Modernización de bases de datos MySQL de alta demanda con Vitess

Vitess

Índice

1. Objetivos del proyecto.

Los objetivos que se pretenden alcanzar en este proyecto son los siguientes:

- Introducir Vitess, describiendo sus principales características, su utilidad en entornos de alta demanda y el procedimiento necesario para su instalación y ejecución.
- Ver las diferencias de gestionar bases de datos MySQL con Vitess y sin él.
- Explicar el funcionamiento interno de Vitess y observar en tiempo real algunas de sus funciones clave.
- Instalar Vitess en una máquina virtual y seguir los ejemplos que nos da Vitess para probar su funcionamiento con una base de datos. Aprender a mover tablas, hacer sharding y crear y restaurar copias de seguridad de shards.

He podido cumplir con todos los objetivos del proyecto.

2. Fundamentos teóricos y conceptos.

2.1 ¿Qué es Vitess?

Vitess es una solución para bases de datos MySQL, es una plataforma de bases de datos distribuidas, que permite implementar, escalar y administrar grandes clústeres de instancias de bases de datos MySQL. Combina y amplía muchas características importantes de SQL y MySQL con la escalabilidad de una base de datos NoSQL.

Vitess pretende ayudar con los siguientes problemas:

- Escalar bases de datos MySQL permitiendo particionarla (sharding) mientras mantiene los cambios de la aplicación al mínimo. A simple vista y desde el exterior puede parecer una única base de datos MySQL pero por dentro puede estar compuesta por múltiples instancias de MySQL (shards).
- Migrar desde servidores físicos o máquinas virtuales a una nube privada o pública.
- Desplegar y gestionar un gran número de instancias de bases de datos MySQL.

En resumen, Vitess convierte múltiples servidores MySQL en una base de datos distribuida, escalable y altamente disponible, sin requerir grandes cambios en las aplicaciones existentes.

2.2 Funciones clave de Vitess

- Sharding automático o manual.
- Balanceo de carga entre réplicas de lectura. Permite miles de conexiones simultáneas entre la aplicación y la base de datos.
- Failover automático. Es capaz de reemplazar nodos caídos de forma automática o manual.
- Compatible con Kubernetes (puede ejecutarse como contenedores).
- Protección contra consultas peligrosas o pesadas. Reescribe las consultas y las sanea antes de ejecutarlas según límites configurables y permite bloquear y cancelar consultas largas o problemáticas.
- Monitoreo del rendimiento de la base de datos.
- Interfaz gráfica web para la administración de la base de datos.

2.3 Vitess vs MySQL tradicional

Diferencias entre Vitess y una base de datos MySQL tradicional.

MySQL tradicional	Vitess
Cada conexión consume bastante memoria y CPU.	Usa conexiones ligeras y puede manejar miles con eficiencia gracias a Go.
Consultas mal hechas o sin optimizar afectan el rendimiento general.	Emplea un analizador SQL que utiliza un conjunto configurable de reglas para reescribir consultas que podrían perjudicar el rendimiento de la base de datos.
No tiene soporte nativo para sharding	Permite fragmentar y migrar datos fácilmente entre servidores.
Requiere gestión manual de réplicas y failover.	Admite y maneja automáticamente varios escenarios, incluida la detección y recuperación de errores primarios. También tiene soporte para copias de seguridad y restauraciones de datos.

2.4 Vitess en empresas

El uso de Vitess en las empresas ha aumentado en los últimos años, especialmente en empresas que buscan bases de datos altamente escalables, con alta disponibilidad y capaces de manejar grandes volúmenes de tráfico. Vitess permite a estas empresas superar muchas de las limitaciones de MySQL tradicional sin tener que abandonar su ecosistema existente, convirtiéndose en una solución estratégica ante los problemas que presenta MySQL tradicional.

Más de 30 compañías tecnológicas famosas utilizan actualmente Vitess, entre ellas están:

- **YouTube**. Fue la empresa que desarrolló e implementó Vitess para solucionar problemas de escalabilidad y rendimiento en su backend basado en MySQL. Esta es su historia.
- **Slack**. Utiliza Vitess para escalar su infraestructura de bases de datos, especialmente en lo relacionado con la mensajería en tiempo real.
- **GitHub**. Migró parte de su infraestructura a Vitess para facilitar el particionado horizontal y mejorar la resiliencia de su base de datos.
- **Pinterest, Vinted, Etsy, Shopify, etc.**

