

# Documentation

---

## Instituto Tecnológico de Costa Rica

## IC4302 - Bases de Datos II

### Documentación Proyecto Opcional

**Profesor:** Nereo Campos Araya

#### Estudiantes:

- Fiorella Zelaya Coto - 2021453615
  - Isaac Araya Solano - 2018151703
  - Melany Salas Fernández - 2021121147
  - Moisés Solano Espinoza - 2021144322
  - Pablo Arias Navarro - 2021024635
- 

## Guía de instalación y uso de la tarea

---

1- Descomprimir el archivo .zip y abrir la línea de comandos wsl en la ubicación de la carpeta **tareaCorta1**

```
/Bases-De-Datos-11 Grupo$ cd tareaCorta1  
/Bases-De-Datos-11 Grupo/tareaCorta1$ []
```

Posteriormente, puede ir a las carpetas **charts\databases** donde encontrará el archivo **values.yaml**, aquí podrá escoger la base de datos a monitorear cambiando el valor de **enabled** a true

```
1  mongodb:  
2      enabled: false  
3      architecture: replicaset  
4      replicaCount: 3  
5      metrics:  
6          enabled: true  
7          serviceMonitor:  
8              enabled: true  
9              namespace: "monitoring"  
10         auth:  
11             rootPassword: "1234"  
12     postgresql:  
13         enabled: false  
14         metrics:  
15             enabled: true  
16             serviceMonitor:  
17                 enabled: true  
18                 namespace: "monitoring"
```

2- Ir a la carpeta **docker** con el comando **cd docker**

```
/Bases-De-Datos-11 Grupo/tareaCorta1$ cd docker
/Bases-De-Datos-11 Grupo/tareaCorta1/docker$
```

3- Ejecutar el comando **bash build.sh**, si da error al ejecutar, intente con el comando **dos2unix build.sh** y, posteriormente, ejecute de nuevo **bash build.sh**

```
melanysf@LAPTOP-28PDHQBR:/mnt/c/Users/melan/OneDrive - Estudiantes ITCR/Dокументos/TEC/V Semestre/Bases de Datos 2/Bases-De-Datos-11 Grupo/tareaCorta1/docker$ bash build.sh
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/
[+] Building 42.1s (3/4)
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 124B
=> [internal] load .dockignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/ubuntu:latest
=> [auth] library/ubuntu:pull token for registry-1.docker.io
 8.3s
 0.4s
 8.4s
 0.0s
 3.5s
 0.0s
```

4- Ir a la carpeta **charts** con el comando **cd ..../charts**

```
/Bases-De-Datos-11 Grupo/tareaCorta1/docker$ cd ../../charts
/Bases-De-Datos-11 Grupo/tareaCorta1/charts$
```

5- Ejecutar el comando **bash install.sh**, si da error al ejecutar, intente con el comando **dos2unix install.sh** y, posteriormente, ejecute de nuevo **bash install.sh**, ya con esto, debería ver los distintos pods en lens

```
melanysf@LAPTOP-28PDHQBR:/mnt/c/Users/melan/OneDrive - Estudiantes ITCR/Dокументos/TEC/V Semestre/Bases de Datos 2/Bases-De-Datos-11 Grupo/tareaCorta1/charts$ bash install.sh
Warning: Immediate deletion does not wait for confirmation that the running resource has been terminated. The resource may continue to run on the cluster indefinitely.
No resources found
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/melanysf/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/melanysf/.kube/config
Release "bootstrap" has been upgraded. Happy Helming!
NAME: bootstrap
LAST DEPLOYED: Fri Mar 24 10:35:05 2023
NAMESPACE: default
STATUS: deployed
REVISION: 2
TEST SUITE: None
sleep: invalid time interval '5\'
Try 'sleep --help' for more information.
WARNING: Kubernetes configuration file is group-readable. This is insecure. Location: /home/melanysf/.kube/config
WARNING: Kubernetes configuration file is world-readable. This is insecure. Location: /home/melanysf/.kube/config
```

6- Una vez instalado y cuando los pods están arriba, abrir la carpeta **Gatling** e ir a **gatling-charts-highcharts-bundle-3.9.2/bin** y ejecutar **galing.bat**

V Semestre > Bases de Datos 2 > Bases-De-Datos-II Grupo > TareaCorta1 > Gatling > gatling-charts-highcharts-bundle-3.9.2 > bin					
Nombre	Estado	Fecha de modificación	Tipo	Tamaño	
gatling	✓	23/3/2023 18:20	Archivo por lotes ...	3 KB	1
gatling	✓	23/3/2023 18:20	Archivo de origen ...	2 KB	2
recorder	✓	23/3/2023 18:20	Archivo por lotes ...	3 KB	
recorder	✓	23/3/2023 18:20	Archivo de origen ...	2 KB	

7- Esperar que abra la consola.

8- Una vez abierta, seleccionar la opción **[1]** presionando las teclas 1, seguido de enter.

```
GATLING_HOME is set to "C:\Users\melan\OneDrive - Estudiantes ITCR\Documentos\TEC\V Semestre\Bases de Datos 2\Bases-De-Datos-ll Grupo\TareaCortal\Gatling\gatling-charts-highcharts-bundle-3.9.2"
JAVA = java
GATLING_HOME is set to C:\Users\melan\OneDrive - Estudiantes ITCR\Documentos\TEC\V Semestre\Bases de Datos 2\Bases-De-Datos-ll Grupo\TareaCortal\Gatling\gatling-charts-highcharts-bundle-3.9.2
Do you want to run the simulation locally, on Gatling Enterprise, or just package it?
Type the number corresponding to your choice and press enter
[0] <Quit>
[1] Run the Simulation locally
[2] Package and upload the Simulation to Gatling Enterprise Cloud, and run it there
[3] Package the Simulation for Gatling Enterprise
[4] Show help and exit
|
```

9- Aparecerá lo siguiente, debe presionar la tecla enter nuevamente para que la prueba de carga comience.

```
GATLING_HOME is set to "C:\Users\melan\OneDrive - Estudiantes ITCR\Documentos\TEC\V Semestre\Bases de Datos 2\Bases-De-Datos-ll Grupo\TareaCortal\Gatling\gatling-charts-highcharts-bundle-3.9.2"
JAVA = java
GATLING_HOME is set to C:\Users\melan\OneDrive - Estudiantes ITCR\Documentos\TEC\V Semestre\Bases de Datos 2\Bases-De-Datos-ll Grupo\TareaCortal\Gatling\gatling-charts-highcharts-bundle-3.9.2
Do you want to run the simulation locally, on Gatling Enterprise, or just package it?
Type the number corresponding to your choice and press enter
[0] <Quit>
[1] Run the Simulation locally
[2] Package and upload the Simulation to Gatling Enterprise Cloud, and run it there
[3] Package the Simulation for Gatling Enterprise
[4] Show help and exit
1
22:46:56.981 [WARN ] i.g.c.ZincCompiler$ - --target is deprecated: Use --release instead to compile against the correct platform API.
22:47:03.210 [WARN ] i.g.c.ZincCompiler$ - one warning found
testDataset is the only simulation, executing it.
Select run description (optional)
```

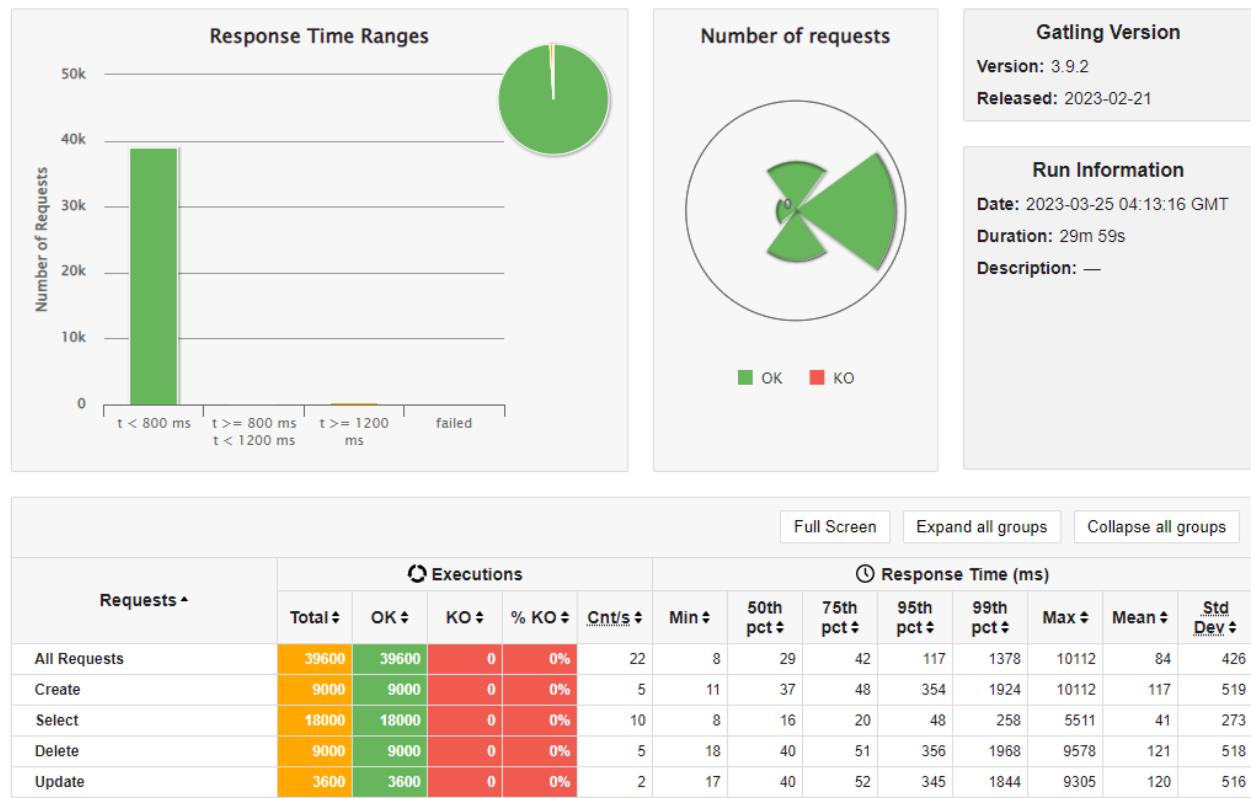
10- Puede entrar a la dirección que sale en consola para verificar la cantidad de usuarios que lograron conectarse con éxito.

```
Simulation testDataset completed in 1800 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

=====
---- Global Information -----
> request count                                39600 (OK=39600 KO=0    )
> min response time                            8 (OK=8      KO=-    )
> max response time                            10112 (OK=10112 KO=-    )
> mean response time                           84 (OK=84     KO=-    )
> std deviation                               426 (OK=426    KO=-    )
> response time 50th percentile                29 (OK=29     KO=-    )
> response time 75th percentile                42 (OK=42      KO=-    )
> response time 95th percentile                117 (OK=117    KO=-    )
> response time 99th percentile                1378 (OK=1378   KO=-    )
> mean requests/sec                          22 (OK=22     KO=-    )
---- Response Time Distribution -----
> t < 800 ms                                    38955 ( 98%)
> 800 ms <= t < 1200 ms                      196 (  0%)
> t >= 1200 ms                                 449 (  1%)
> failed                                       0 (  0%)
=====

Reports generated in 0s.
Please open the following file: file:///C:/Users/melan/OneDrive%20-%20Estudiantes%20ITCR/Documentos/TEC/V%20Semestre/Bases%20de%20Datos%20/Bases-De-Datos-ll%20Grupo/TareaCortal/Gatling/gatling-charts-highcharts-bundle-3.9.2/results/testdataset-20230325041314762/index.html
Presione una tecla para continuar . . .
```

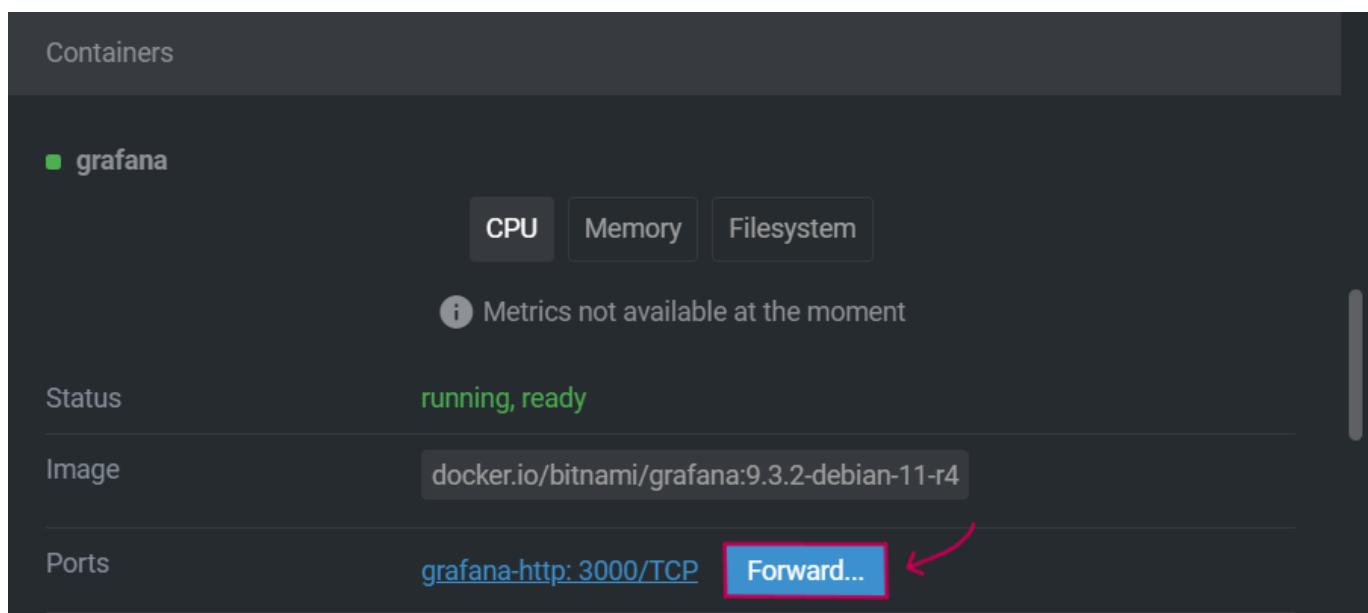
Si ingresa, puede que salga algo similar a lo siguiente:



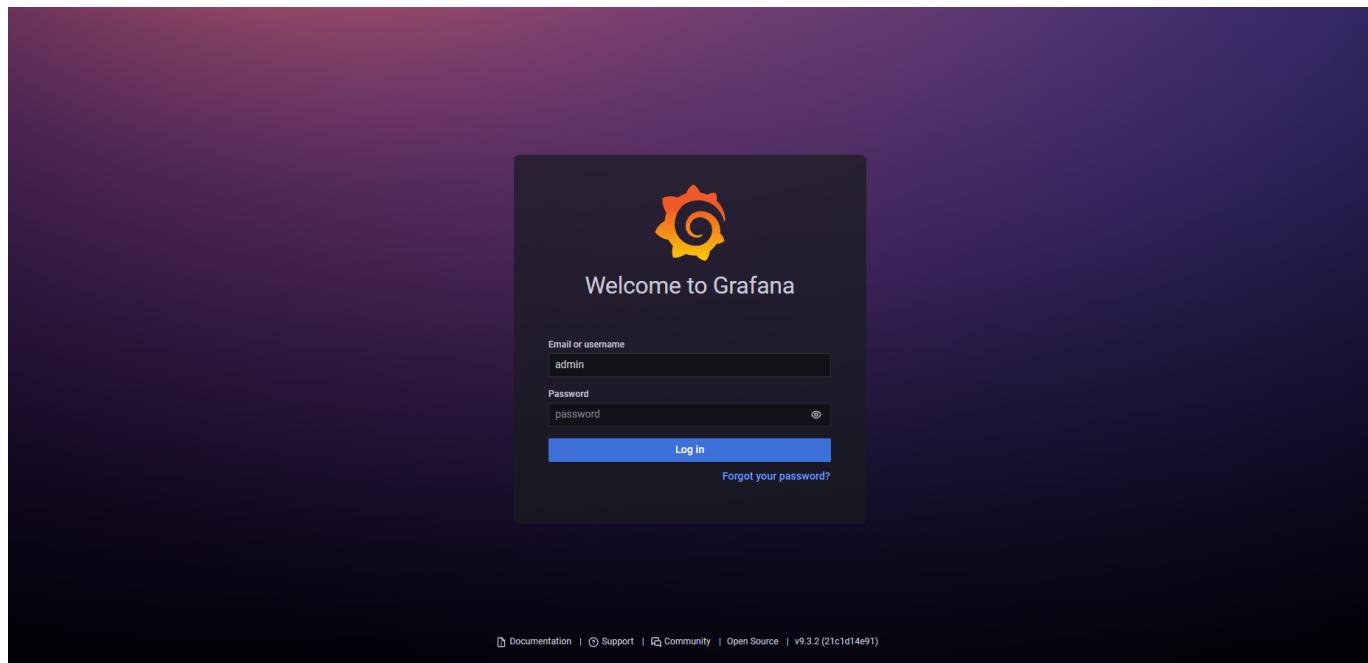
11- Posteriormente, puede observar las métricas en **Grafana**, seleccionan al pod **grafana-deployment** desde lens



12- Ir a la sección de containers-ports y seleccionar **forward** para ingresar al Grafana.



13- Va a abrir la siguiente ventana:



14- Para encontrar el password de grafana, dirigirse a secrets en Lens y seleccionar **grafana-admin-credentials** y ver el password, copiarlo y pegarlo en la ventana que se abrió anteriormente.

