# CMPT 756.203 T4-Wednesday Milestone 2

| | | | |
|---|---|---|---|
| Team | T4-Wednesday | | |
| Section | G100 | | |
| Team Member Name | Kunal Niranjan Desai | Rohan Harode | Juan Ospina |
| Email (SFU) | kdesai@sfu.ca | rharode@sfu.ca | jospina@sfu.ca |
| Email (G-Suite/Gmail) | kdesai1505@gmail.com | harode48@gmail.com | juandavidospina@gmail.com |
| Github ID | kunaldesai97 | rohanharode | juandavidospina |
| Additional Notes | | | |

# Section 1 - Problem Statement

## Problem Domain & Application

**Online Bill Payment System: "My Bill Payments"**

Our client, a recognized Fintech named "ABCFintech," wants to offer its current and potential customers an application to pay their bills online. This application is called "My Bill Payments". Through this application, a customer gets a centralized platform to pay all their bills from different companies like BC Hydro (Electricity), Rogers (cellphone), Telus (Internet) (from now on, named "billers") as well as manage them, thus saving time and avoiding overdue balances and late fees.

For example, a person named John Doe wants to pay three bills at the end of the month: BC Hydro, Cellphone (Rogers), and Internet (Telus). John will be able to log in on our application, and through the "Bill retrieval" functionality, he can look up the details of each bill, using as input the biller name and the bill identifier, e.g., Biller name: "Rogers" and Bill ID: 12345. Assuming that the bill exists, our system will return the bill's relevant information: Biller name, Bill identifier, amount to pay, and due date. Then, John can pay this bill using his credit card. Hence, John will enter its credit card information into the system: credit card number (16 digits), cardholder name, expiration year-month, and the three-digit security code. In case that John has funds available to cover this payment, our system will flag the bill as paid (preventing that it can be paid again by another user) and register the payment information (credit card first four and last four digits, cardholder name and expiration year-month). The user can repeat this process as many times as the number of bills. Also, if John wants to review his paid bills, our system will provide a functionality to review all his past payments (payments history).

# Specification of REST API (Microservices Contract)

Version: v1
Service: Gateway
Visibility: Public
Domain: Ingress-gateway
Serialized Data/Content-Type: json

| API | Description |
|---|---|
| /api/v1/user | Users service URL |
| /api/v1/bill | Bills service URL |
| /api/v1/billers | Billers service URL |

### Service: Users

Version: v1
Visibility: Private
Domain: Users
Serialized Data/Content-Type: json

| API | Description | Request Body/Parameters | Response Body | HTTP Response Code | Error Codes | Request Example | Response Example |
|---|---|---|---|---|---|---|---|

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| PUT - /api/v1/user/login | Login | JSON:{"uname": user_name, “pword”: password} | Hash of user context suitable for passing to other calls | 200 | 500 | PUT https://host:5000/api/v1/user/login | {“UserContext”: “”} |
| PUT - /api/v1/user/logoff | Logoff | JSON: {"jwt": token} | None | 200 | 500 | PUT https://host:5000/api/v1/user/logoff | { Message: ok } |
| PUT - /api/v1/user/ | UPDATE one user | Body: { uname: string, pword: string, Fname: string, Lname: string, Email: string, Secquestion: string, Secanswer:string } | OK response | 200 | 500 | PUT https://host:5000/api/v1/user/ | { Message: ok } |
| POST - /api/v1/user/ | CREATE one user | Body: { uname: string, Fname: string, Lname: string, Email: string, Secquestion:st | OK response | 200 | 500 | POST https://host:5000/api/v1/user | { ResponseMetadata: { … etc. } } |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | ring, Secanswer:string }<br><br>Params: None | | | | | |
| DELETE - /api/v1/user/ | DELETE one user by username | | JSON of response from aws | 200 | 500 | DELETE https://host:5000/api/v1/user/ | { ResponseMetadata: { … etc. } } |
| GET - /api/v1/user/<uname> | GET one user by username | | JSON of User entity | 200 | 500 | GET https://host:5000/api/v1/user?user=johndoe | {User Object} |
| GET - /api/v1/user?userkey | GET user by key | Query Params:<br><br>userkey:string | JSON of User entity | 200 | 500 | GET https://host:5000/api/v1/user?userkey=email:johndoe@gmail.com | {User Object} |

