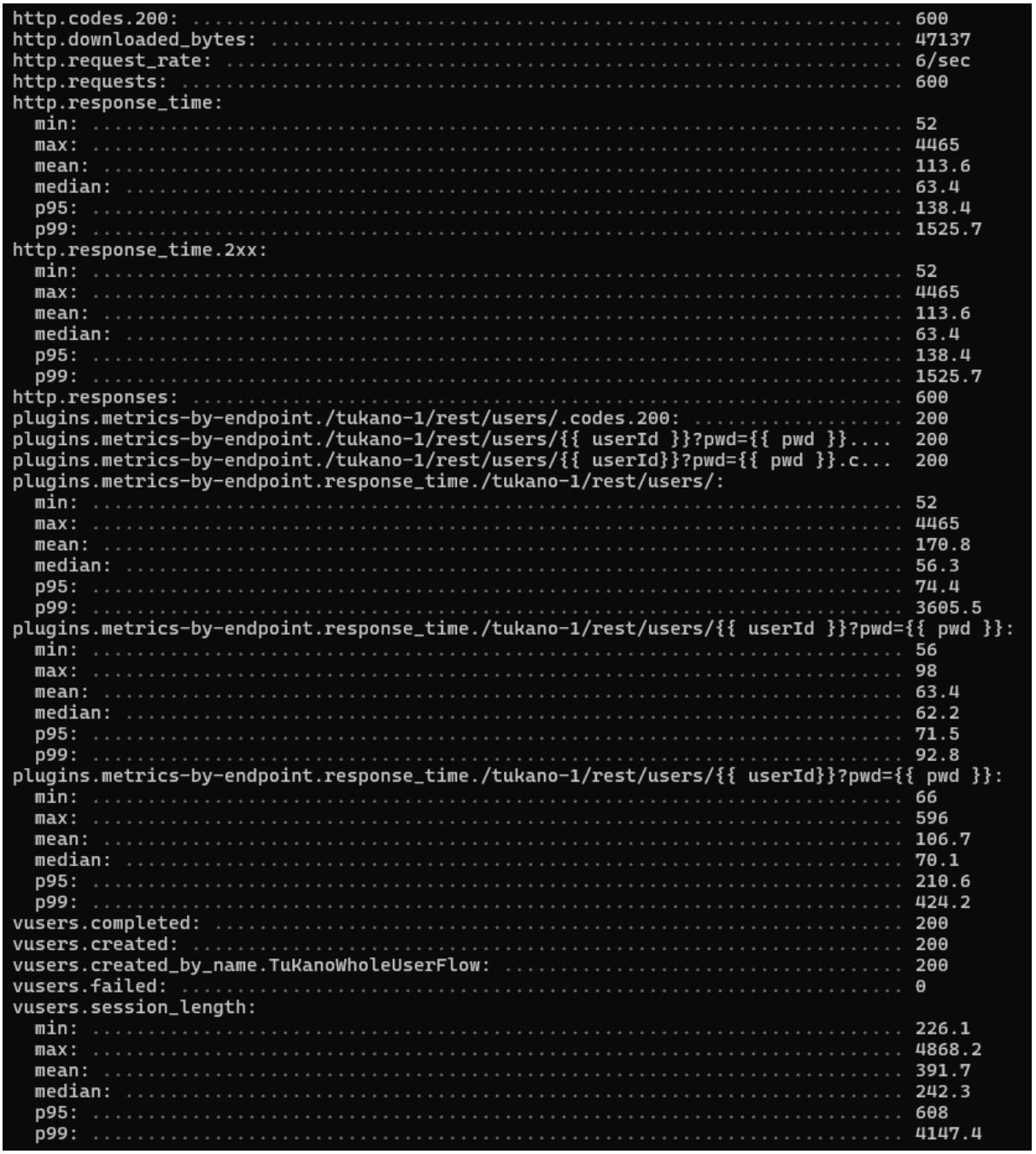

Fig 5. Artillery test ‘user_register.yaml’ run with the tukano kubernetes without cache

```
http.codes.200: ..... 30
http.codes.204: ..... 30
http.downloaded_bytes: ..... 9707
http.request_rate: ..... 7/sec
http.requests: ..... 60
http.response_time:
  min: ..... 65
  max: ..... 301
  mean: ..... 119.7
  median: ..... 135.7
  p95: ..... 179.5
  p99: ..... 186.8
http.response_time.2xx:
  min: ..... 65
  max: ..... 301
  mean: ..... 119.7
  median: ..... 135.7
  p95: ..... 179.5
  p99: ..... 186.8
http.responses: ..... 60
plugins.metrics-by-endpoint.proj2-71736-70242-blobs.northeurope.cloudapp.azu... 30
plugins.metrics-by-endpoint.proj2-71736-70242-users-shorts.northeurope.cloud... 30
plugins.metrics-by-endpoint.response_time.proj2-71736-70242-blobs.northeurope.cloudapp.azure.com/tukano-1/rest/blobs/{{ blobUrl }}:
  min: ..... 92
  max: ..... 301
  mean: ..... 143.9
  median: ..... 138.4
  p95: ..... 183.1
  p99: ..... 186.8
plugins.metrics-by-endpoint.response_time.proj2-71736-70242-users-shorts.northeurope.cloudapp.azure.com/tukano-1/rest/shorts/{{ userId }}?pwd={{ pwd }}:
  min: ..... 65
  max: ..... 147
  mean: ..... 95.4
  median: ..... 70.1
  p95: ..... 141.2
  p99: ..... 141.2
vusers.completed: ..... 30
vusers.created: ..... 30
vusers.created_by_name.Upload short: ..... 30
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 255.4
  max: ..... 658.6
  mean: ..... 342.7
  median: ..... 333.7
  p95: ..... 391.6
  p99: ..... 415.8
```

Fig 6. Artillery test ‘upload_shorts.yaml’ for Tukano Kubernetes Without Cache

```
http.response_time:
  min: ..... 45
  max: ..... 164
  mean: ..... 95.1
  median: ..... 70.1
  p95: ..... 156
  p99: ..... 162.4
http.response_time.2xx:
  min: ..... 55
  max: ..... 164
  mean: ..... 104.1
  median: ..... 77.5
  p95: ..... 147
  p99: ..... 147
```

Fig 7. Artillery test ‘realistic_flow.yaml’ for Tukano Kubernetes Without Cache Average Respose Time