Name	Namespace	Labels	Data
grafana-admin-credentials	monitoring		GF_SECURITY_ADMIN_PASSWORD: ZqZF9-5JN-GuQQ== GF_SECURITY_ADMIN_USER: admin

15- Ingresar y ver las métricas.

16- Para desinstalar, ejecute el comando **bash uninstall.sh**

## Configuración de las herramientas

### MariaDB

Para instalar el cliente de MariaDB se creó la imagen de MariaDB que incluye al cliente utilizando un Dockerfile. En nombre de esta imagen se guarda en el archivo ubicado en stateless/values.yaml que contiene otros valores de configuración como el mapName, name y volumeName.

```
mariadbDb:
  mapName: script-db
  name: mariadb-init
  volumeName: scripts
  image: moisose/mariadb-client
```

En charts/databases/values.yaml se tienen algunas configuraciones para la instancia de MariaDB, como datos necesarios para la autenticación, la cantidad de replicas que se necesitan, los nombres del nodo primario y nodos secundarios, el namespace y también podemos activar o deactivate la ejecución del pod de MariaDB.

```
mariadb:
  enabled: false
  architecture: replication
  primary:
    name: "master-node"
  secondary:
    name: "slave"
    replicaCount: 2
  metrics:
    enabled: true
    serviceMonitor:
      enabled: true
      namespace: "monitoring"
  auth:
    rootPassword: "1234"
```

Para iniciar el cliente de MariaDB que permite acceder a la base de datos se utilizó un *Job*. En el archivo *mariadbDB.yaml* se define un ConfigMap y el Job.

Dentro del Job se declaran las variables de entorno y se asigna el valor de cada una. Se declaran las variables de entorno para el *host*, *password* y *database*. También se utilizan los valores de las configuraciones que guardamos en stateless/values.yaml para asignar la imagen que le corresponde al Pod y asignar nombres a volúmenes u otros.

Asimismo, en este Job se indica el comando que se debe ejecutar en *args*, el cual se encarga de cargar el archivo sql (*babynames.sql*) que contiene las tablas y los procedimientos almacenados de la base de datos.

## MariaDB Galera

Para instalar el cliente de MariaDB Galera se creó la imagen de MariaDB Galera que incluye al cliente utilizando un Dockerfile. En nombre de esta imagen se guarda en el archivo ubicado en stateless/values.yaml que contiene otros valores de configuración como el *mapName*, *name* y *volumeName*.

```
mariadbGalera:
  mapName: script-db
  name: mariadb-galera-init
  volumeName: scripts
  image: moisose/mariadb-galera-client
```

En charts/databases/values.yaml se tienen algunas configuraciones para la instancia de MariaDB Galera, como la cantidad de replicas que se requieren y los datos de autenticación (contraseña) para el usuario y la base de

datos. También podemos activar o deactivate la ejecución del pod de MariaDB.

```
mariadbgalera:  
  enabled: true  
  replicaCount: 2  
  metrics:  
    enabled: true  
    serviceMonitor:  
      enabled: true  
      namespace: "monitoring"  
  rootUser:  
    password: "1234"  
  db:  
    password: "1234"
```

Para iniciar el cliente de MariaDB Galera que permite acceder a la base de datos se utilizó un *Job*. En el archivo *mariadbDBGalera.yaml* se define un ConfigMap y el Job.

Dentro del Job se declaran las variables de entorno y se asigna el valor de cada una. Se declaran las variables de entorno para el *host* y *password*. También se utilizan los valores de las configuraciones que guardamos en *stateless/values.yaml* para asignar la imagen que le corresponde al Pod y asignar nombres a volúmenes u otros.

Asimismo, en este Job se indica el comando que se debe ejecutar en *args*, el cual se encarga de cargar el archivo sql (*babynames.sql*) que contiene las tablas y los procedimientos almacenados de la base de datos.

## PostGreSQL

Para instalar el cliente de PostGreSQL se creó la imagen de PostGreSQL que incluye al cliente utilizando un Dockerfile. En nombre de esta imagen se guarda en el archivo ubicado en *stateless/values.yaml* que contiene otros valores de configuración como el *mapName*, *name* y *volumeName*.

```
postgresql:  
  mapName: script-db  
  name: postgresql-init  
  volumeName: scripts  
  image: moisose/postgresql-client
```

En *charts/databases/values.yaml* se tienen algunas configuraciones para la instancia de PostGreSQL, como datos necesarios para la autenticación (contraseña y nombre de base de datos), el namespace y también podemos activar o desactivar la ejecución del pod de PostGreSQL.

```

postgresql:
  enabled: false
  metrics:
    enabled: true
    serviceMonitor:
      enabled: true
      namespace: "monitoring"
  auth:
    postgresPassword: "1234"
    database: "babynames"

```

Para iniciar el cliente de PostGreSQL que permite acceder a la base de datos se utilizó un *Job*. En el archivo *postgresql.yaml* se define un ConfigMap y el Job.

Dentro del Job se declaran las variables de entorno y se asigna el valor de cada una. Se declaran las variables de entorno para el *host* y *password*. También se utilizan los valores de las configuraciones que guardamos en *stateless/values.yaml* para asignar la imagen que le corresponde al Pod y asignar nombres a volúmenes u otros.

Asimismo, en este Job se indica el comando que se debe ejecutar en *args*, el cual se encarga de cargar el archivo sql (*postgres.sql*) que contiene las tablas y los procedimientos almacenados de la base de datos.

## PostGreSQL High Availability

Para instalar el cliente de PostGreSQL High Availability se creó la misma imagen de PostGreSQL que incluye al cliente utilizando un Dockerfile. En nombre de esta imagen se guarda en el archivo ubicado en *stateless/values.yaml* que contiene otros valores de configuración como el *mapName*, *name* y *volumeName*. Se utiliza la misma configuración de PostGreSQL.

```

postgresql:
  mapName: script-db
  name: postgresql-init
  volumeName: scripts
  image: moisose/postgresql-client

```

En *charts/databases/values.yaml* se tienen algunas configuraciones para la instancia de PostGreSQL High Availability, como datos necesarios para la autenticación (contraseñas), el namespace y también podemos activar o desactivar la ejecución del pod de PostGreSQL.

```
postgresqlha:  
  enabled: false  
  metrics:  
    enabled: true  
    serviceMonitor:  
      enabled: true  
      namespace: "monitoring"  
global:  
  postgresql:  
    password: "1234"  
  pgpool:  
    adminPassword: "12345"
```

Para iniciar el cliente de PostGreSQL High Availability que permite acceder a la base de datos se utilizó un *Job*. En el archivo *postgresqlha.yaml* se define un ConfigMap y el Job.

Dentro del Job se declaran las variables de entorno y se asigna el valor de cada una. Se declaran las variables de entorno para el *host* y *password*. También se utilizan los valores de las configuraciones que guardamos en *stateless/values.yaml* para asignar la imagen que le corresponde al Pod y asignar nombres a volúmenes u otros.

Asimismo, en este Job se indica el comando que se debe ejecutar en *args*, el cual se encarga de cargar el archivo sql (*postgres.sql*) que contiene las tablas y los procedimientos almacenados de la base de datos.

## Otros componentes

---

### API con Flask

Para realizar las pruebas de carga se necesita usar Gatling, pero no todas las bases de datos tienen una interfaz para utilizar endpoints HTTP, por lo que se necesitó usar una aplicación intermediaria.

Esta aplicación se desarrolló en Python utilizando la librería Flask. Cada base de datos utiliza un API distinto.

Para poder ejecutar un API de la base de datos se debe tener un Deployment corriendo en Kubernetes. Para lograr esto, se creó la imagen de cada API con un Dockerfile. El nombre de las imágenes se guarda en el archivo ubicado en *stateless/values.yaml* junto a otras configuraciones que servirán para configurar el Deployment, como *name* y *nameApp*.

```

apiMariaDBDeployment:
  replicas: 1
  name: apimariadb-deployment
  image: moisose/apimariadb
  nameApp: apimariadb-deployment
apiMariaDBGDeployment:
  replicas: 1
  name: apimariadbga-deployment
  image: moisose/apimariadb
  nameApp: apimariadbga-deployment
apiPostgresDeployment:
  replicas: 1
  name: apipostgresha-deployment
  image: moisose/apipostgres
  nameApp: apipostgresha-deployment
apiPostgresDeployment:
  replicas: 1
  name: apipostgres-deployment
  image: moisose/apipostgres
  nameApp: apipostgresha-deployment

```

El Deployment de cada API se ubica en la carpeta charts/stateless/templates. Todos los Deployments siguen la misma estructura: Primero se define el servicio de tipo Deployment y luego un Servicio de NodePort para exponer el puerto y que se pueda comunicar el API con la base de datos.

En el archivo yaml del Deployment de cada API se accede a los valores definidos en el archivo values.yaml para configurar el Deployment, como nombres de volúmenes o labels. También se definen las variables de entorno con sus valores.

El servicio NodePort de todos los Deployments están configurados en el port 5000, targetPort 5000, nodePort 3000 y porotocol TCP. Aquí también se accede a los valores definidos en el archivo values.yaml para configurar algunos aspectos como el nombre de la app del selector o labels.

## Pruebas de carga

---

### Dataset

#### Dataset utilizado: “babynames.csv”

Con respecto al dataset utilizado, el cual tiene el nombre de “babynames.csv”, contiene datos relacionados con los nombres de bebés más populares por sexo y origen étnico de la madre en la ciudad de Nueva York entre los años 2011 y 2014. En resumen, este dataset posee un total de, 13962 filas y 6 columnas. Los datos incluyen elementos tales como:

- **BRTH\_YR:** Año de nacimiento del bebe.
- **GNDR:** Género.
- **ETHCTY:** Etnia de la madre.
- **NM:** Nombre del bebe.
- **CNT:** Recuento del nombre.

- **RNK:** Clarificación del nombre.

## Configuración de las Pruebas

Se realizó una prueba de carga con la siguiente configuración para los usuarios:

```
// Users per second
setUp(
    //Create - Load test 5 users per Second during 30 minutes
    createD.inject(constantUsersPerSec(5) during (1800 seconds)),
    // Select - Load test 10 users per Second during 30 minutes
    selectD.inject(constantUsersPerSec(10) during (1800 seconds)),
    //Update - Load test 2 users per Second during 30 minutes
    updateD.inject(constantUsersPerSec(2) during (1800 seconds)),
    //Delete - Load test 5 users per Second during 30 minutes
    deleteD.inject(constantUsersPerSec(3) during (1800 seconds))
).protocols(httpConf)
```

Nota: Para elasticsearch la configuración de los usuarios se hace de otra manera.

Para las distintas consultas se realizaron distintos escenarios, permitiendo simulación de usuarios haciendo distintas consultas de la siguiente manera:

- **Post - Create**

Se hace la simulación de 5 usuarios por segundo durante 30 minutos, para esto se generan escenario que va a mandar consultas post al http que se definió previamente.

```
//scenario for creating data
val createD = scenario("Crear datos")
    .exec(http("Create")
        .post("/babynames")
        .check(status.is(200)) /*Verifies status equals 200*/
    )
```

- **Get - Select**

Se hace la simulación de 2 usuarios por segundo durante 30 minutos, se genera el escenario para enviar consultas get al http que se definió previamente.

```
//scenario for get information
val selectD = scenario("Obtener datos")
    .exec(http("Select"))
    .get("/babynames")
    .check(status.is(200)) /*Verifies status equals 200*/
)
```

- **Put - Update**

Se hace la simulación de 2 usuarios por segundo durante 30 minutos, se genera el escenario para enviar consultas put al http que se definió previamente.

```
//scenario for updating data
val updateD = scenario("Actualizar datos")
    .exec(http("Update"))
    .put("/babynames")
    .check(status.is(200)) /*Verifies status equals 200*/
)
```

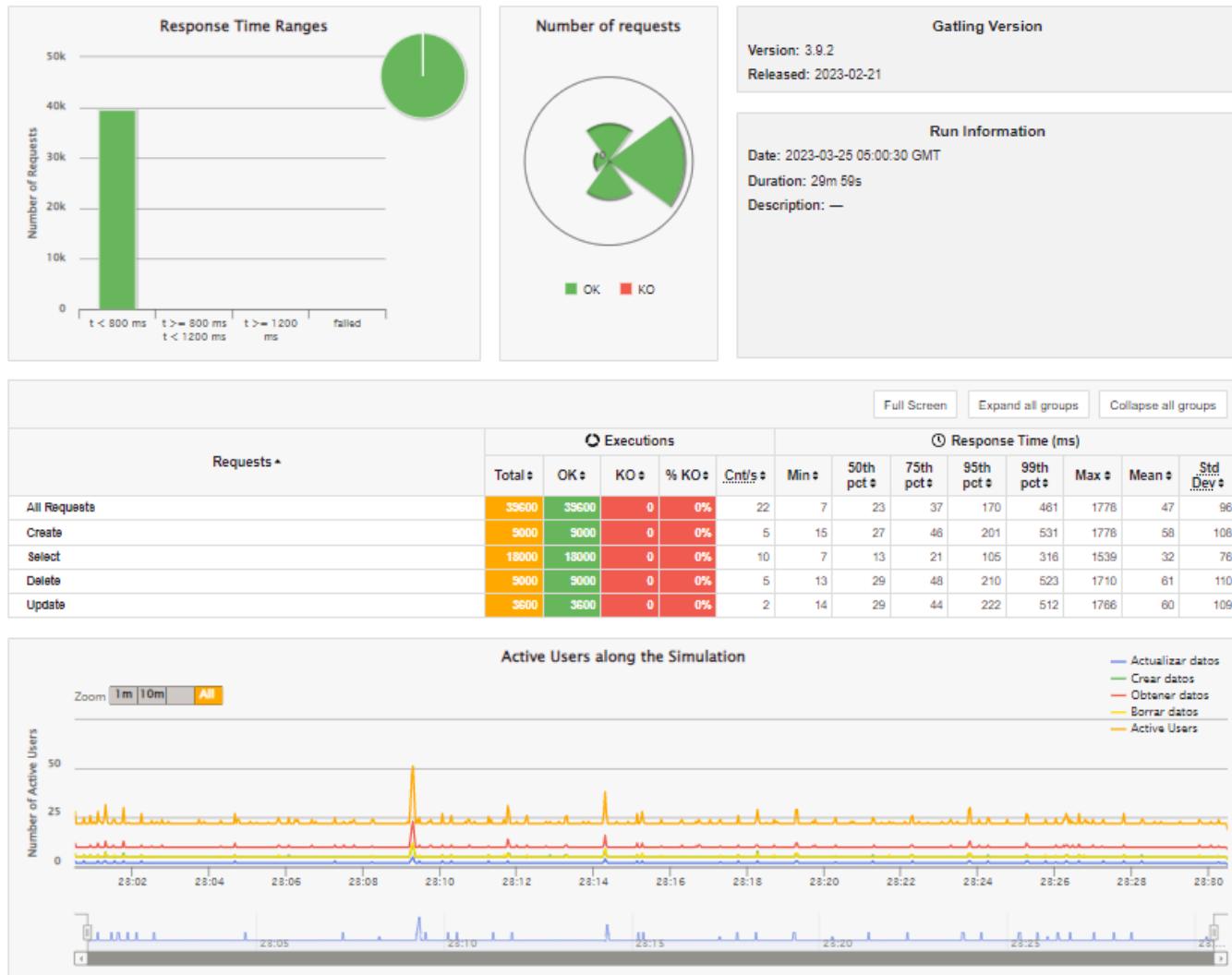
- **Delete - Delete**

Se hace la simulación de 3 usuarios por segundo durante 30 minutos, se genera el escenario para enviar consultas delete al http definido.