**Service: Bill**

Version: v1
Visibility: Private
Domain: Bills
Serialized Data/Content-Type: json

| API | Description | Request Body/Parameters | Response Body | HTTP Response Code | Error Codes | Request Example | Response Example |
|---|---|---|---|---|---|---|---|
| GET - /api/v1/bill/ | Retrieve one bill | Param: bill_id | None | 200 | 500 | GET https://host:5001/api/v1/bill/ | { bill_id: string, biller_id:string, amount:float, due_date: date, bill_paid:boolean} |
| POST - /api/v1/bill/ | Insert one bill | Body: { user_id: string, biller_id:string, bill_amount:float, due_date:date, bill_paid: boolean (default: False)} User-context token | OK response | 200 | 500 | POST https://host:5001/api/v1/bill/ | { Message: ok } |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| DELETE - /api/v1/bill/ | DELETE one bill | Param: bill_id | JSON of response from aws | 200 | 500 | DELETE https://host:5001/api/v1/bill/ | { ResponseMetadata: { … etc. } } |
| PUT - /api/v1/bill/pay/<bill_id> | Pay bill | Body: {“cc_number”:, cc_exp_dat: String, payment_date: String}<br><br>Header: User-context token | OK response | 200 | 500 | PUT https://host:5000/api/v1/bill/pay | { Message: ok } |
| GET - /api/v1/bill/<uname> | GET bills by username | Query Params:<br><br>username:string | JSON of Bill entity | 200 | 500 | GET https://host:5000/api/v1/bill/johndoe | {Bill Object} |

**Service: Biller**

Version: v1
Visibility: Private
Domain: Billers
Serialized Data/Content-Type: json

| API | Description | Request Body/Parameters | Response Body | HTTP Response Code | Error Codes | Request Example | Response Example |
|---|---|---|---|---|---|---|---|
| GET - /api/v1/biller/ | Retrieve one biller | Param: biller_id | None | 200 | 500 | GET https://host:5001/api/v1/biller/ | { biller_id: string, biller_name:string, biller_description:string} |
| POST - /api/v1/biller | Insert one biller | Body: { biller_id: string, biller_name: string, biller_description:string } | OK response | 200 | 500 | POST https://host:5001/api/v1/biller/ | { Message: ok } |
| DELETE - /api/v1/biller/ | DELETE one biller | Param: biller_id | JSON of response from aws | 200 | 500 | DELETE https://host:5001/api/v1/biller/ | { ResponseMetadata: { … etc. } } |

**Service: db**

Version: v1
Visibility: Private
Domain: Datastore
Serialized Data/Content-Type: json

| API | Description | Request Body/Parameters | Response Body | HTTP Response Code | Error Codes | Request Example | Response Example |
|---|---|---|---|---|---|---|---|
| GET - /api/v1/datastore/read | Read an object | Param: object-type, object-key | JSON of aws response | 200 | 500 | GET https://host:5002/api/v1/datastore/read?objtype=user&objkey=johndoe | { "Count": 1, "Items": [ { "uname": "johndoe", "fname": "John", "lname": "Doe", "email": "johndoe@gmail.com", “secquestion”:”Name of your first pet?”, “secanswer”:”Lucky” } ], "ResponseMetadata": { "HTTPHeaders": { "connection": "keep-alive", "content-length": "165", "content-type": "application/x-amz-json-1.0", "date": "Sat, 12 Sep 2020 18:16:15 GMT", "server": "Server", "x-amz-crc32": "196980578", "x-amzn-requestid": "AOGKN903DF66VLU3GEBEEO8DK3VV4KQNSO5AEMVJF66Q9ASUAAJG" }, "HTTPStatusCode": 200, "RequestId": "AOGKN903DF66VLU3GEBEEO8DK3VV4K |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | QNSO5AEMVJF66Q9ASUAAJG", "RetryAttempts": 0 }, "ScannedCount": 1 } |
| POST - /api/v1/datastore/write | Write an object | Body: objtype, object-key(s) | ID of new entity | 200 | 500 | POST https://host:5002/api/v1/datastore/write | { "uname": "johndoe" } |
| DELETE = /api/v1/datastore/delete | Delete an object | Param: objtype, object-key | JSON of aws response | 200 | 500 | DELETE https://host:5000/api/v1/datastore/delete?objtype=user&objkey=johndoe | { "ResponseMetadata": { "HTTPHeaders": { "connection": "keep-alive", "content-length": "2", "content-type": "application/x-amz-json-1.0", "date": "Sat, 12 Sep 2020 18:13:04 GMT", "server": "Server", "x-amz-crc32": "2745614147", "x-amzn-requestid": "N7R6LO93FFBDH1A5GRRL55LBS7VV4KQNSO5AEMVJF66Q9ASUAAJG" }, "HTTPStatusCode": 200, "RequestId": "N7R6LO93FFBDH1A5GRRL55LBS7VV4KQNSO5AEMVJF66Q9ASUAAJG", "RetryAttempts": 0 } } |
| PUT - /api/v1/datastore/update | Update an object | Params: objtype, objkey | JSON of aws response | 200 | 500 | PUT https://host:5002/api/v1/datastore/update?objtype=user&objkey=email:johndoenew@gmail.com | { "ResponseMetadata": { "HTTPHeaders": { "connection": "keep-alive", "content-length": "2", "content-type": "application/x-amz-json-1.0", "date": "Sat, 12 Sep 2020 18:13:04 GMT", "server": "Server", "x-amz-crc32": "2745614147", "x-amzn-requestid": "N7R6LO93FFBDH1A5GRRL55LBS7VV4K |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | Body: object-key(s)<br><br>Example: email | | | | | QNSO5AEMVJF66Q9ASUAAJG" }, "HTTPStatusCode": 200, "RequestId": "N7R6LO93FFBDH1A5GRRL55LBS7VV4KQNSO5AEMVJF66Q9ASUAAJG", "RetryAttempts": 0 } } |