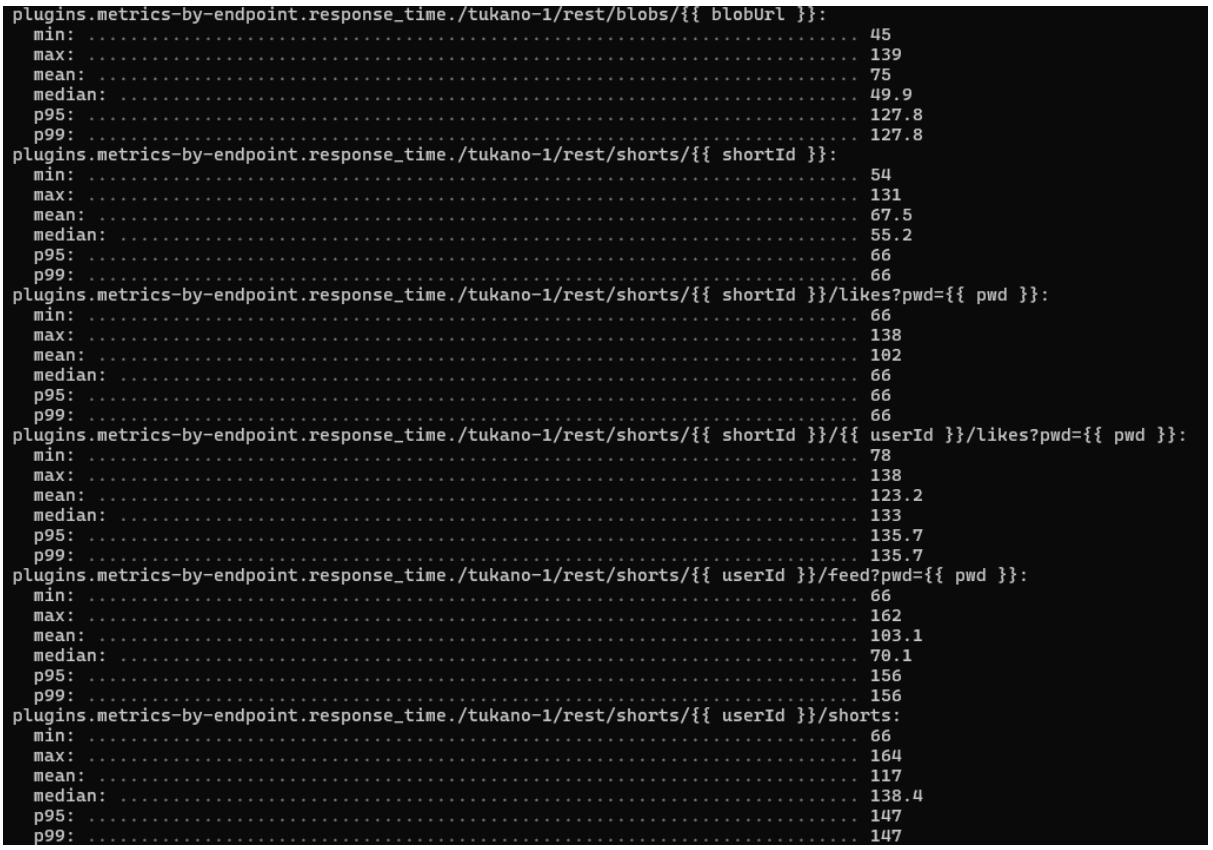

Fig 8. Artillery test ‘realistic_flow.yaml’ for Tukano Kubernetes Without Cache Response Time per Operation

Tukano Kubernetes with Cache

By comparing the following results, we can see that the addition of the Redis Cache severely improved results, drastically lowering response times (somewhat contradictory to project phase 1 where the cache actually worsened response times).