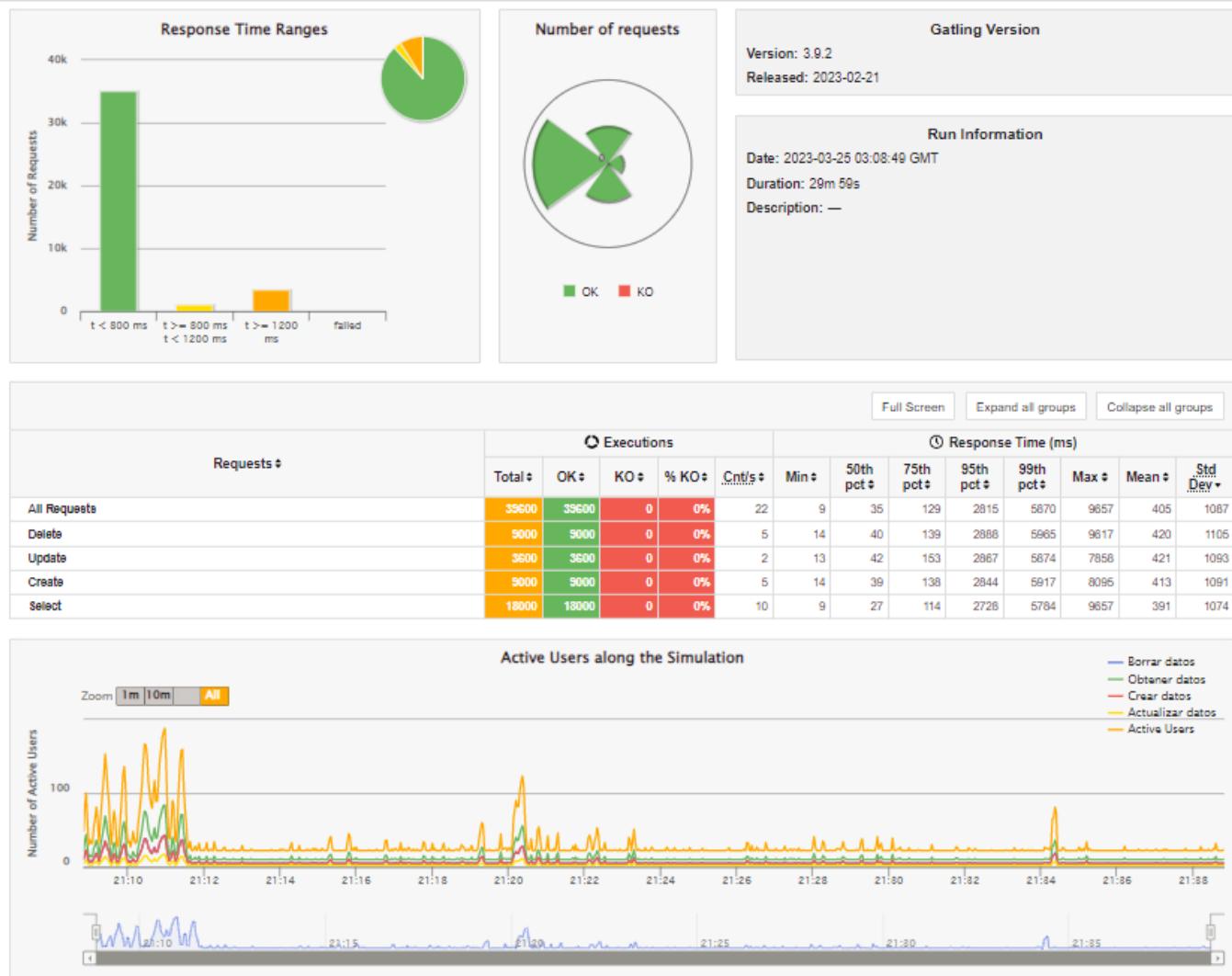
```
//scenario for deleting data
val deleteD = scenario("Borrar datos")
    .exec(http("Delete"))
    .delete("/babynames")
    .check(status.is(200)) /*Verifies status equals 200*/
)
```