## Database Schema (DynamoDB)

Table: Users

| **Tag** | **Value** | **Comment** |
|---|---|---|
| lname | string | Last name of user |
| fname | string | First name of user |
| uname | string | Email ID of user |
| password | string | User’s account password |
| security_question | string | Security question in case user forgets password |

| | | |
|---|---|---|
| security_answer | string | Answer to the security question |
| user_id | string | Unique id generated by DynamoDB |

Table: Bills

| **Tag** | **Value** | **Comment** |
|---|---|---|
| bill_amount | decimal | Amount billed |
| bill_paid | boolean | true/false |
| due_date | string | Due date for the bill |
| payment_date | string | Date of the bill payment |
| cc_first_four_digits | integer | Identification of credit card franchise: Visa, MasterCard etc |
| cc_last_four_digits | integer | Credit card information for troubleshooting |
| cc_exp_date | string | Expiry date of credit card |
| user_id | string | User id |
| biller_id | string | Unique id generated by DynamoDB |
| bill_id | string | Unique id generated by DynamoDB |

Table: Billers

| **Tag** | **Value** | **Comment** |
|---|---|---|
| biller_name | string | Name of the billing company |
| active | boolean | true/false |
| biller_description | string | Description of the billing company |
| biller_id | string | Unique id not interpreted by DynamoDB |

# Section 2 - Github Repo Guide

| **Path** | **Note** |
|---|---|
| /docs | Information of various services |
| /docs/user_service | Details of users service and functionalities |
| /docs/bill_service | Details of bills service and functionalities |
| /docs/biller_service | Details of billers service and functionalities |
| /docs/minikube | Minikube steps, commands |
| /docs/docker | Docker steps, commands |
| /docs/AKS | Azure Kubernetes commands |
| /code | Code for API calls of different services |
| /code/users | Code for API calls of users services |

| | |
|---|---|
| /code/bills | Code for API calls of bills services |
| /code/billers | Code for API calls of billers services |
| /IaC | Infrastructure as a Code directory |
| /IaC/cloudformation | Cloudformation stacks |
| /IaC/cluster | Cluster creation |
| /IaC/k8s | Service gateway |

# Section 3 - Reflection on Development

We adopted the idea of creating a product backlog to have a clear understanding of all the services we plan to implement for the project. We also had weekly meetings for 30 minutes each to catch up on the progress that will help us stay on track with our project plans, collaborate with others and brainstorm regarding the application and REST APIs. Since we are a team of 3 (with no scrum master or product owner), each of us took responsibility to make sure we were focusing on the bigger picture of the project while undertaking tasks. We have already started the Scrum process by using a Kanban dashboard on Github where we are creating stories and tasks. We are also maintaining a shared Google Doc to suggest and incorporate changes to the proposed application, architecture, REST API's and resolving them to promote a continuous cycle of improvement.

The readings which were most helpful to us were R2 (Scrum guide), R3 (Epics, stories, tasks, subtasks), and R6 (Revision control systems). The scrum guide gave us a good understanding of how scrum works and methodologies. It motivated us to try it for the term project starting with the product backlog and weekly meetings proving beneficial to keep us on track and discuss the project requirements. The R3 reading introduced us to organizing tasks within the team and implementing the ideas on Github's dashboards for tracking progress. R6 reading helped us use Git for our tasks with individual branches and then review each of our code to finally piece together each of our contributions to the master for submission.