2.5 Compatibilidad de Vitess

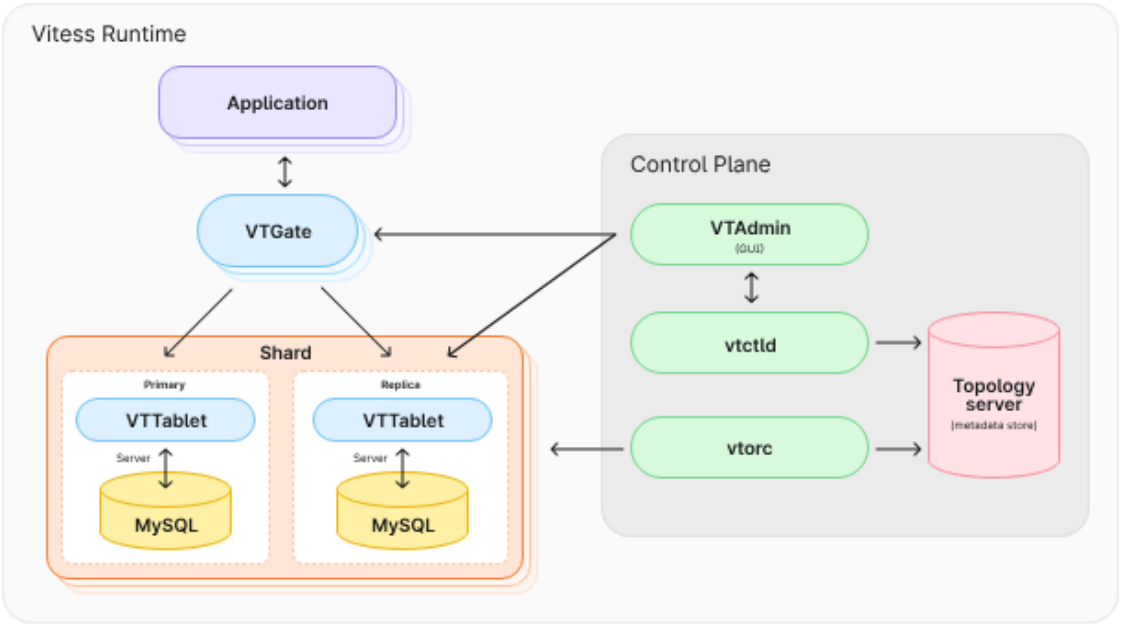
Vitess es compatible con:

- **Linux.** Vitess está desarrollado y probado principalmente en Linux y es completamente compatible y recomendado para la producción. Funciona muy bien en Ubuntu, Debian, Rocky Linux, CentOS, Fedora y Alma Linux.
- **Docker.** Perfecto para desarrollo y pruebas.
- **Kubernetes.** Es el entorno más recomendado para producción.

Vitess es compatible con bases de datos MySQL 8.0 y Percona server y es capaz de poder importar bases de datos desde MariaDB.

2.6 Funcionamiento interno de Vitess

Veamos el funcionamiento interno de Vitess, analizando sus componentes principales, partiendo de este diagrama:



- **Keyspace**

Un keyspace es una base de datos lógica. A simple vista desde la aplicación un keyspace aparece como una única base de datos, sin embargo este keyspace puede estar formado por distintos shards, respaldados por varias instancias de MySQL que contienen sus datos. Estos datos se fragmentan según una clave definida. Esto permite distribuir los datos horizontalmente y escalar la base de datos sin que la aplicación se entere de nada del sharding.

- **Shard**

Un shard es una partición física de los datos dentro de un keyspace. Cada shard suele contener un tablet primario y otro o varios tablets réplicas. Vitess decide a qué shard enviar cada consulta en función de la clave de fragmentación.

- **Tablet**

Un tablet es una combinación de un proceso mysqld (que gestiona el almacenamiento y la ejecución de consultas de datos) y un proceso VTTablet, que actúa como intermediario y permite a Vitess comunicarse con la base de datos (con el proceso mysqld). En cada shard suele haber un tablet primario, un tablet réplica y un tablet de sólo lectura.