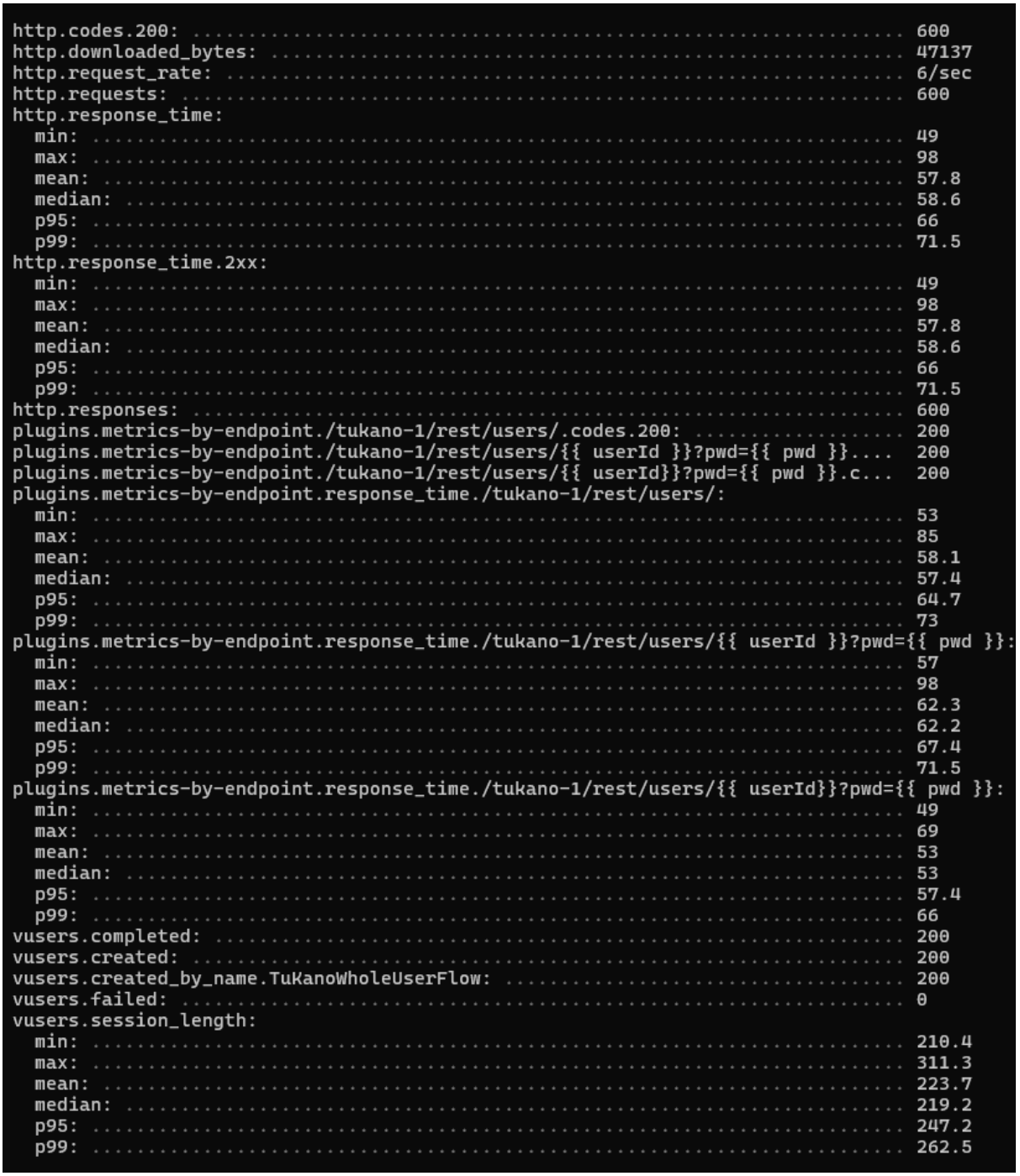

Fig 9. Artillery test ‘user_register.yaml’ run with the tukano kubernetes with cache

```
http.codes.200: ..... 30
http.codes.204: ..... 30
http.downloaded_bytes: ..... 9674
http.request_rate: ..... 6/sec
http.requests: ..... 60
http.response_time:
  min: ..... 59
  max: ..... 192
  mean: ..... 114.1
  median: ..... 133
  p95: ..... 141.2
  p99: ..... 190.6
http.response_time.2xx:
  min: ..... 59
  max: ..... 192
  mean: ..... 114.1
  median: ..... 133
  p95: ..... 141.2
  p99: ..... 190.6
http.responses: ..... 60
plugins.metrics-by-endpoint.proj2-71736-70242-blobs.northeurope.cloudapp.azu... 30
plugins.metrics-by-endpoint.proj2-71736-70242-users-shorts.northeurope.cloud... 30
plugins.metrics-by-endpoint.response_time.proj2-71736-70242-blobs.northeurope.cloudapp.azure.com/tukano-1/rest/blobs/{{ blobUrl }}:
  min: ..... 90
  max: ..... 140
  mean: ..... 133.9
  median: ..... 135.7
  p95: ..... 138.4
  p99: ..... 138.4