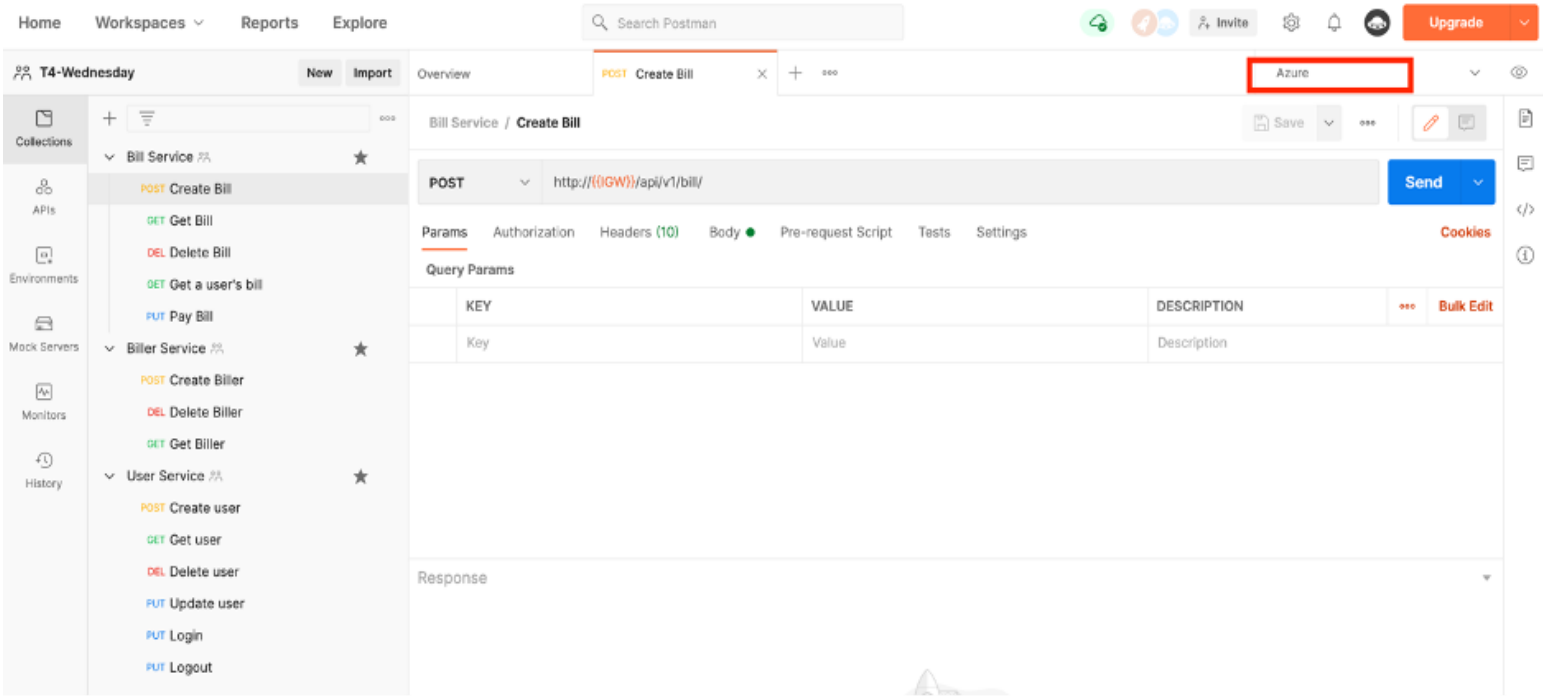
Our team performs well, adheres to the scrum principles, makes good progress, and handles tasks regularly. In our professional experience, we belonged to larger teams with a scrum master to supervise the progress. For this term project, the unavailability of

the product owner or scrum master added a bit more load on each of us, but we managed to take responsibility for our tasks and bring it together for the project. We believe that scrum success depends a lot on the team members if they actively participate in the meetings, collaborate well, etc. Consequently, we will ensure that each of us is proactive in his work and makes valuable contributions at every stage of the development lifecycle.

# Section 4 - Analysis

## Section 4.1 - Approach

1. We created and set up our team Postman workspace (T4-Wednesday) with all microservices (collections) and Request API's that are required for our application. We also created 2 different environments (Minikube, Azure) for testing and recording our API's through Gatling.




2. We set Postman proxy (localhost:8000) for sending requests to our API's. We recorded our basic application flow (Coverage) using Gatling recorder (HTTP Proxy mode) to create a scala file.

3. We took the generated script and ran the Coverage simulation. We analyzed the Coverage simulation results (report) for possible inferences and network analysis (refer to Section 4.2).
4. For Load Simulation, we edited the generated file with Azure cluster details and ramped up the users (50), requests/s and ran it for 30 minutes to see its behaviour. (refer to Section 4.2)

## Section 4.2 - Testing Analysis

**Planning:**

- API methods in scope for testing:
  - Create Bill
  - Get Bill
  - Delete Bill
  - Get User Bill
  - Pay Bill
  - Create Biller
  - Delete Biller
  - Get Biller
  - Create User
  - Get User
  - Delete User
  - Update User
  - User Login
  - User Logout

- Testing scenarios:

  - Scenario 1 - Coverage simulation: **One active user** tests all APIs.