## Resultados de las pruebas

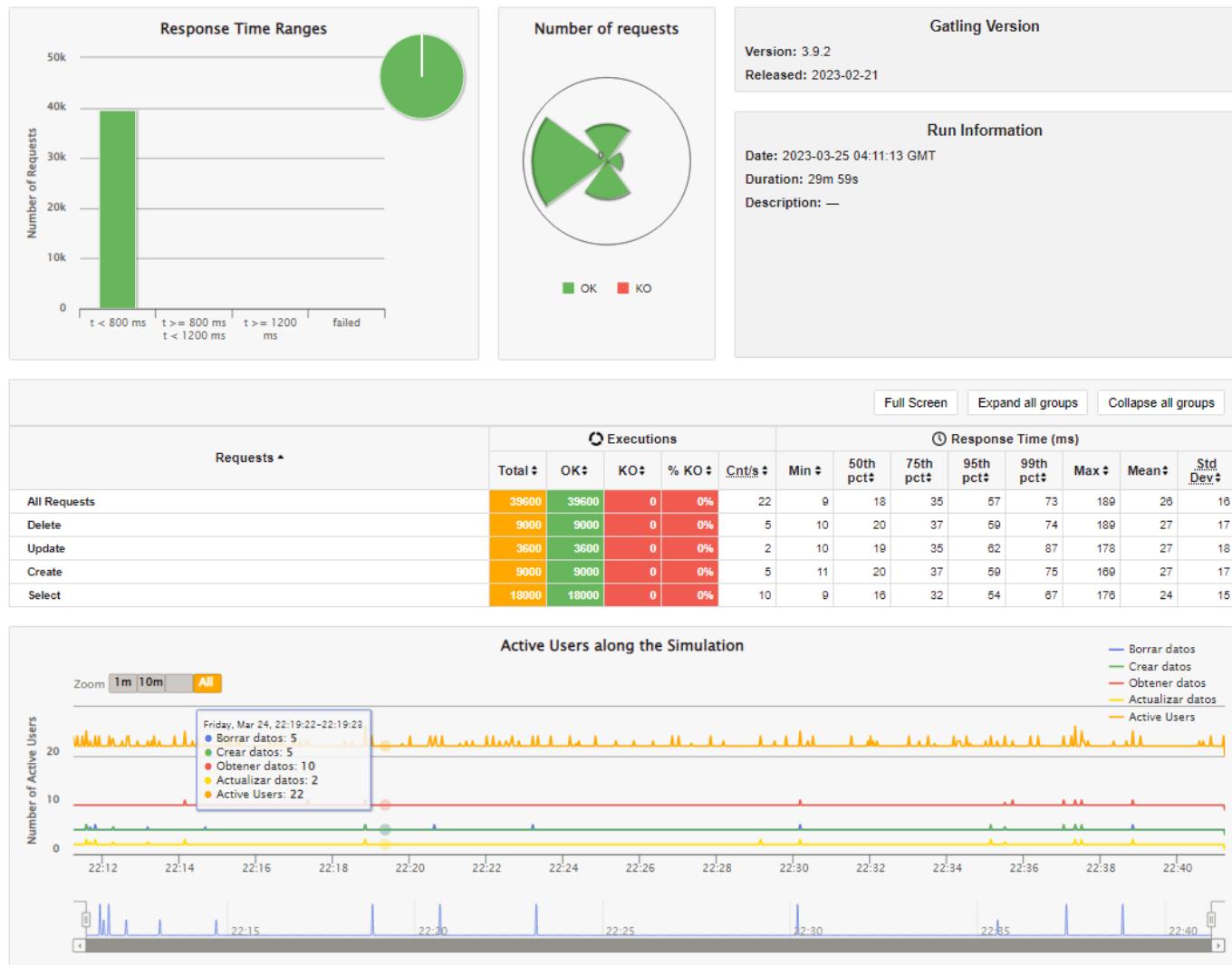
### 1- MariaDB



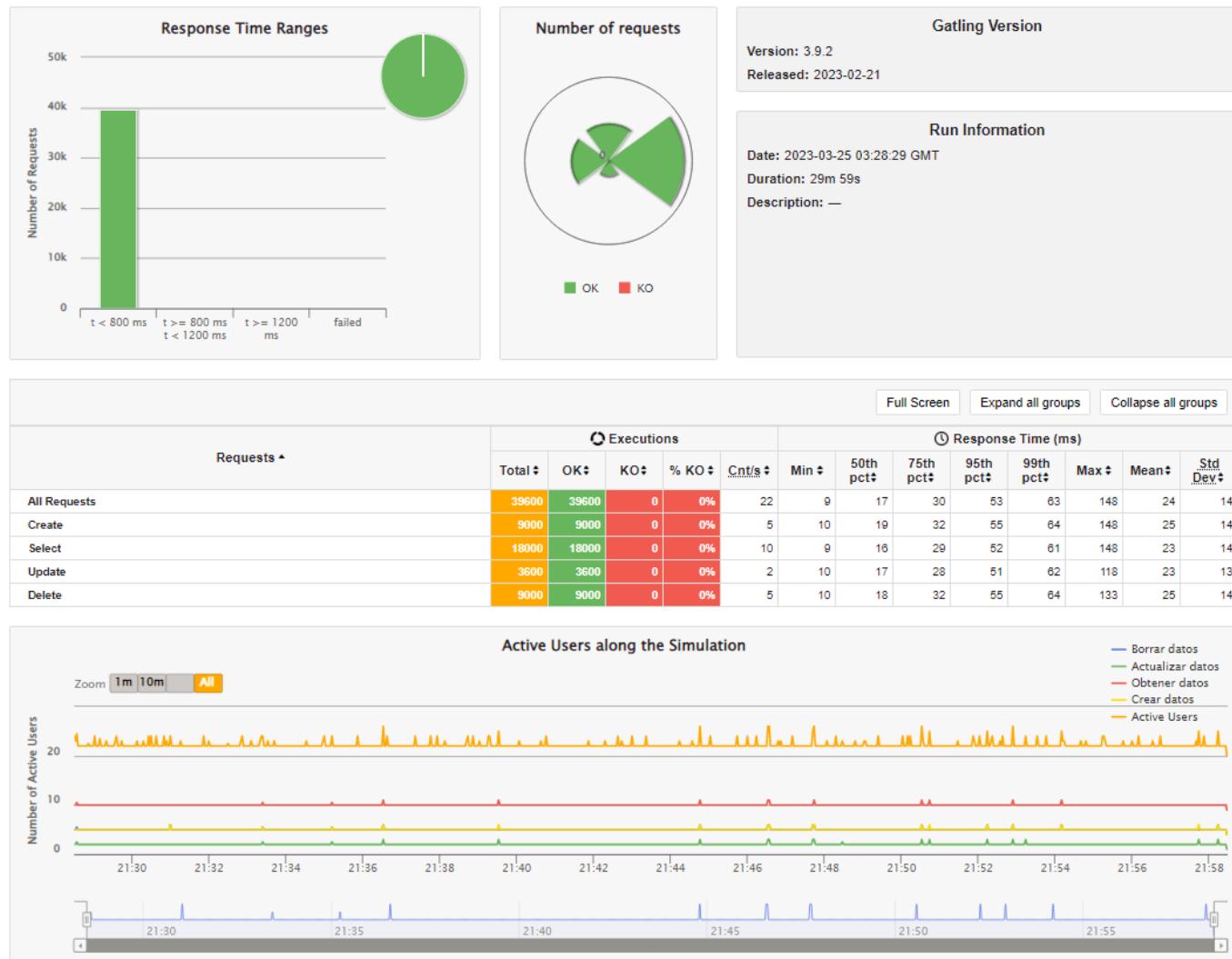
## 2- MariaDB Galera



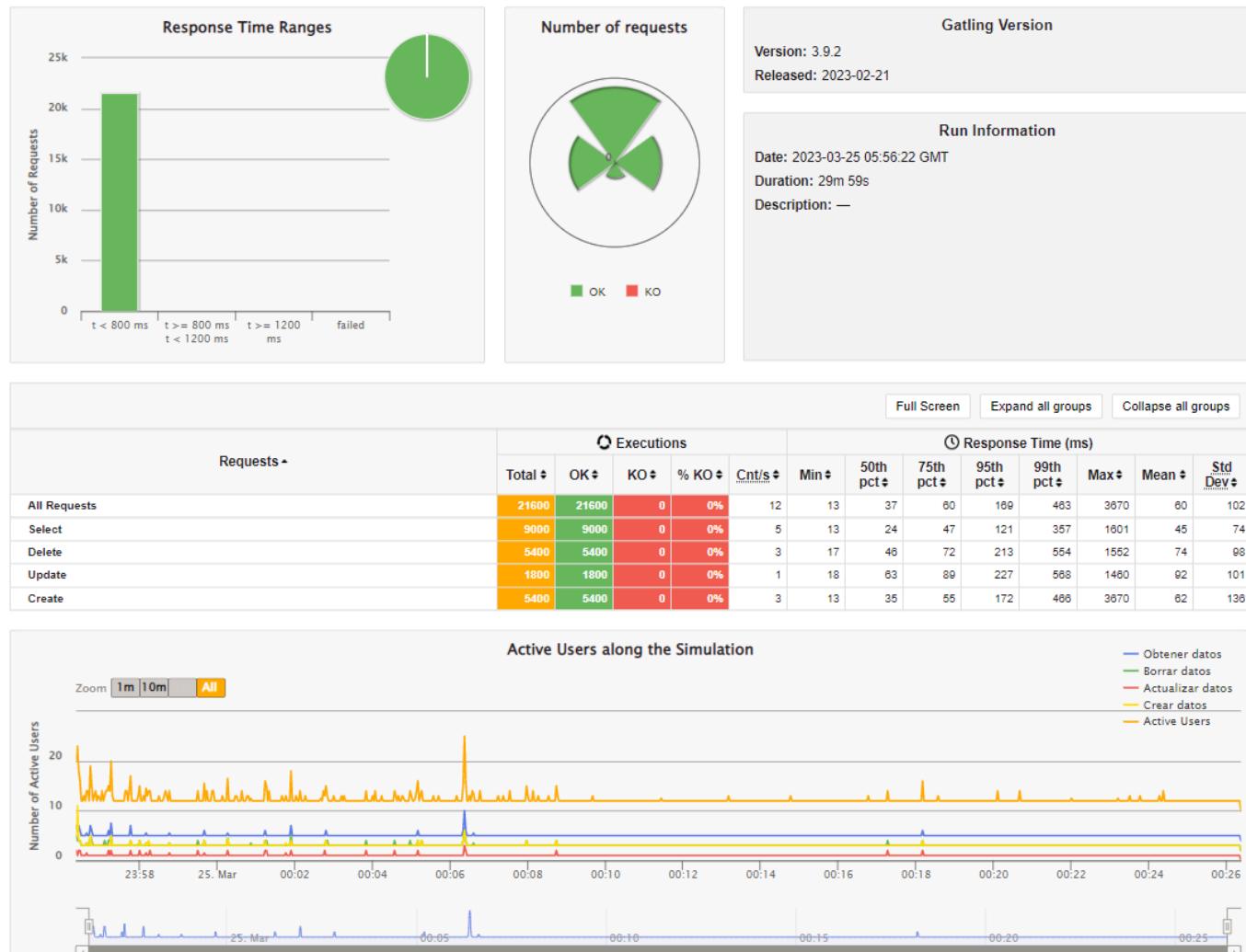
### 3- PostGreSQL



## 4- PostGRE HA



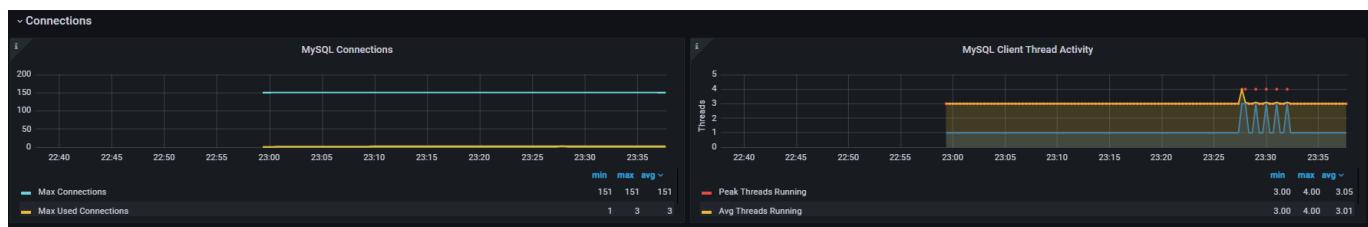
## 5- Elasticsearch

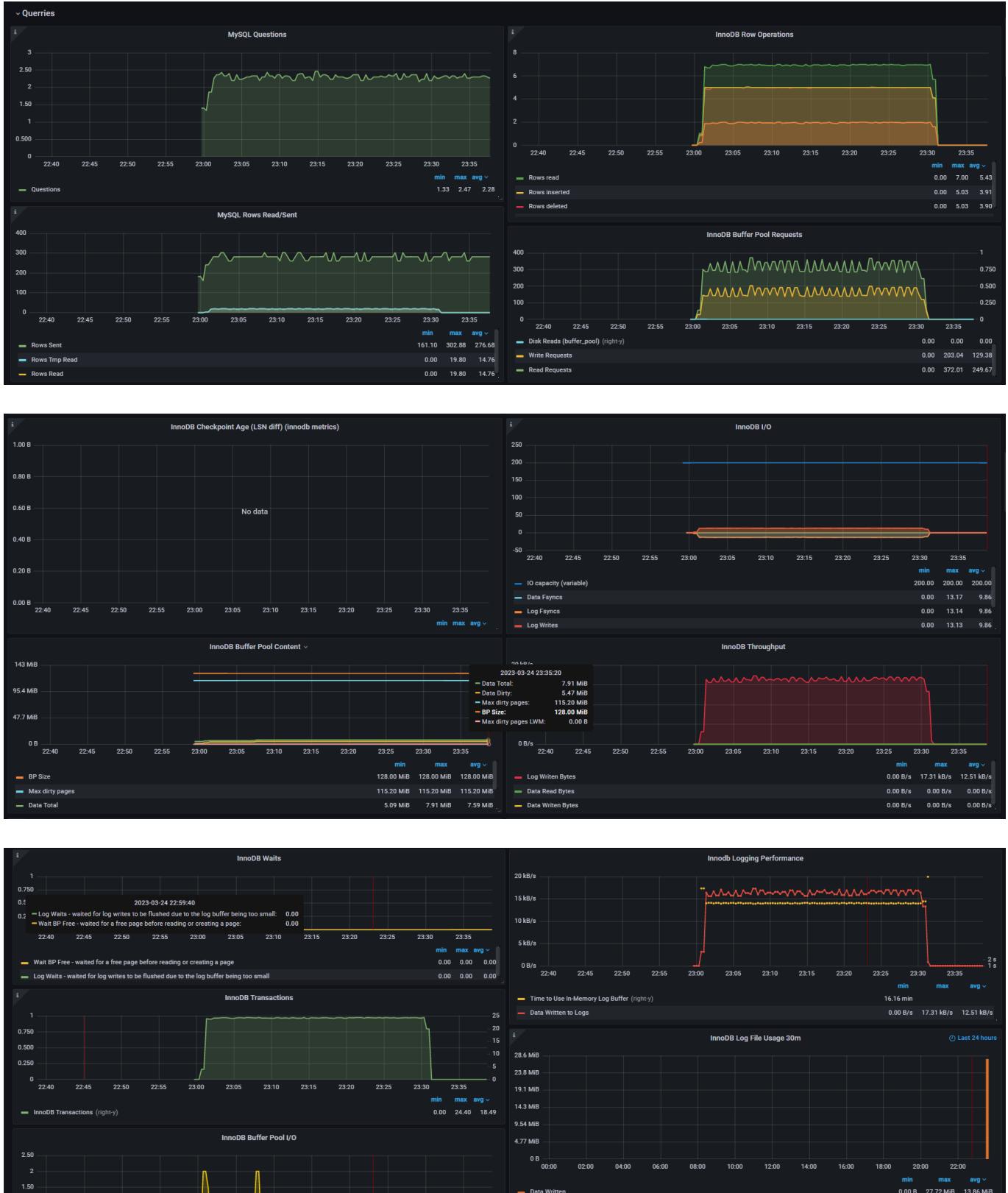


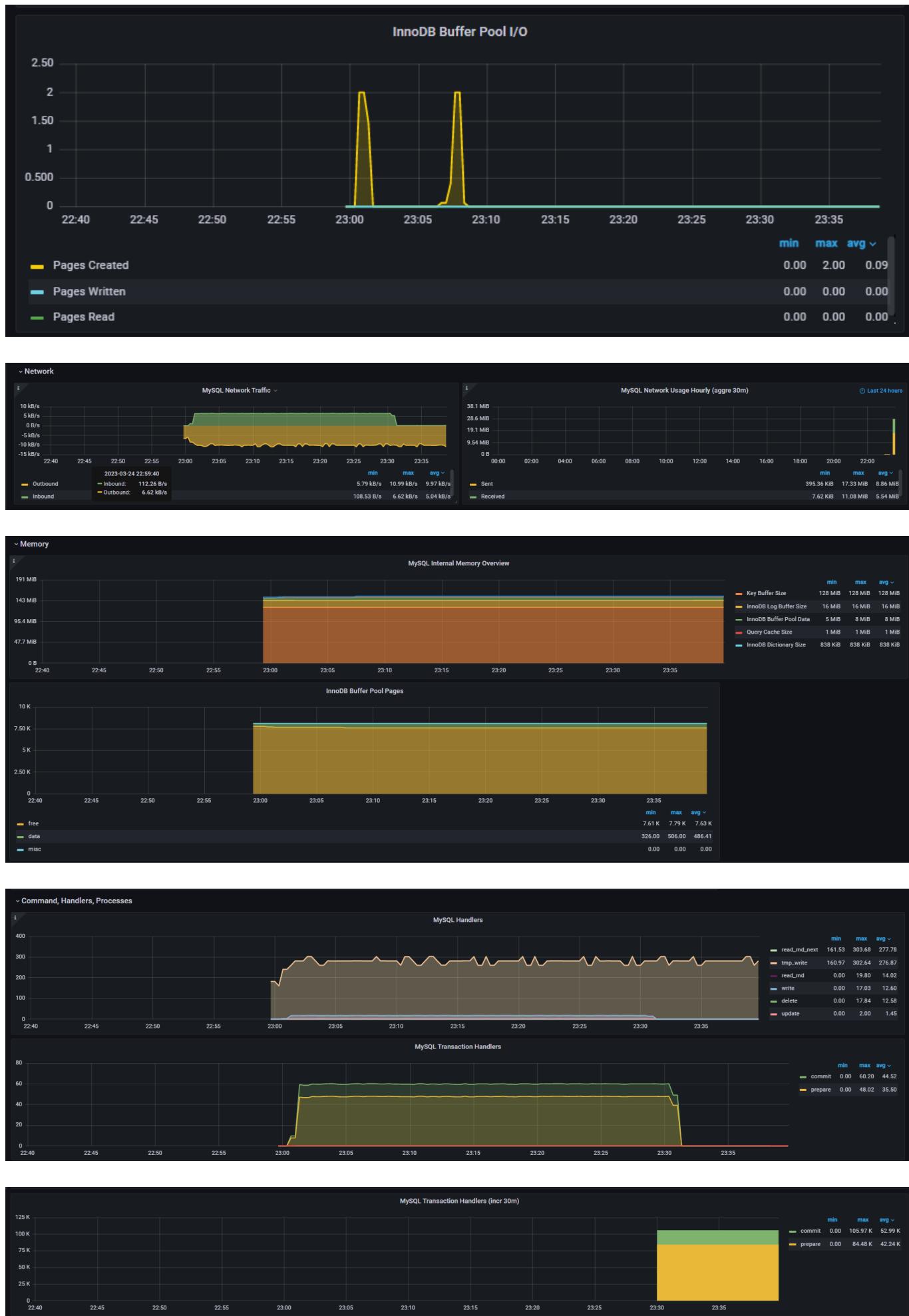
## Resultados en grafana

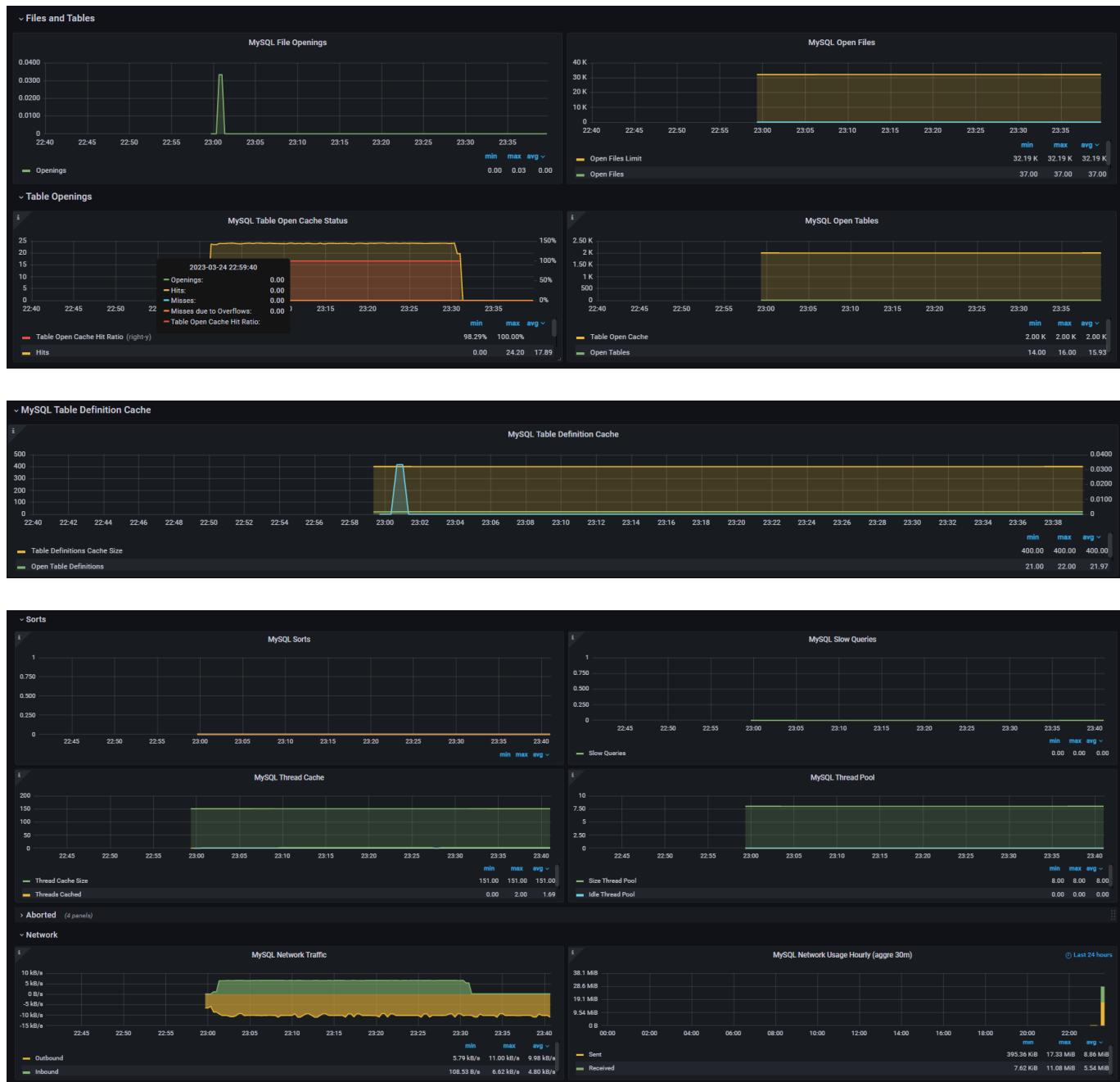
### 1- MariaDB

- Pruebas de creación, borrado, actualización y búsqueda



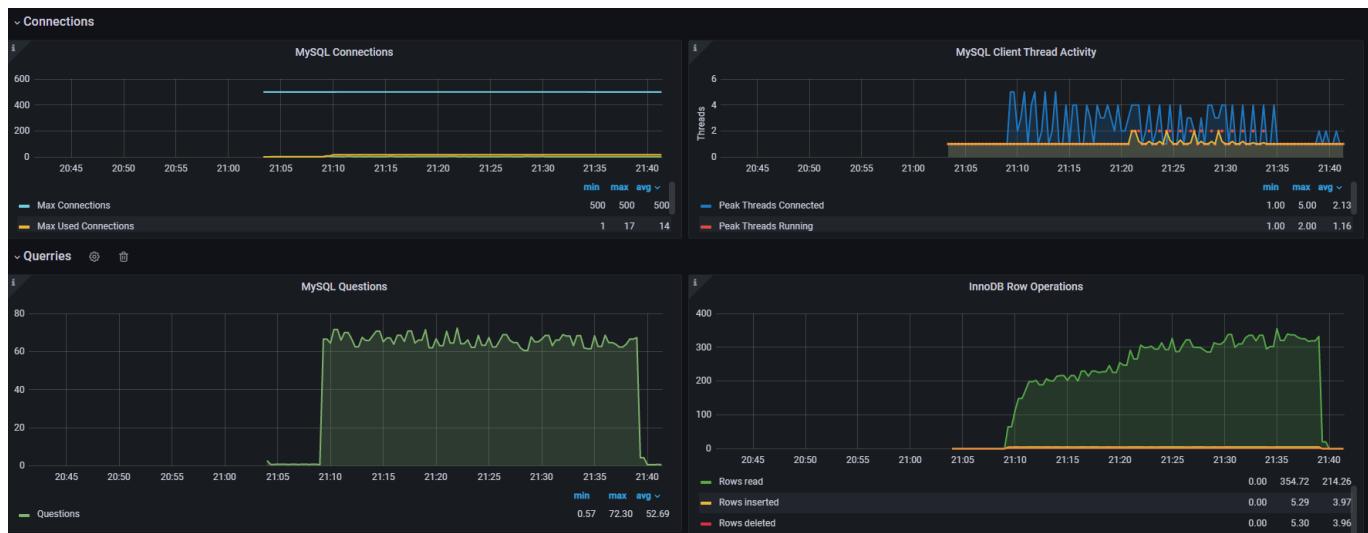


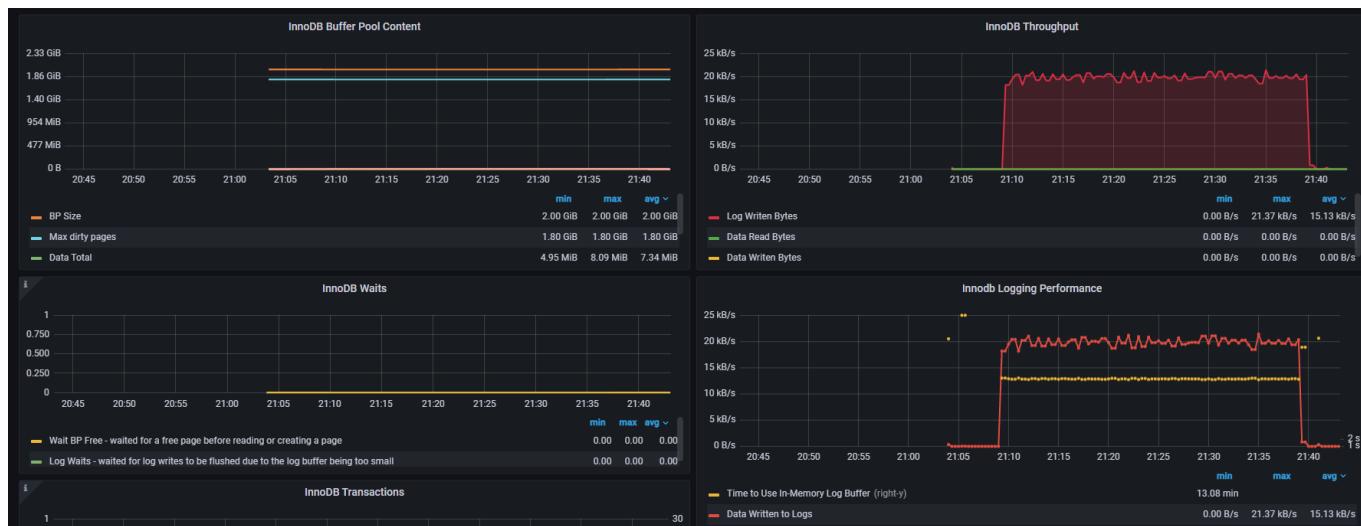
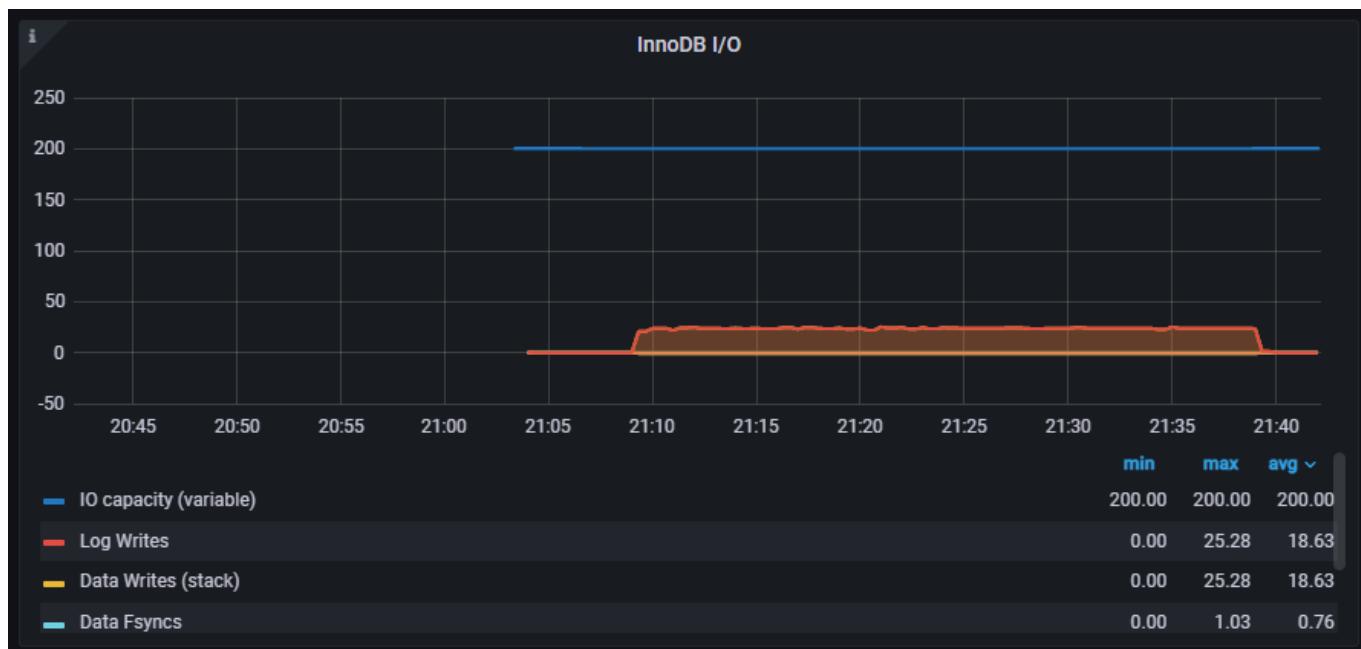
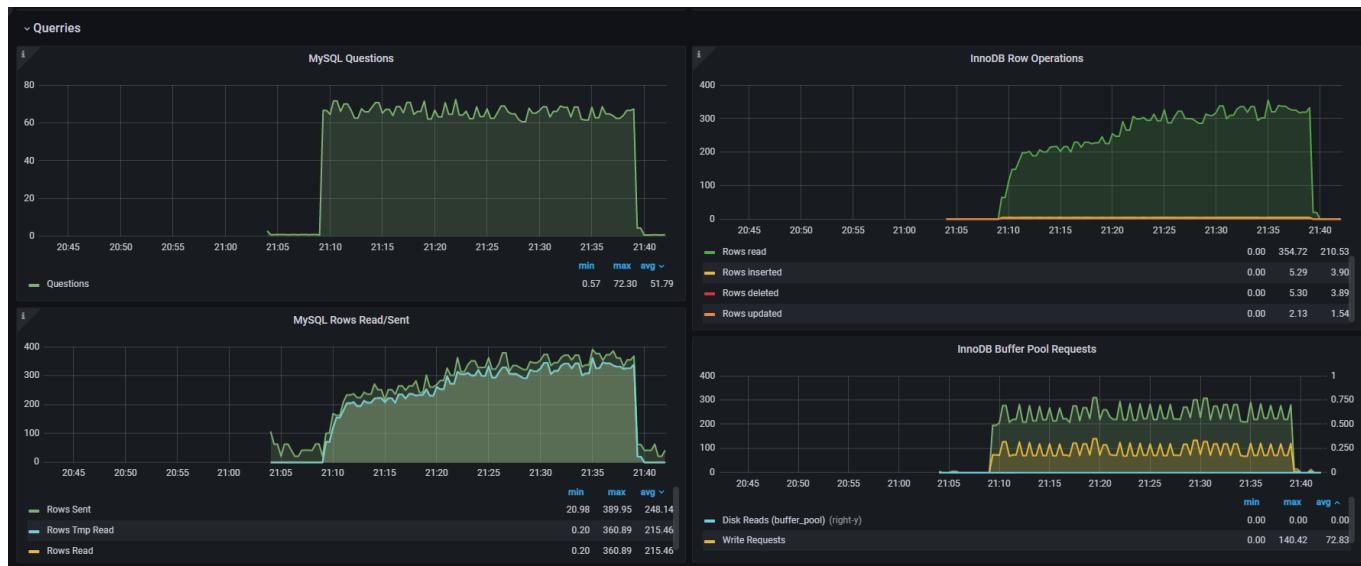