plugins.metrics-by-endpoint.response_time.proj2-71736-70242-users-shorts.northeurope.cloudapp.azure.com/tukano-1/rest/shorts/{{ userId }}?pwd={{ pwd }}:
  min: ..... 59
  max: ..... 192
  mean: ..... 94.2
  median: ..... 70.1
  p95: ..... 183.1
  p99: ..... 190.6
vusers.completed: ..... 30
vusers.created: ..... 30
vusers.created_by_name.Upload short: ..... 30
vusers.failed: ..... 0
vusers.session_length:
  min: ..... 250.1
  max: ..... 492.3
  mean: ..... 328.1
  median: ..... 301.9
  p95: ..... 424.2
  p99: ..... 432.7
```

Fig 10. Artillery test ‘upload_shorts.yaml’ for Tukano Kubernetes With Cache

```
http.response_time:
  min: ..... 53
  max: ..... 187
  mean: ..... 97.3
  median: ..... 71.5
  p95: ..... 172.5
  p99: ..... 179.5
http.response_time.2xx:
  min: ..... 53
  max: ..... 179
  mean: ..... 89.5
  median: ..... 71.5
  p95: ..... 153
  p99: ..... 153
```

Fig 11. Artillery test ‘realistic_flow.yaml’ for Tukano Kubernetes With Cache Average Respose Time