  - Scenario 2 - Load simulation: **50 active users, 20 API calls/sec, 30 minutes duration**

  - Scenario 3 - Load simulation with failure analysis:
    - Possible failover cases:
      - Istio Service Mesh
        - Circuit breakers
        - Retries
        - Fault injections
      - Cluster nodes failover

## Testing results:

Scenario 1: **Coverage simulation**:

- **Findings:**

  - It was possible to execute all the API operations in scope without errors. No HTTP failed requests occurred.

  - "Create" operations": *Create Biller,* and *Create User* had a similar response time: 111ms and 106ms, respectively. However, in the case of "*Create Bill*", the response time was 3064ms. We'll wait for load simulation results to determine if this issue was a one-time situation or if it is necessary to review the implementation of the "*Create Bill*" method to identify potential optimizations.

  - The remaining operations had a similar response time of 100ms on average, which is the expected response time defined by us for API responses (max 300ms):
    - "Retrieve" operations (*Get Bill*, *Get User Bill*, *Get Biller*, and *Get User*) response time: 100ms on average.

    - "Delete" operations (*Delete Bill*, *Delete Biller,* and *Delete User*) response time: 100ms on average.

    - "*Pay Bill*" operation: This PUT method had a response time of 104ms.

    - "*Update User*" operation: This PUT method had a response time of 143ms.

    - "Login/Logout" operations (*User Login* and *User Logout*): These PUT methods had a response time of 100ms on average.

Scenario 2: **Load simulation**:

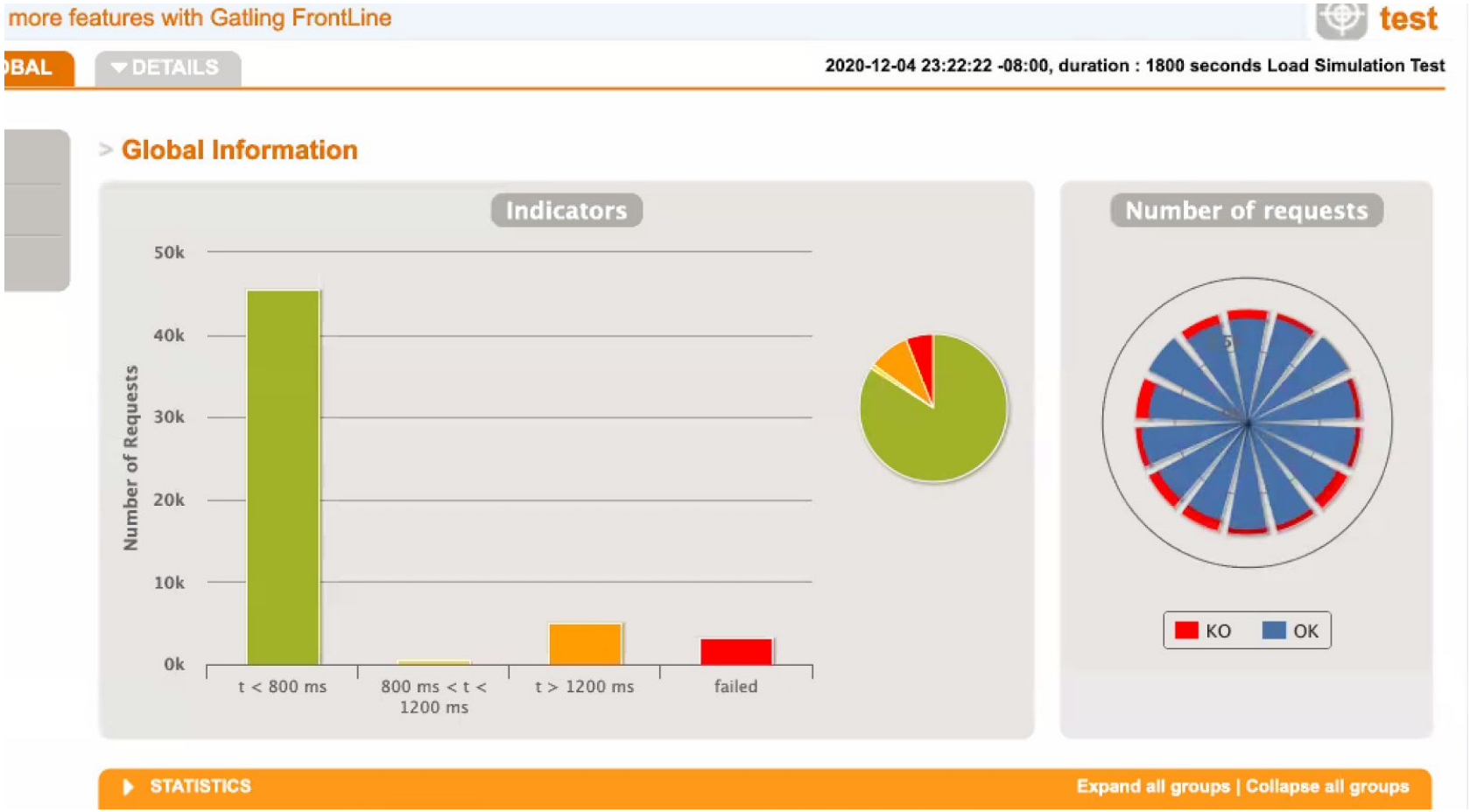
- **Findings:**

  - "Create" operations (*Create New User*, *Create New Bill* and *Create New Biller*) had response times on average of about 2195 ms.