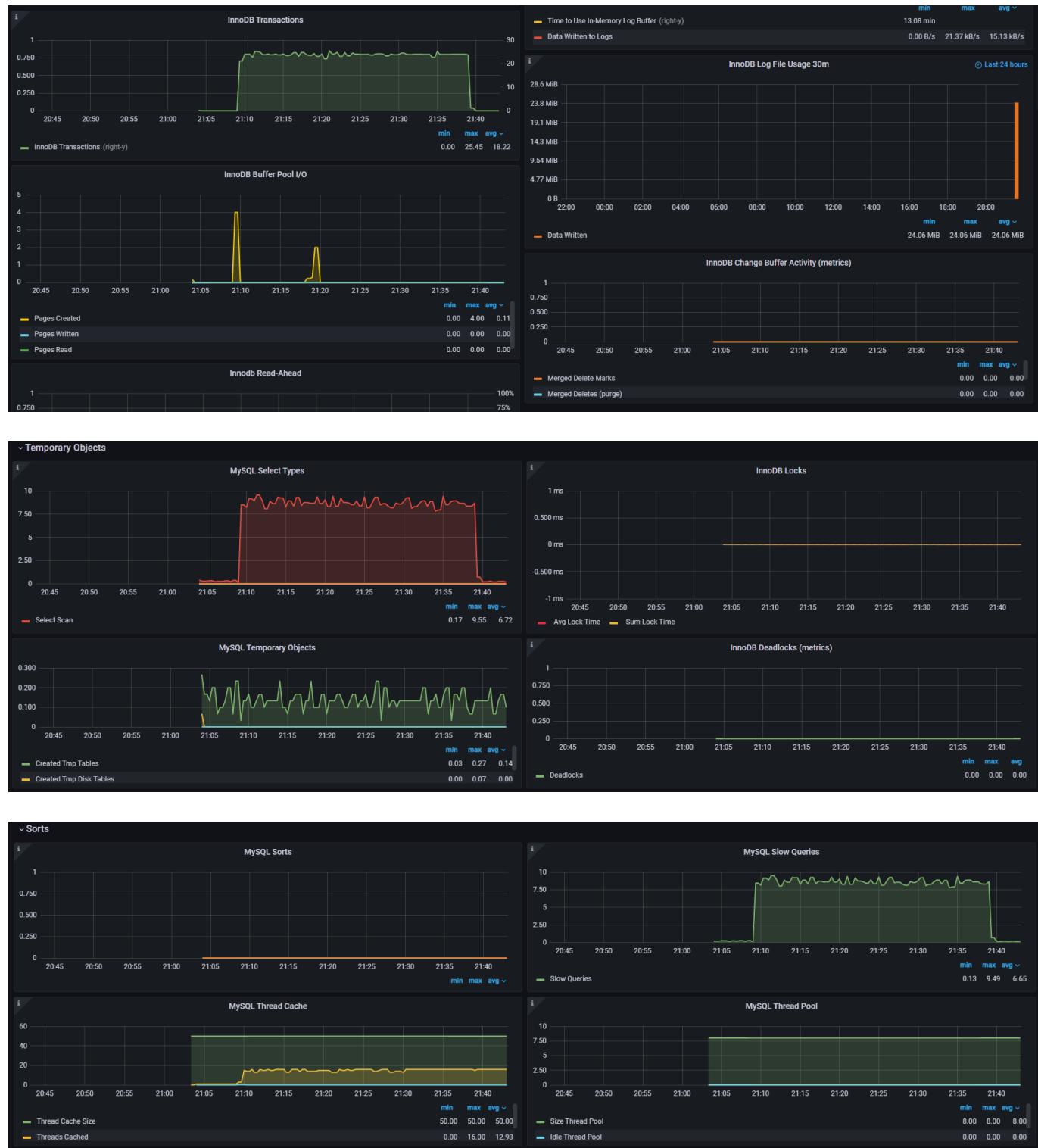


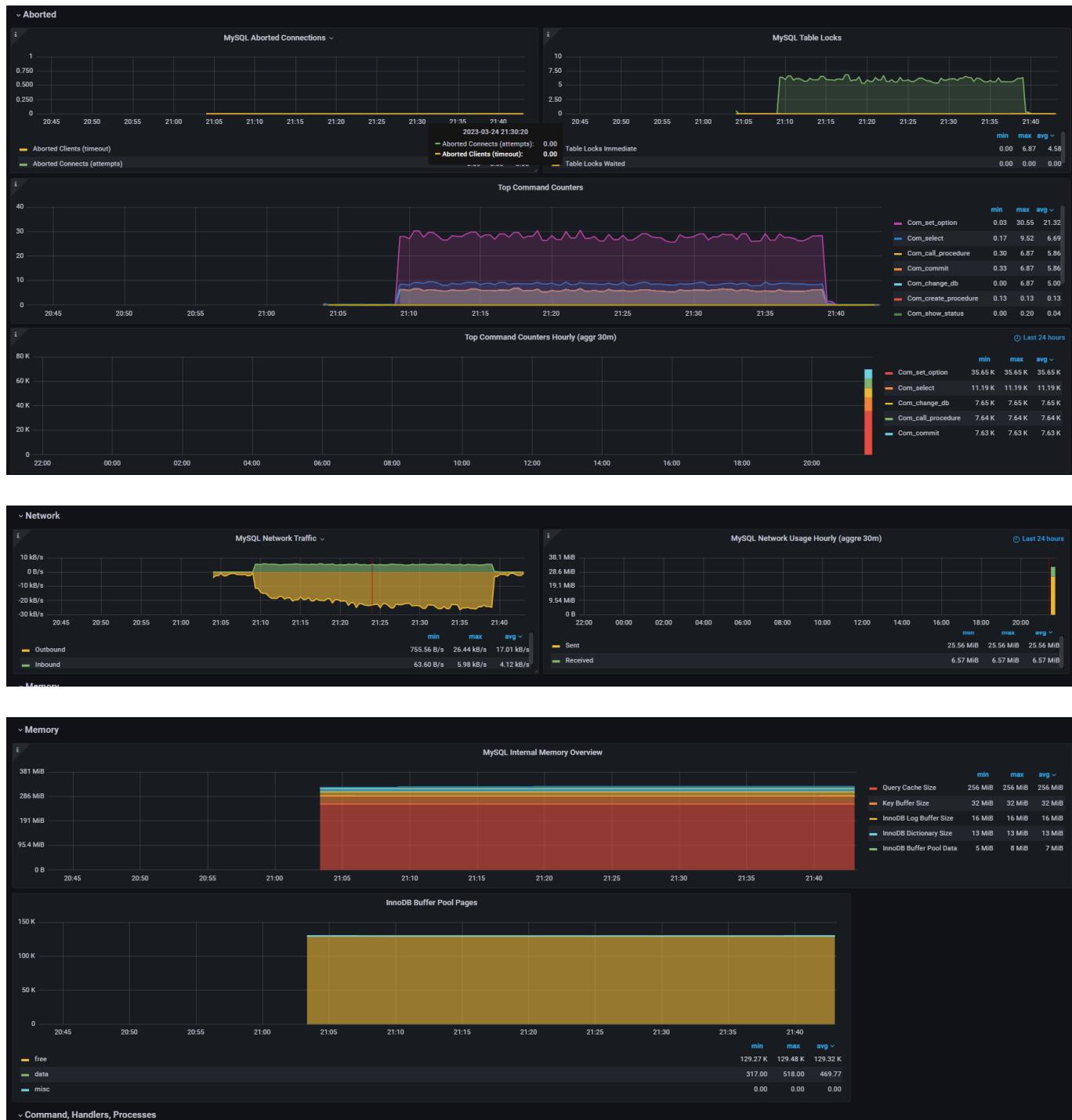
## 2- MariaDB Galera

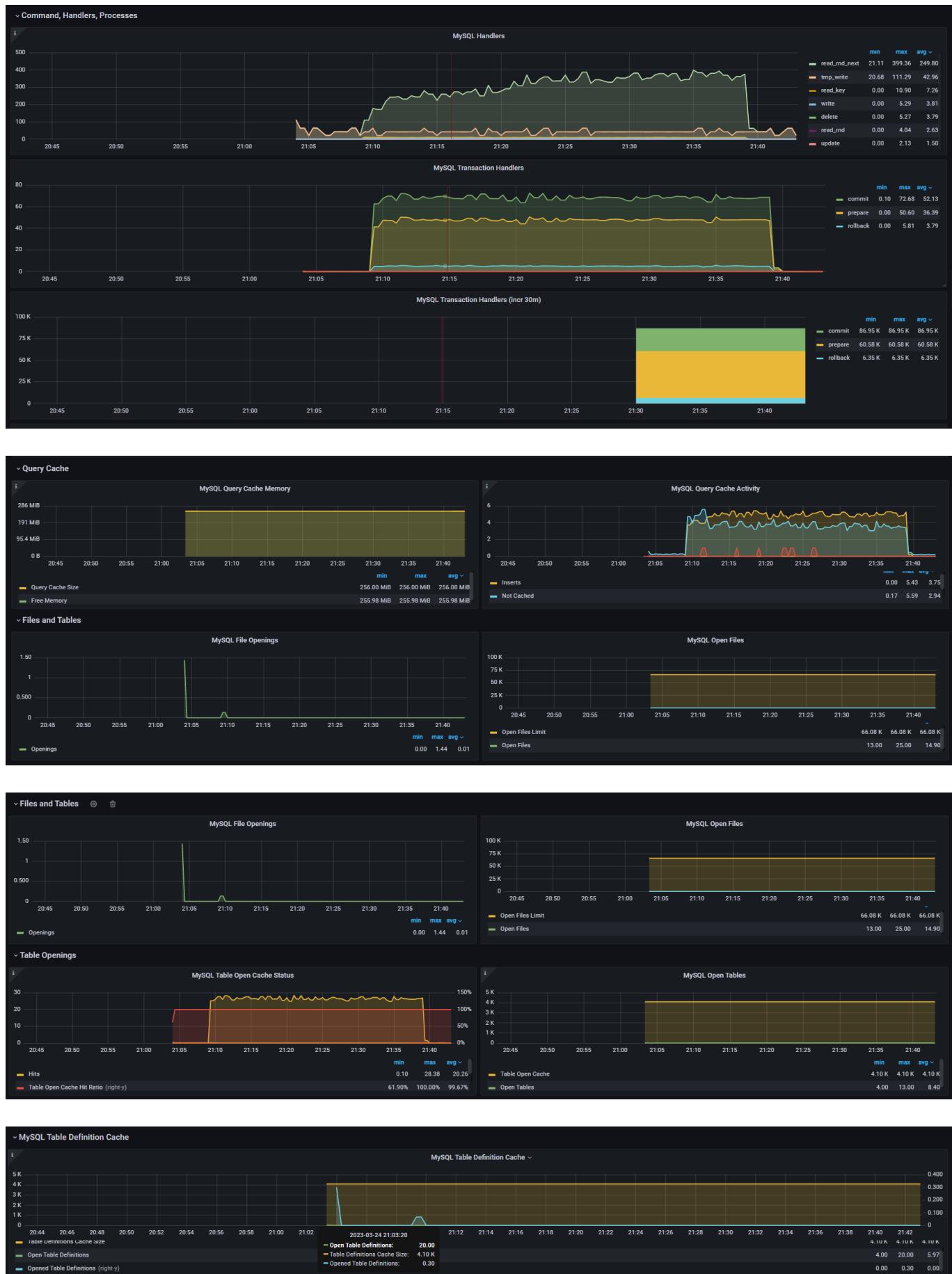
- Pruebas de creación, borrado, actualización y búsqueda





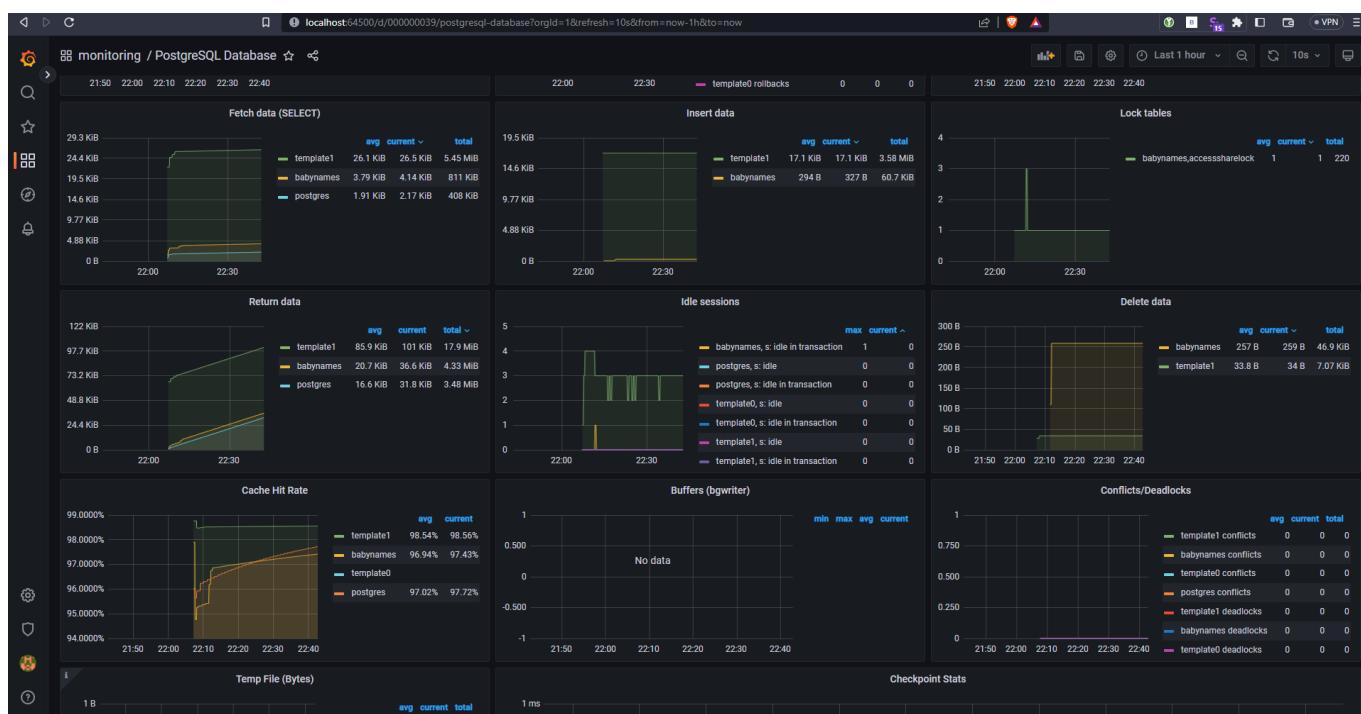
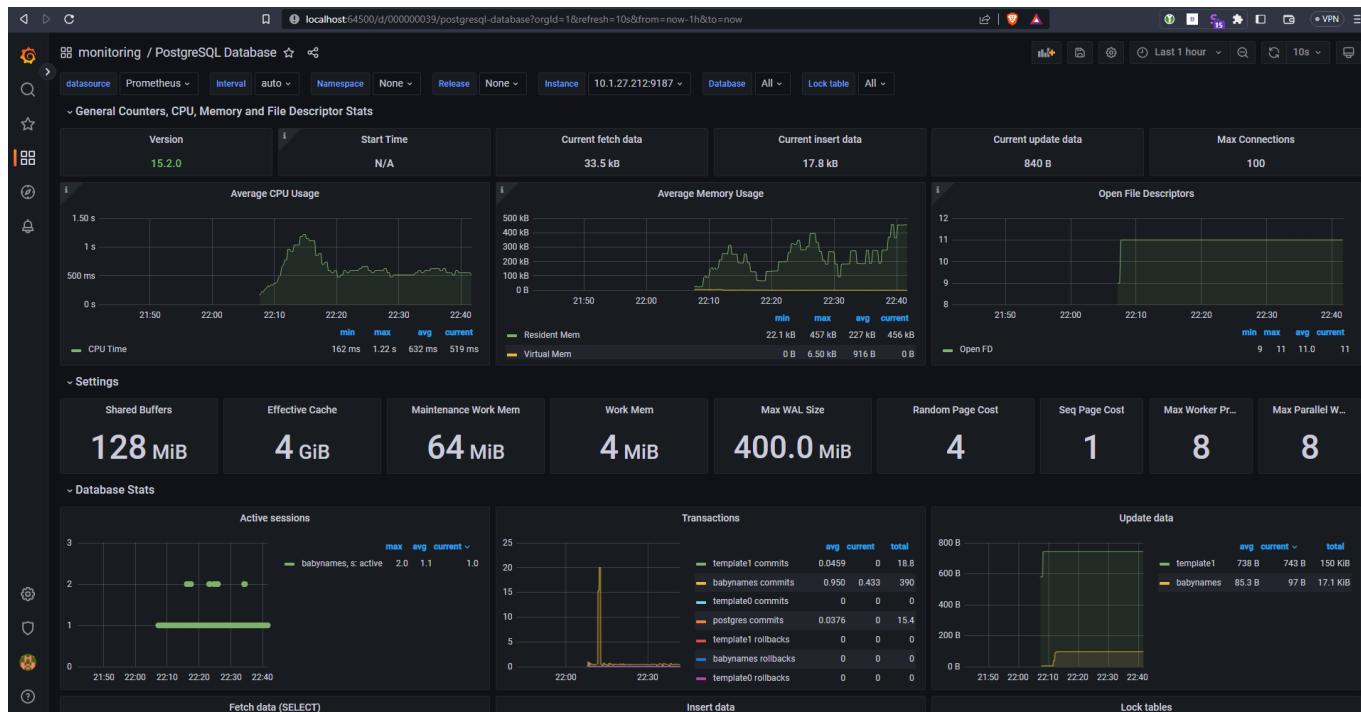