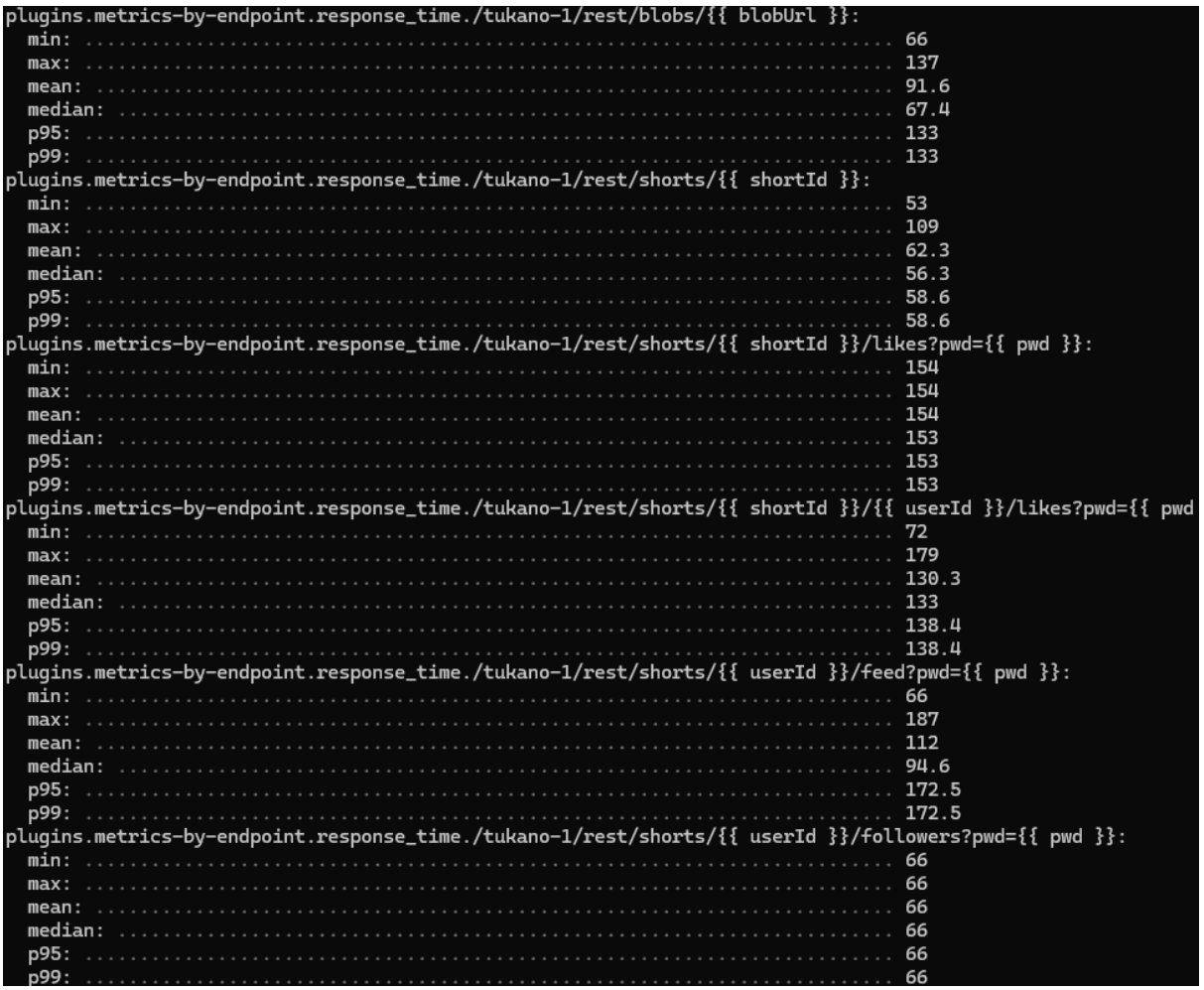

Fig 12. Artillery test ‘realistic_flow.yaml’ for Tukano Kubernetes With Cache Response Time per Operation

Performance Conclusions

By doing these tests, and testing both the Base Tukano App and Tukano Kubernetes with Cache and no Cache, we can conclude that the implementation of Cache is extremely beneficial to a Kubernetes implementation of Tukano and improves response times by a significant margin.