  - "Retrieve" operations: The *Get* Requests had the best average response times of about 350 ms.
  -

- “Pay Bill” operation: This PUT method had an average response time of 3497 ms, with zero failed requests.

- “Delete” operations (*Delete User*, *Delete Bill*, and *Delete Biller*): The three delete operations except the *Delete Biller* (212 ms) had high response times of about 2500 ms.
- As we can infer from the below graph that many KO (failed) requests were observed when we performed Load testing as compared to Coverage simulation. But, overall the OK requests are many and we can say that our cluster infrastructure (2 worker nodes) was not good enough to sustain that load. Given more nodes and capacity we could have minimized the failed requests.



STATISTICS

Expand all groups | Collapse all groups

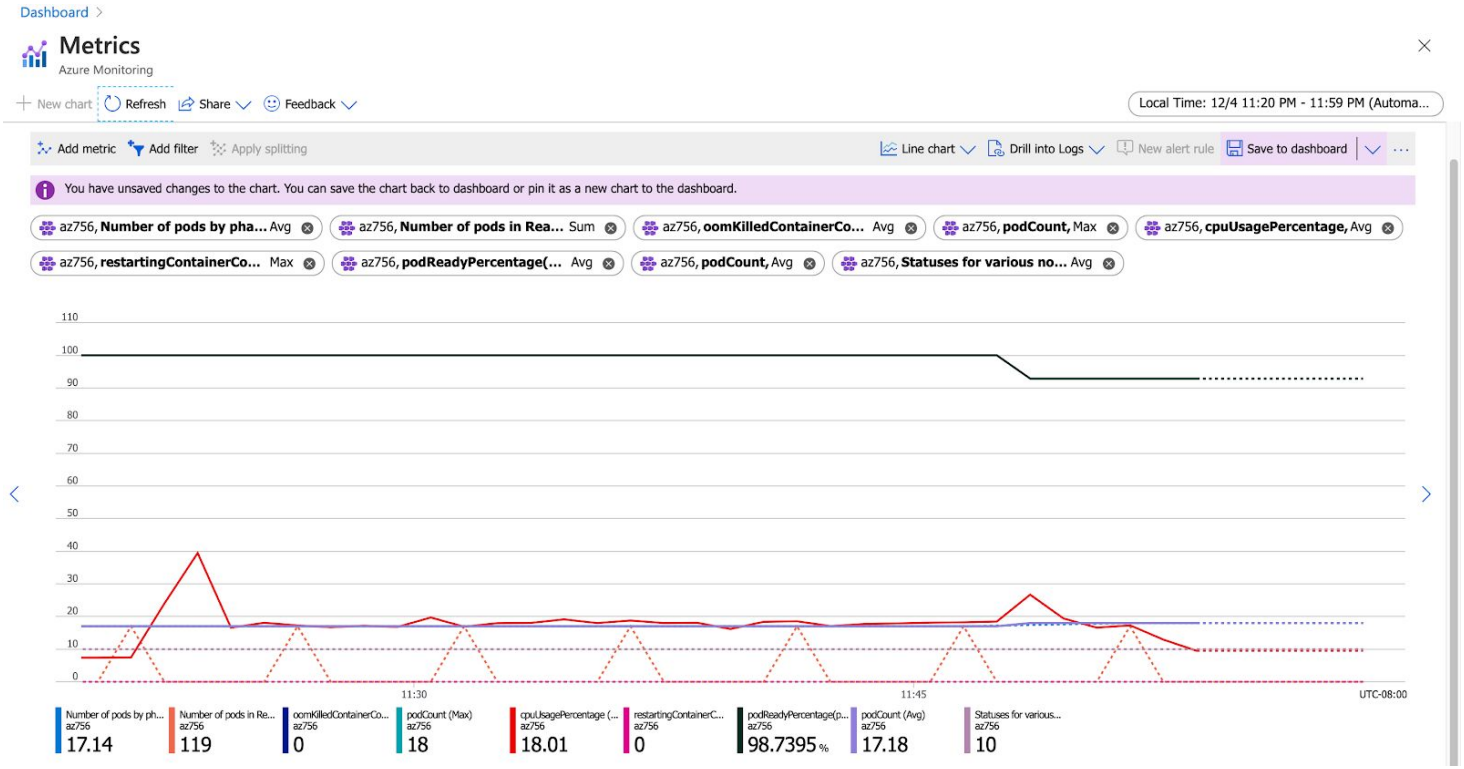
| Requests | Executions | | | | | Response Time (ms) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Total | OK | KO | % KO | Cnt/s | Min | 50th pct | 75th pct | 95th pct | 99th pct | Max | Mean | Std Dev |
| Global Information | 54385 | 51064 | 3321 | 6% | 30.214 | 66 | 172 | 377 | 13195 | 26145 | 27487 | 1648 | 4984 |
| Create User | 3911 | 3600 | 311 | 8% | 2.173 | 106 | 253 | 438 | 13452 | 26268 | 27338 | 2195 | 5927 |
| Get User | 3911 | 3735 | 176 | 5% | 2.173 | 105 | 156 | 345 | 642 | 1191 | 3199 | 254 | 232 |
| Log in User | 3911 | 3911 | 0 | 0% | 2.173 | 67 | 75 | 80 | 172 | 599 | 2513 | 98 | 124 |
| Create Biller | 3911 | 3737 | 174 | 4% | 2.173 | 74 | 150 | 328 | 568 | 910 | 1891 | 235 | 180 |
| Get Biller | 3911 | 3738 | 173 | 4% | 2.173 | 73 | 148 | 331 | 573 | 852 | 1808 | 230 | 172 |
| Create Bill | 3911 | 3493 | 418 | 11% | 2.173 | 74 | 555 | 6712 | 26009 | 26385 | 27270 | 4873 | 7710 |
| Get Bill | 3879 | 3681 | 198 | 5% | 2.155 | 74 | 156 | 333 | 527 | 1162 | 27293 | 348 | 1698 |
| Get Bills for User | 3863 | 3678 | 185 | 5% | 2.146 | 74 | 154 | 330 | 410 | 856 | 26467 | 243 | 749 |
| Pay Bill | 3863 | 3502 | 361 | 9% | 2.146 | 74 | 335 | 847 | 25997 | 26348 | 27262 | 3497 | 7125 |
| Delete Bill | 3863 | 3505 | 358 | 9% | 2.146 | 74 | 237 | 439 | 25990 | 26378 | 27325 | 2529 | 6488 |
| Delete Biller | 3863 | 3666 | 197 | 5% | 2.146 | 74 | 139 | 322 | 585 | 748 | 1351 | 221 | 157 |
| Update User | 3863 | 3414 | 449 | 12% | 2.146 | 105 | 659 | 6733 | 26104 | 26433 | 27467 | 5501 | 8194 |
| Logoff User | 3863 | 3863 | 0 | 0% | 2.146 | 66 | 75 | 81 | 203 | 530 | 1957 | 97 | 102 |
| Delete User | 3862 | 3541 | 321 | 8% | 2.146 | 106 | 311 | 595 | 25988 | 26293 | 27487 | 2788 | 6526 |