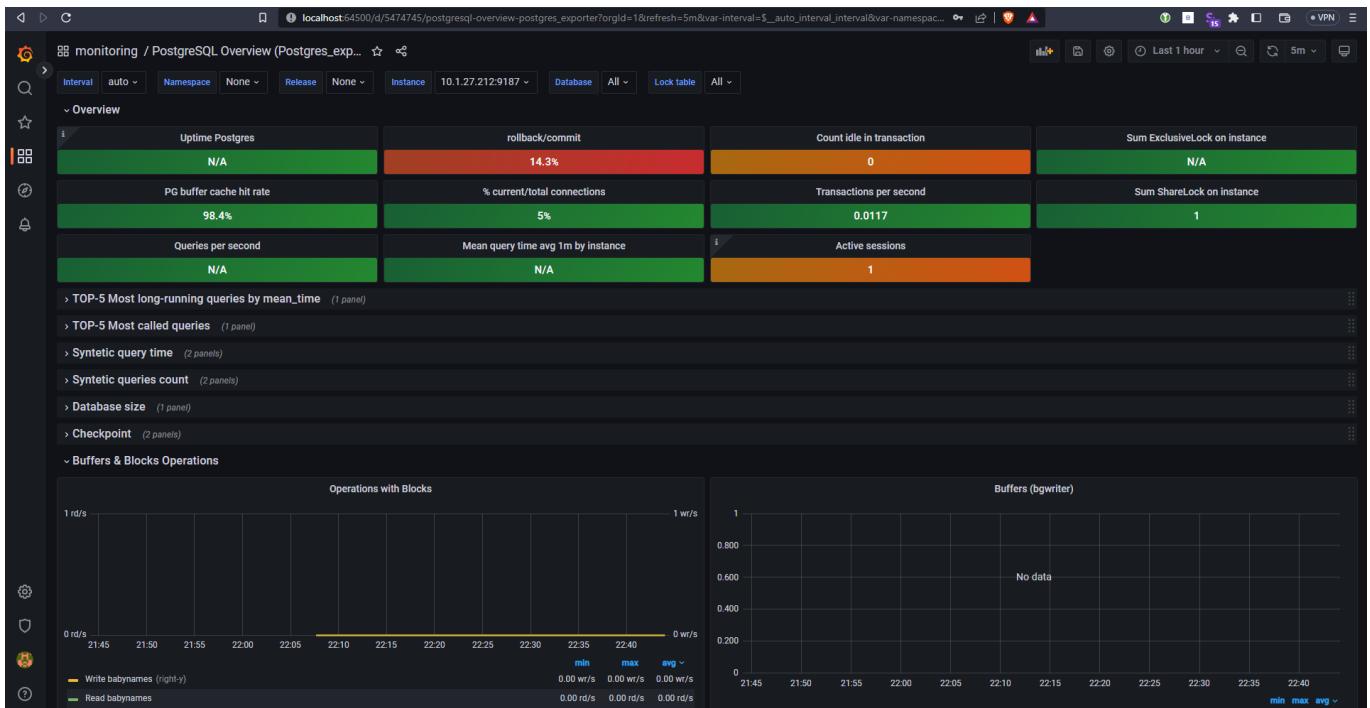
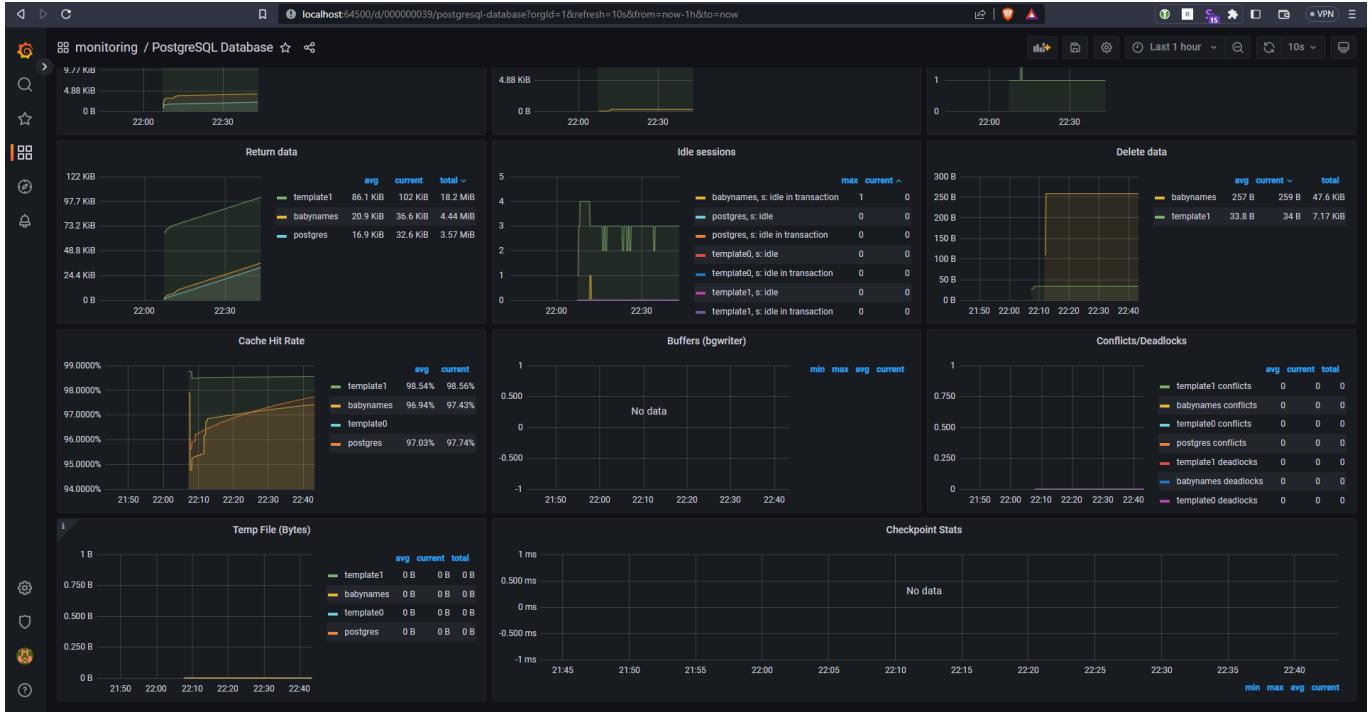


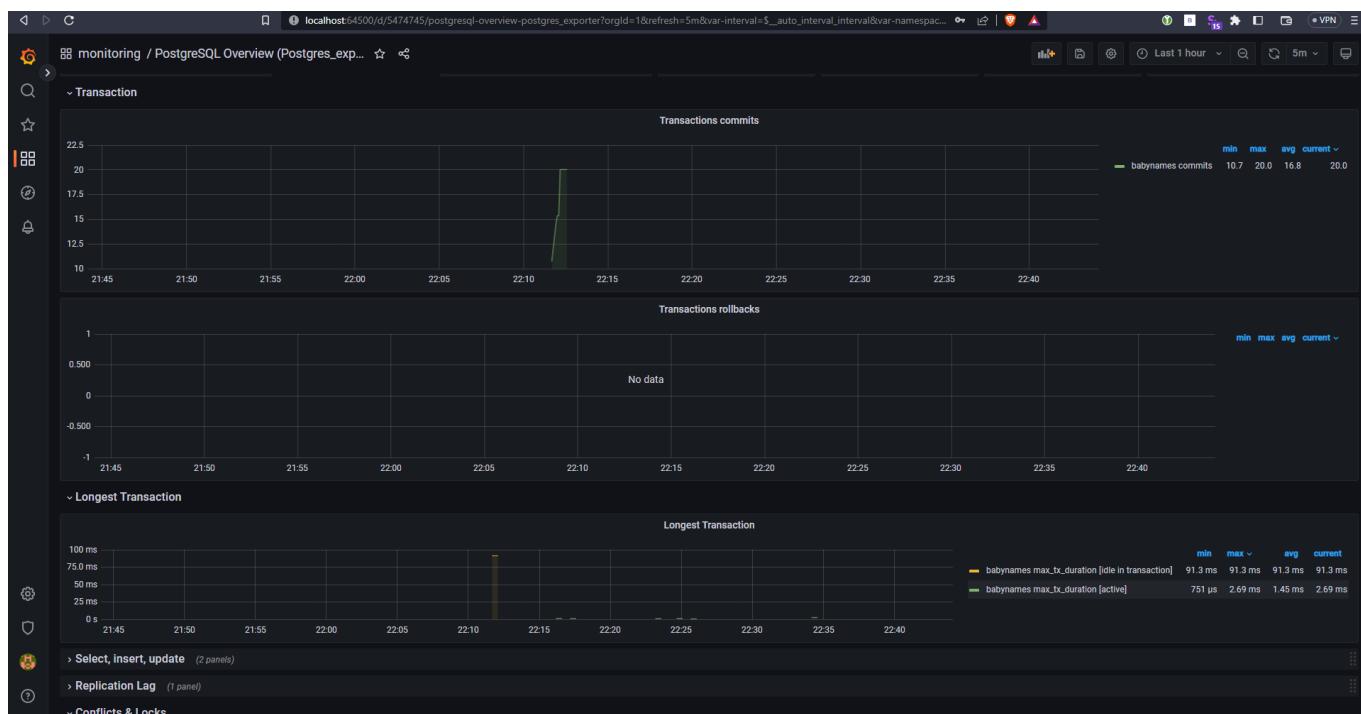
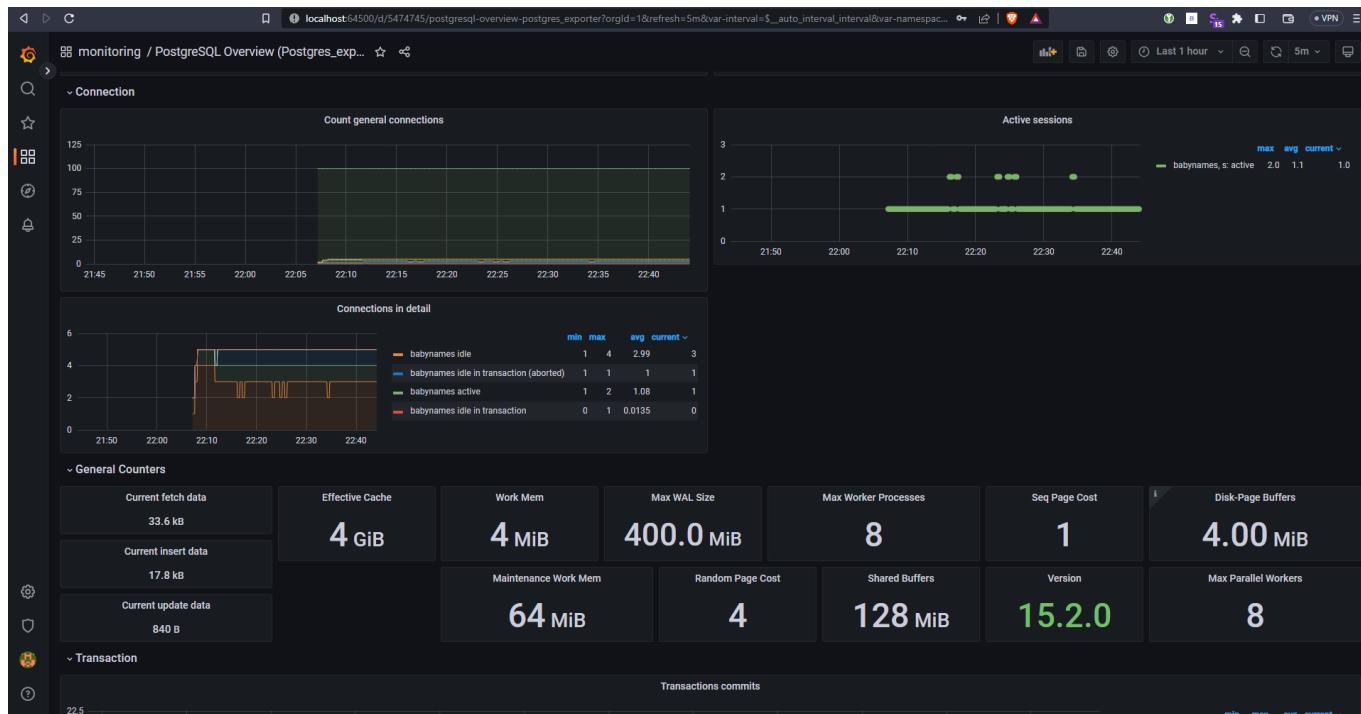


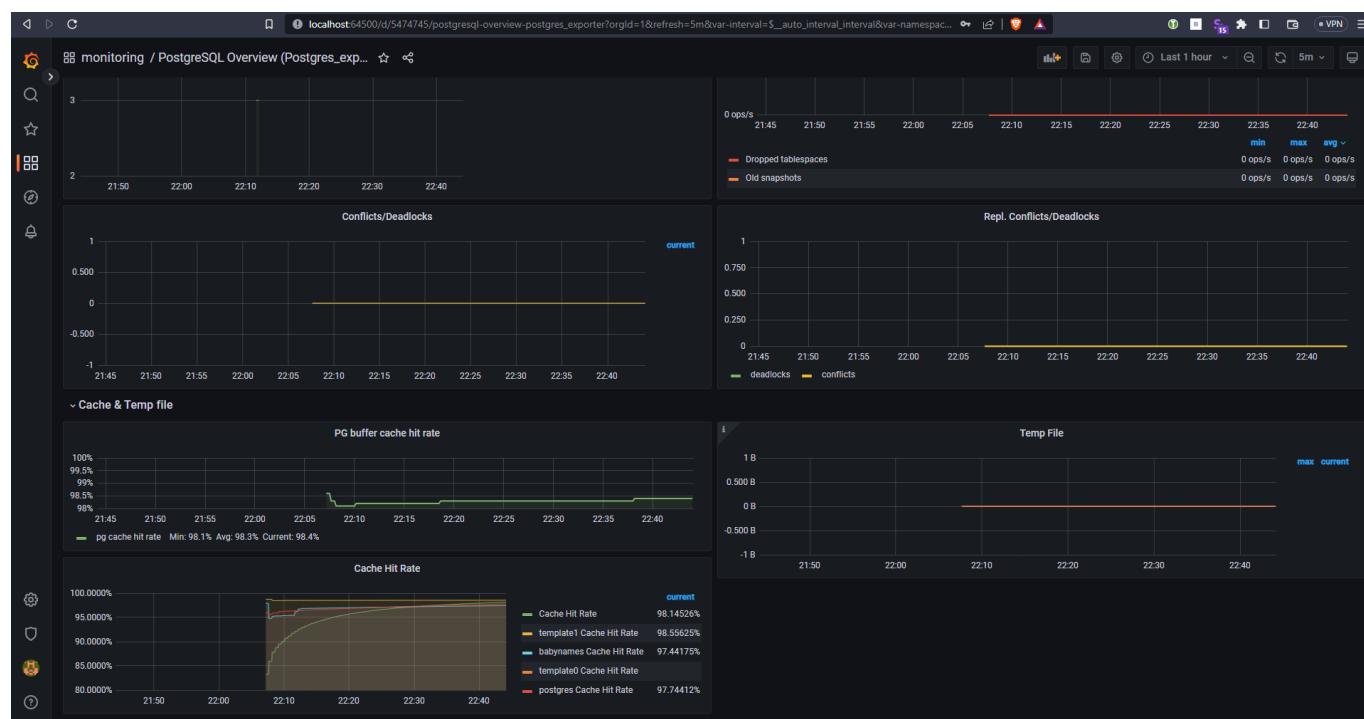
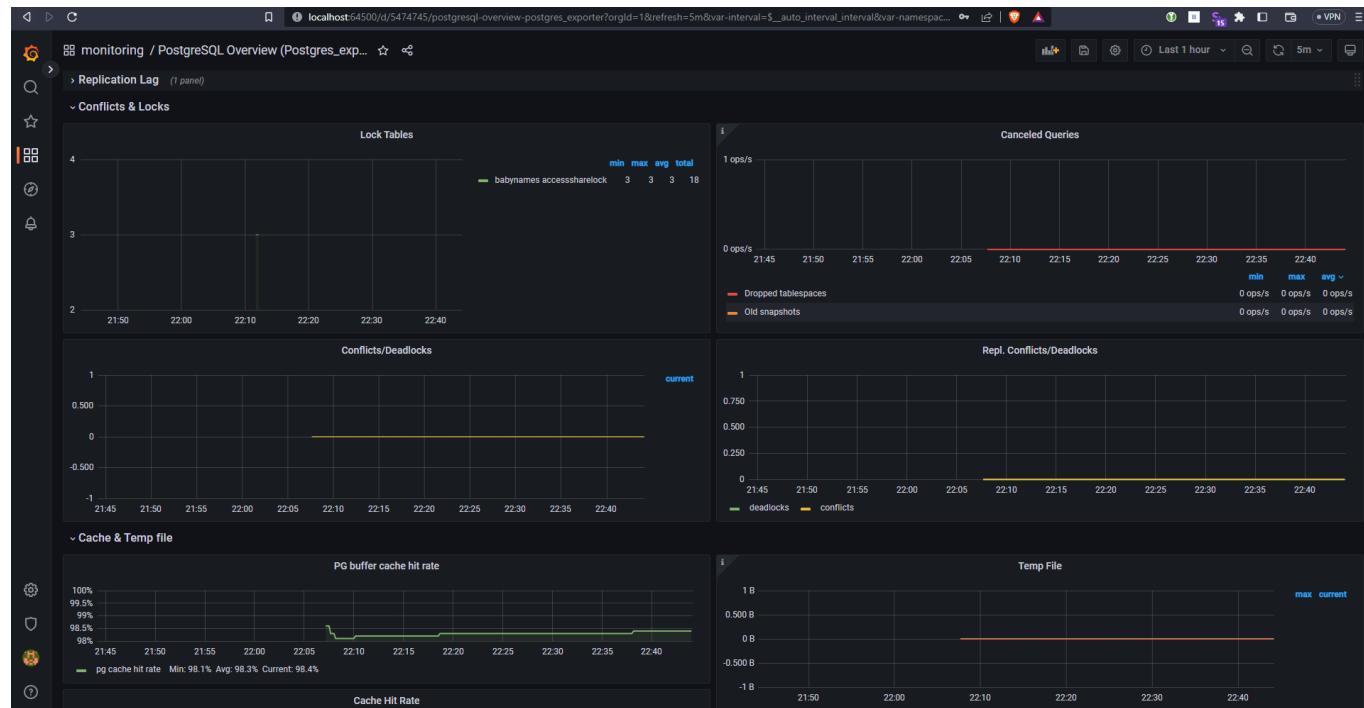
### 3- PostgreSQL

- Pruebas de creación, borrado, actualización y búsqueda



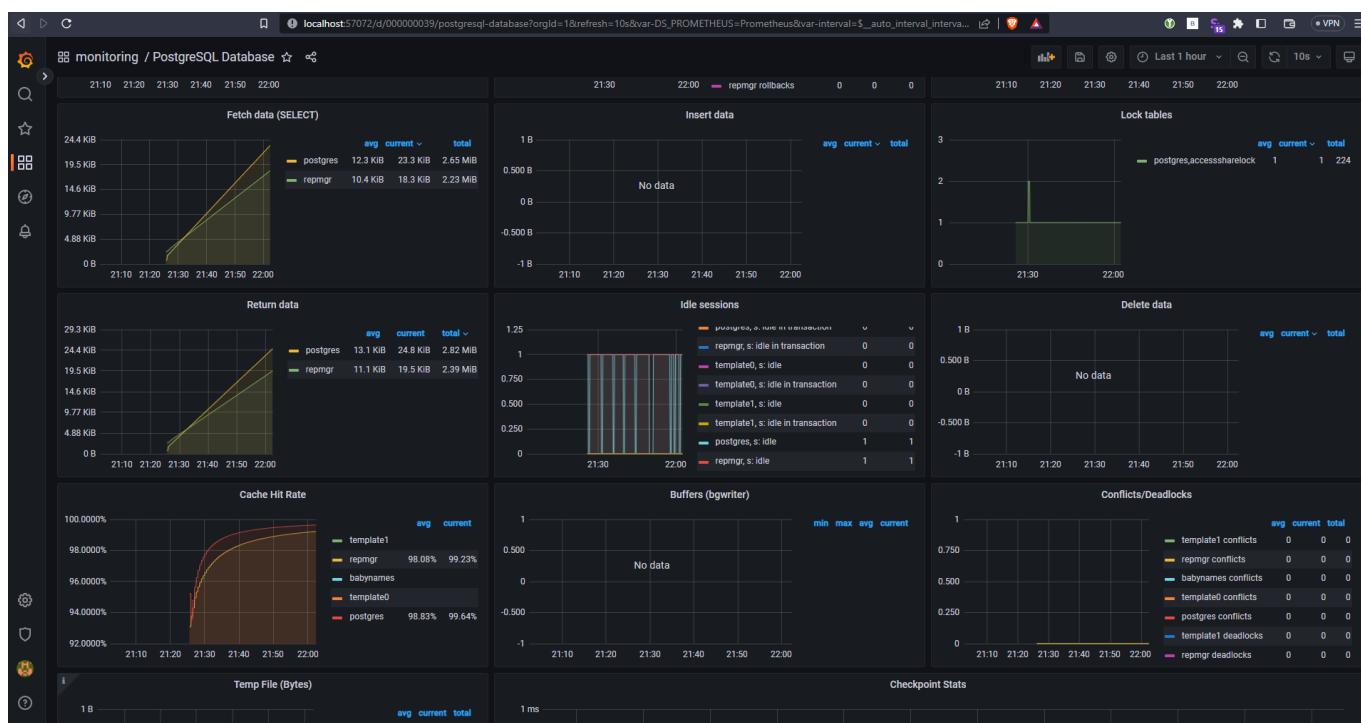
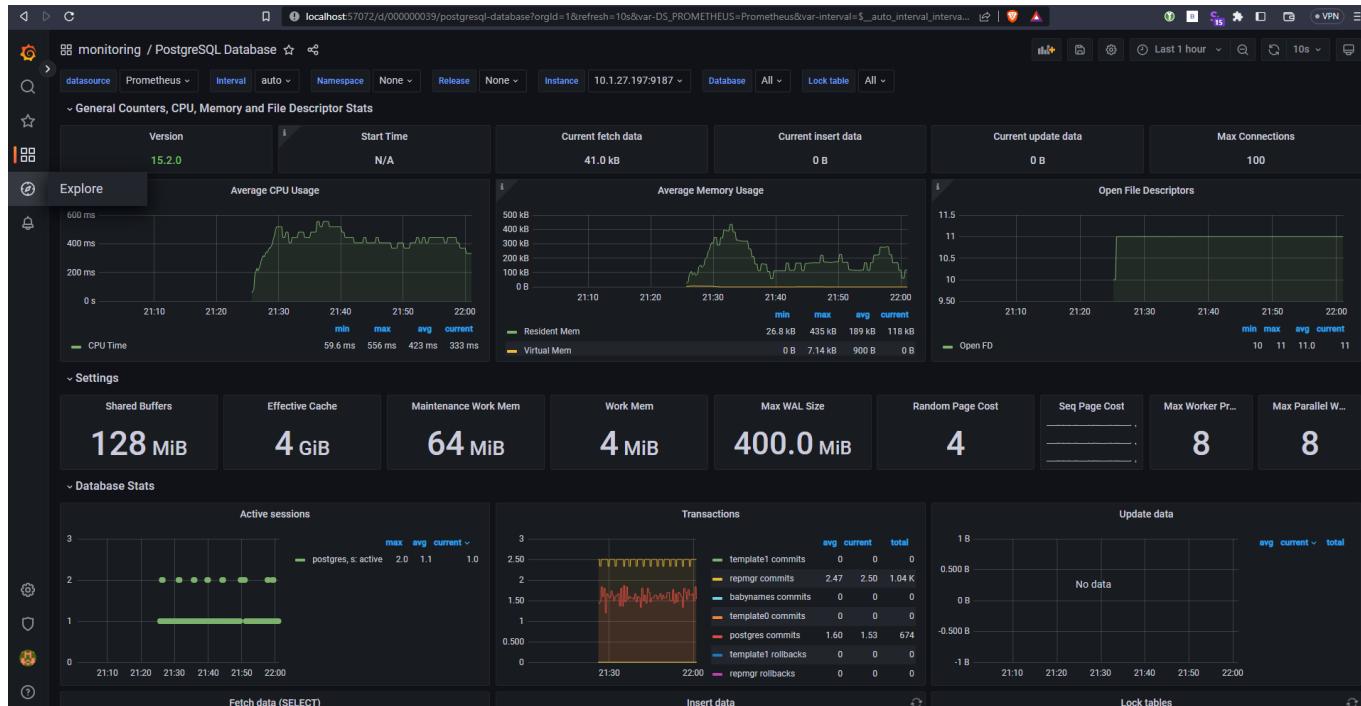


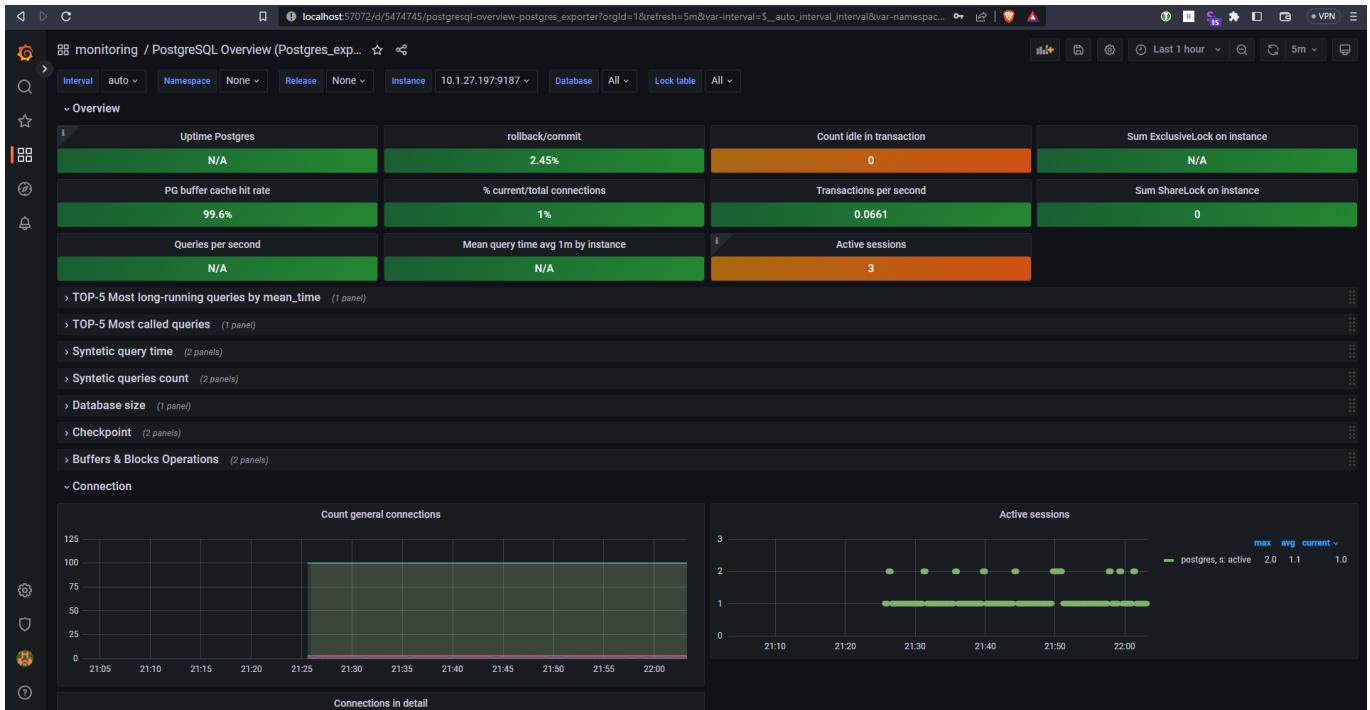
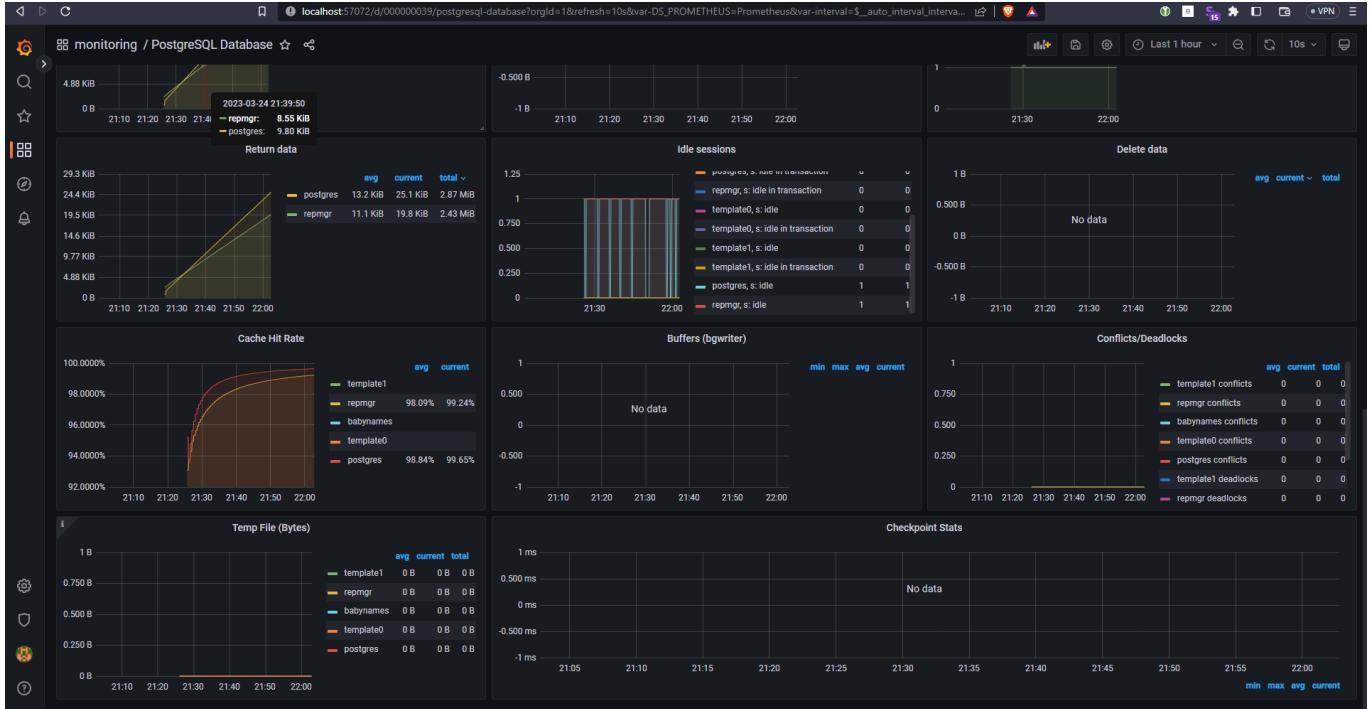