- During the load simulation we found that our db-container was in an **Evicted state** during the run. As we can see that initially we had 17 pods running in total (all types) but after db-container pod was evicted, a new pod was initialized and now there are 18 total pods (See stats below line graph).

```
kubectl get gw,deployments,pods
NAME                                          AGE
gateway.networking.istio.io/my-gateway   4d14h

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/billcontainer         1/1     1            1           4d14h
deployment.apps/billercontainer       1/1     1            1           4d14h
deployment.apps/dbcontainer           1/1     1            1           4d14h
deployment.apps/usercontainer         1/1     1            1           4d14h

NAME                                      READY   STATUS    RESTARTS   AGE
pod/billcontainer-5c869cf66f-d9jbx        2/2     Running   0          26m
pod/billercontainer-99c4447c8-2x4nf       2/2     Running   0          26m
pod/dbcontainer-6bf7b5ff77-4f7x9          2/2     Running   0          58s
pod/dbcontainer-6bf7b5ff77-swjn2          0/2     Evicted   0          26m
pod/usercontainer-5d44df8dcf-pkqbl        2/2     Running   0          26m
kubectl -n termproject get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
billcontainer     ClusterIP   10.0.27.12      <none>        5002/TCP   4d14h
billercontainer   ClusterIP   10.0.156.143    <none>        5001/TCP   4d14h
dbcontainer       ClusterIP   10.0.149.147    <none>        5000/TCP   4d14h
usercontainer     ClusterIP   10.0.79.13      <none>        5003/TCP   4d14h
```




*Azure metrics*

Scenario 3: **Load Simulation with Failure analysis**:

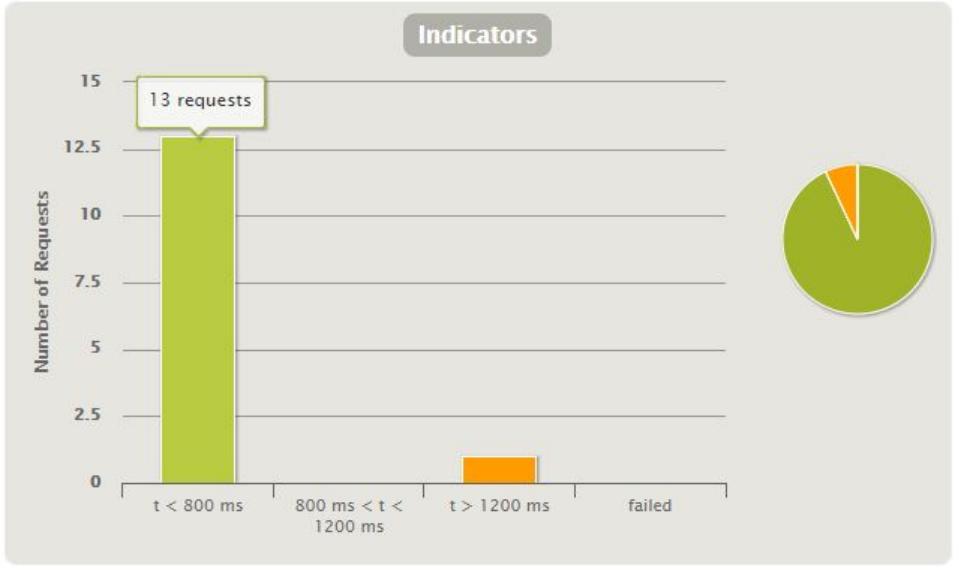
- **Deleting a pod (Azure Metrics):** We observed that after deleting a pod, k8s create a new set of pod to compensate for the down node and try to pick up the load again. This essentially works in a similar way how HDFS and other distributed systems work which separates the services and the underlying system.

- **Increased user load (150):** When we increased the user load for our application, we found out that few containers were in evicted state and k8s initialized new pods.

- **Re-route/ Shutdown isio service mesh** - Pending.

Appendix A: Gatling statistics for the **coverage simulation**:

- Global information:
  - Indicators:




  - Statistics:

STATISTICS

Expand all groups | Collapse all groups

| Requests | Executions | | | | | Response Time (ms) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Total | OK | KO | % KO | Cnt/s | Min | 50th pct | 75th pct | 95th pct | 99th pct | Max | Mean | Std Dev |
| Global Information | 14 | 14 | 0 | 0% | 0.318 | 4 | 106 | 108 | 1165 | 2684 | 3064 | 312 | 764 |
| Create Bill | 1 | 1 | 0 | 0% | 0.023 | 3064 | 3064 | 3064 | 3064 | 3064 | 3064 | 3064 | 0 |
| Get Bill | 1 | 1 | 0 | 0% | 0.023 | 106 | 106 | 106 | 106 | 106 | 106 | 106 | 0 |
| Delete Bill | 1 | 1 | 0 | 0% | 0.023 | 106 | 106 | 106 | 106 | 106 | 106 | 106 | 0 |
| Get User Bill | 1 | 1 | 0 | 0% | 0.023 | 107 | 107 | 107 | 107 | 107 | 107 | 107 | 0 |
| Pay Bill | 1 | 1 | 0 | 0% | 0.023 | 104 | 104 | 104 | 104 | 104 | 104 | 104 | 0 |
| Create Biller | 1 | 1 | 0 | 0% | 0.023 | 111 | 111 | 111 | 111 | 111 | 111 | 111 | 0 |
| Delete Biller | 1 | 1 | 0 | 0% | 0.023 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 0 |
| Get Biller | 1 | 1 | 0 | 0% | 0.023 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 0 |
| Create User | 1 | 1 | 0 | 0% | 0.023 | 106 | 106 | 106 | 106 | 106 | 106 | 106 | 0 |
| Get User | 1 | 1 | 0 | 0% | 0.023 | 103 | 103 | 103 | 103 | 103 | 103 | 103 | 0 |
| Delete User | 1 | 1 | 0 | 0% | 0.023 | 104 | 104 | 104 | 104 | 104 | 104 | 104 | 0 |
| Update User | 1 | 1 | 0 | 0% | 0.023 | 143 | 143 | 143 | 143 | 143 | 143 | 143 | 0 |
| User Login | 1 | 1 | 0 | 0% | 0.023 | 108 | 108 | 108 | 108 | 108 | 108 | 108 | 0 |
| User Logout | 1 | 1 | 0 | 0% | 0.023 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 0 |