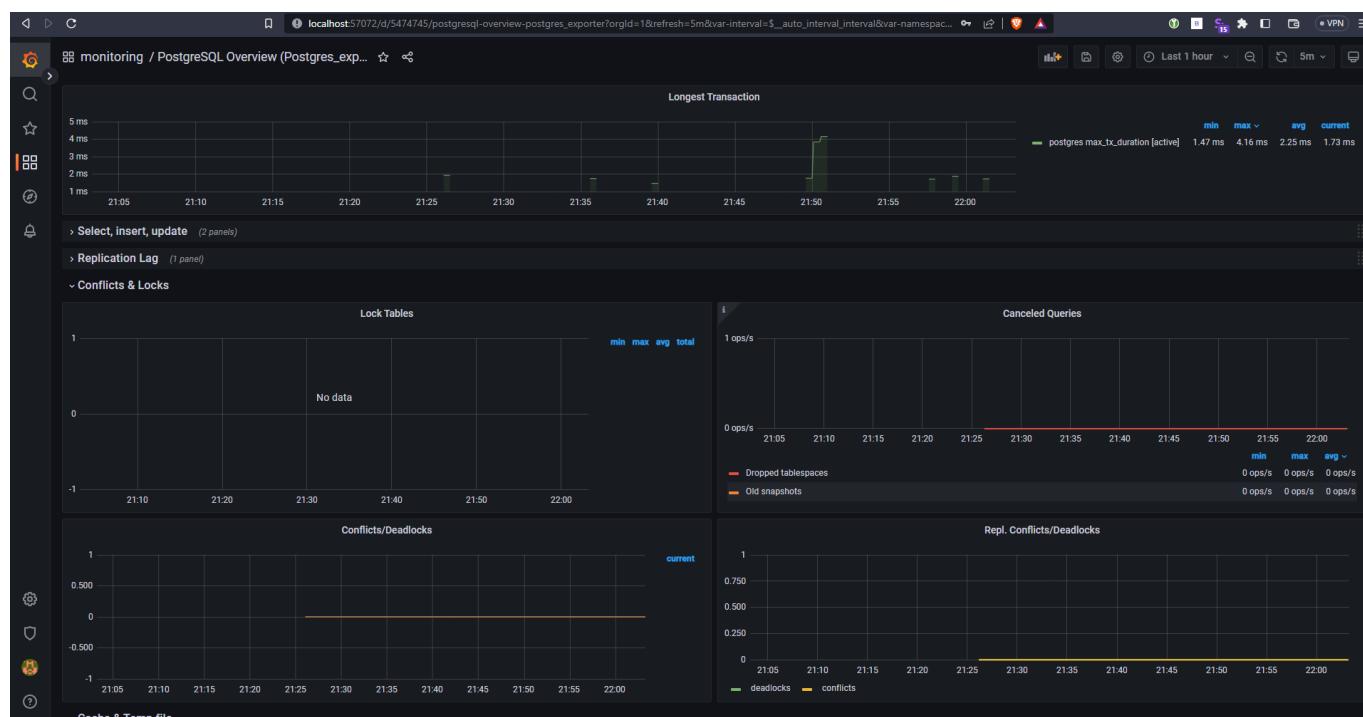
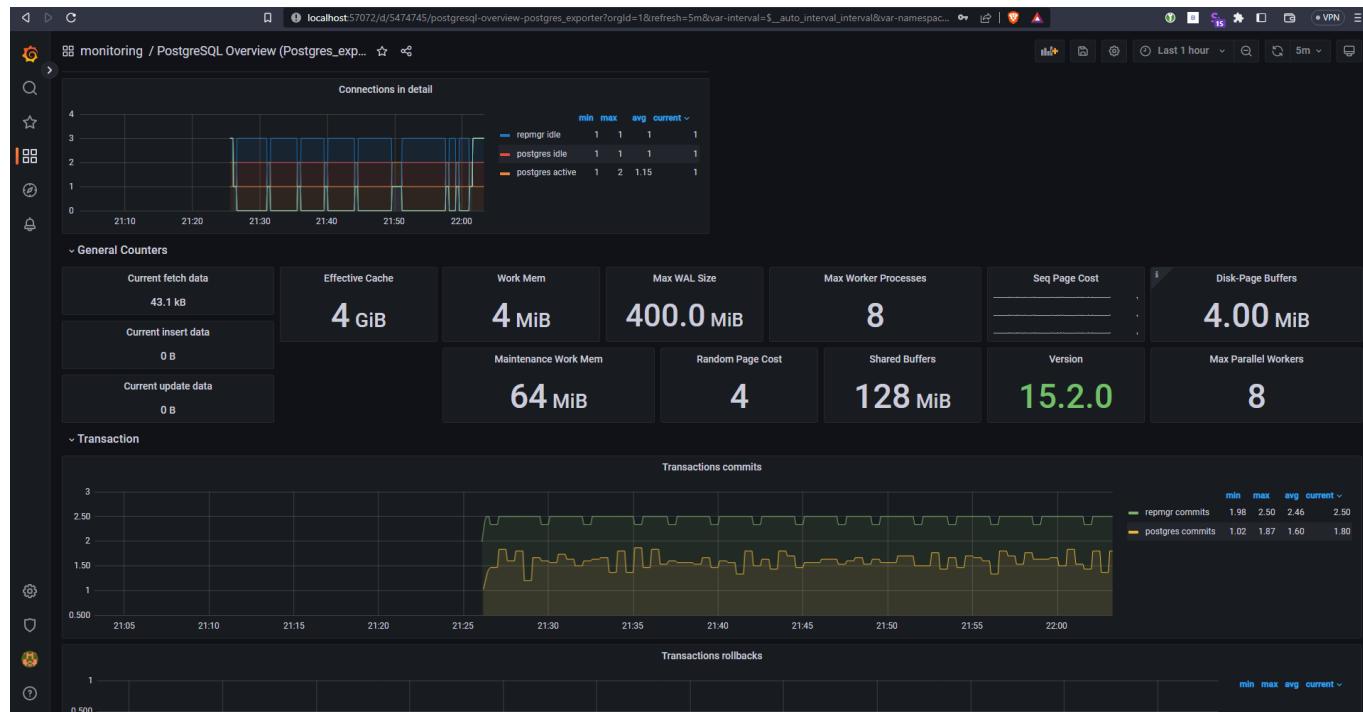


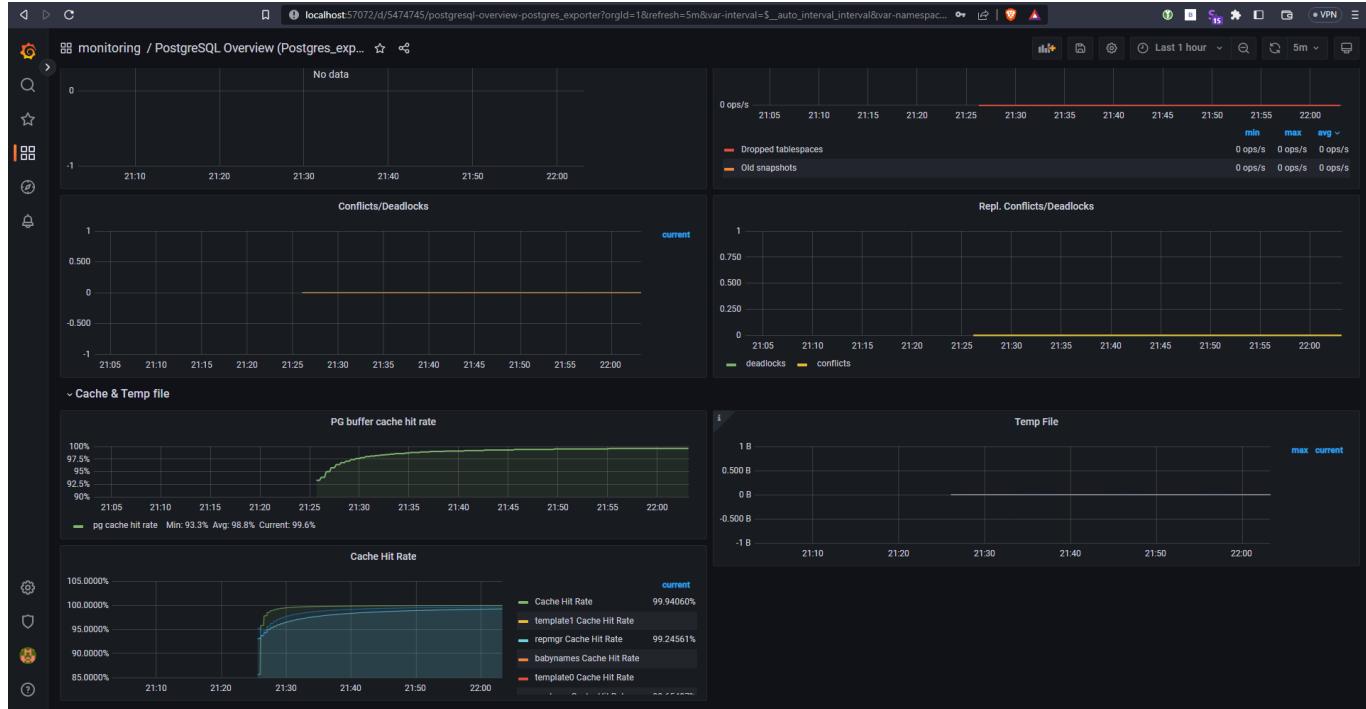
## 4- PostGre HA

- Pruebas de creación, borrado, actualización y búsqueda



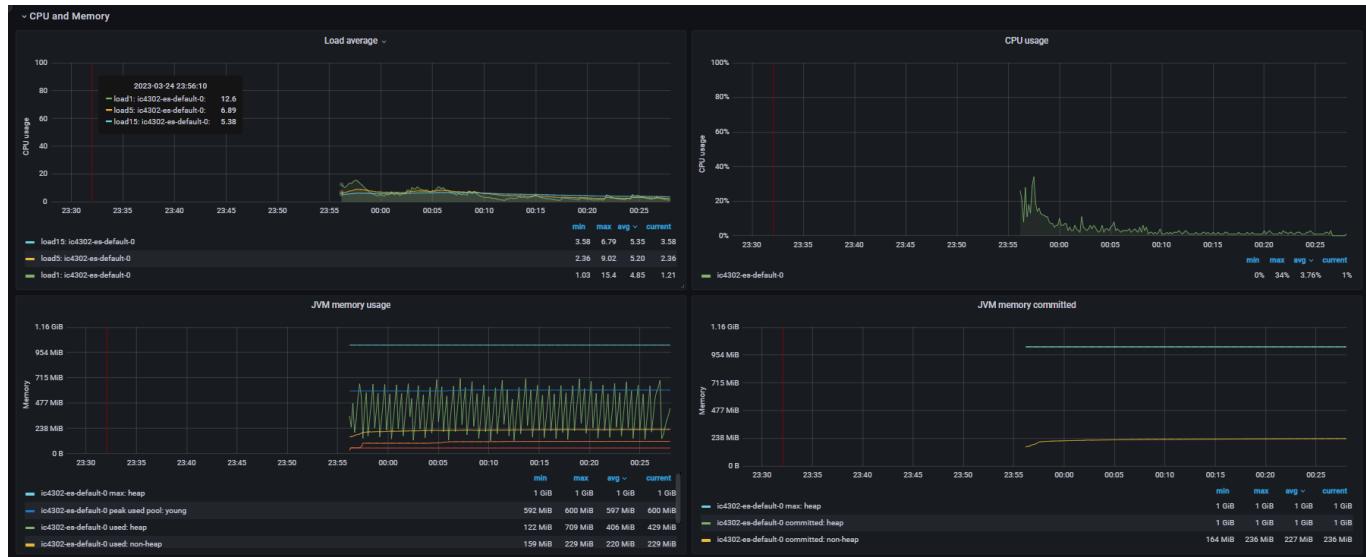


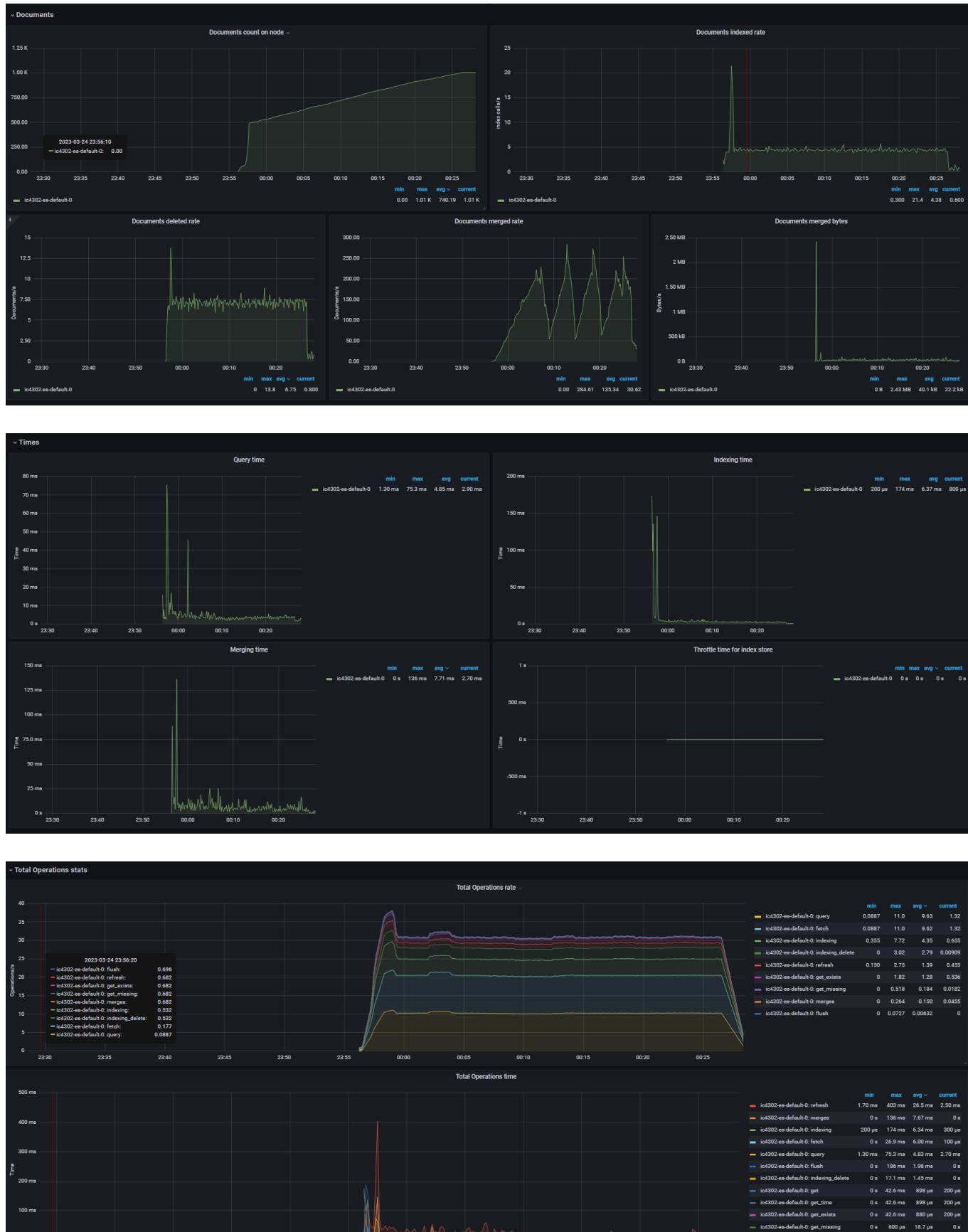




## 5- Elasticsearch

- Pruebas de creación, borrado, actualización y búsqueda con 5 usuarios por segundo para inserciones, 10 usuarios por segundo para selects, 2 usuarios por segundo para updates y 5 usuarios por segundo para deletes. Estas configuraciones son diferentes a las bases de datos anteriores debido a que si se hacía la prueba de carga con más usuarios se caía Elasticsearch.







## Conclusiones de los resultados de las pruebas de carga

El objetivo para las pruebas de carga era simular el comportamiento real de una base de datos, por lo que se intentó mantener un flujo constante de usuarios y consultas. En las gráficas se puede observar que no hubieron cambios radicales entre las pruebas.

En la vida real los usuarios no se comportarán de una forma constante, por lo que el resultado de estas pruebas no es realista. Sin embargo, ayudan a reflejar problemas que pueden surgir en diferentes escenarios. Entre algunos problemas que se pueden identificar al realizar estas pruebas de carga son cuellos de botella.

En el caso de Elasticsearch, durante la ejecución de pruebas tuvimos problemas al cargar la cantidad de usuarios que se utilizó para las pruebas de las demás bases de datos debido a que la prueba de carga generó errores ya que Elastic no soportó la cantidad de consultas. Por esta razón nos vimos obligados a disminuir la cantidad de usuarios considerablemente para asegurar una prueba de carga de al menos 30 minutos.

Antes de la ejecución de las pruebas de carga el uso de disco, memoria y CPU son constantes puesto que no se está ejecutando nada. Cuando se comienza la inyección de datos, el uso comienza a aumentar.

En el caso de MariaDB, se puede observar en el reporte de Gatling que el tiempo de respuesta fue menor en MariaDB Galera. Esto tiene sentido, puesto que MariaDB Galera es una versión High Availability de MariaDB.

En el caso de PostgreSQL, tanto PostgreSQL como PostgreSQL High Availability dieron un buen tiempo de respuesta en el reporte de Gatling. Por otro lado, en el monitoreo en Grafana el uso de CPU y de memoria es mayor en PostgreSQL High Availability.

En el caso de Elasticsearch se puede observar en las gráficas que la cantidad de documentos del nodo aumentó una vez que se ejecutaron las pruebas de carga.

## Conclusiones

---

- 1-** Es importante la comunicación entre los miembros de grupo de trabajo.
- 2-** Se debe mantener la organización para poder realizar la tarea.
- 3-** Entender los conceptos vistos en clase ayuda en la realización de la tarea.
- 4-** El tener un buen control de versiones y saber utilizar github facilita el trabajo en equipo.
- 5-** Se deben aplicar buenas prácticas de programación para mantener el orden.
- 6-** Mantener la estructura del proyecto es esencial.
- 7-** Se debe asegurar un código legible y entendible.
- 8-** Docker es una herramienta muy útil para nosotros como desarrolladores, pues nos permite ejecutar una aplicación de manera consistente en cualquier entorno, facilitando en muchos casos el desarrollo del software y aún más importante, permitiendo la portabilidad y escalabilidad de la aplicación.
- 9-** Grafana es una herramienta de visualización y monitoreo que es muy utilizada en una amplia gama de casos y entornos de trabajo, ya que por medio de paneles de visualización nos brinda una visualización clara de los sistemas y aplicaciones, esto nos ayuda a nosotros como usuarios, debido a que nos permite ver el rendimiento de la aplicación o sistema, identificar problemas, reconocer irregularidades y tomar medidas correctivas rápidamente.
- 10-** Similar a Grafana, Gatling es una herramienta de mucha utilidad principalmente para ver el rendimiento de un sistema e identificar posibles errores que podrían presentarse en diferentes entornos antes de que el sistema sea lanzado o se implemente en una empresa. Con esta somos capaces de realizar pruebas de carga y estrés en el sistema para identificar posibles cuellos de botella en el rendimiento simulando grandes cargas de trabajo. Lo anterior es fundamental para nosotros, pues nos permite asegurar el buen rendimiento y funcionamiento del sistema o aplicación que estemos desarrollando.

## Recomendaciones

---

- 1-** Hacer reuniones periódicas para ver el avance de la tarea.
- 2-** Mantener la organización de la tarea, según el ejemplo del profesor.
- 3-** Repasar los conceptos vistos en clase y complementar con investigación.
- 4-** Aprender a hacer uso de github.

- 5-** Seguir un estándar de código.
- 6-** Seguir aprendiendo y enriqueciendo el conocimiento después de finalizar la tarea.
- 7-** Investigar sobre las diferentes herramientas esenciales para desarrollar la solución e ir tomando apuntes sobre los aspectos importantes de cada uno de estas.
- 8-** Tener una buena estructura del proyecto y dividir el proyecto de forma funcional.
- 9-** Repartir y asignar tareas a cada integrante del equipo para progresar.
- 10-** Definir roles en el equipo de trabajo para mantener el orden.