Estos tablets tienen distintos modos de uso :

- **primary**. Un tablet que actúa como el equivalente primario de MySQL para su shard. Donde se escriben los datos.
- **replica**. Una réplica de MySQL que puede ser promovida a primary. Se utilizan para atender solicitudes en vivo orientadas al usuario, como el frontend de una web.
- **rdonly**. Una réplica de MySQL que no puede ser promovida a primary. Normalmente se usa para tareas de procesamiento en segundo plano, como copias de seguridad, exportación de datos, etc.
- **backup**. Un tablet que ha detenido la replicación en un punto consistente, con el fin de subir una nueva copia de seguridad para su shard. Al finalizar, reanuda la replicación y vuelve a su tipo anterior.
- **restore**. Un tablet que se ha iniciado sin datos y está en proceso de restaurarse a partir de la última copia de seguridad. Una vez finalizado, comenzará a replicar desde la copia de seguridad y pasará a ser un tablet *replica* o *rdonly*.
- **drained**. Un tablet que ha sido reservado por un proceso en segundo plano de Vitess.

Las consultas se enrutan al tablet más oportuno mediante VTGate.

- **VTGate**

Es el servidor proxy de Vitess. Recibe las consultas de las aplicaciones y se encarga de enrutar, balancear y reescribir las consultas SQL hacia los shards y tablets más adecuados.

- **VTAdmin**

Es la interfaz web de administración de Vitess. Permite visualizar la topología del clúster, ver el estado de los shards, keyspaces, tablets y realizar acciones como migraciones, cambios de rol, backups, etc.

- **vtctl**

Es el sistema de control de Vitess utilizado para administrar un clúster de Vitess por línea de comandos. Permite entre otras cosas: Inicializar keyspaces y shards, promover tablets, migrar esquemas o hacer backups y restauraciones. Permite el uso de scripts para la automatización.

- **vtctld**

Es un servidor HTTP que permite examinar la información almacenada en el topology server.

- **Topology service**

Es un almacén de datos distribuido que mantiene el estado y la configuración del clúster. Vitess puede usar diferentes servicios para esto, como **etcd** o ZooKeeper. Guarda la información de los keyspaces, shards, tablets y más. Todos los componentes consultan este servicio para tener una visión coherente del sistema.

- **vtorc**

Es el sistema para la detección y reparación automática de fallos en Vitess. Monitorea continuamente los tablets primarios y réplicas. Si detecta un fallo en un tablet primario, promueve una réplica y actualiza la topología para garantizar la disponibilidad del sistema. Es clave para la alta disponibilidad.

3. Probando Vitess

3.1 Instalación de Vitess

Los siguientes comandos los estaré ejecutando en una máquina virtual Debian 12 llamada "IvanR-Vitess1" con 8GB de RAM. Voy a instalar la versión 22 de Vitess.

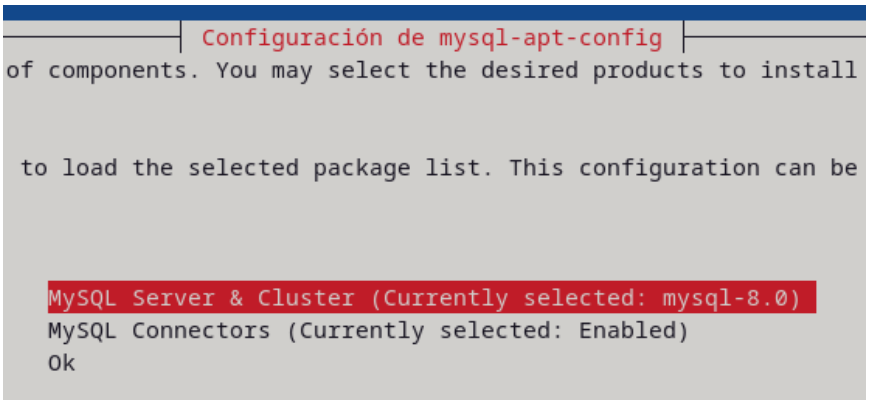
Instalamos etcd y git.

```
$ sudo apt install -y etcd-server etcd-client curl git
```

Instalamos MySQL 8.0.

```
$ wget https://repo.mysql.com/mysql-apt-config_0.8.32-1_all.deb
$ sudo dpkg -i mysql-apt-config_0.8.32-1_all.deb
$ sudo apt update
$ sudo apt install mysql-server
```

Durante la ejecución del comando dpkg anterior, nos asegurarnos que en MySQL Server & Cluster hemos elegido mysql-8.0.



Una vez instalados etcd y MySQL, paramos sus servicios para evitar conflictos con Vitess, que se encargará de iniciarlos cuando los necesite.

```
$ sudo service mysql stop
$ sudo service etcd stop
```

```
$ sudo systemctl disable mysql
$ sudo systemctl disable etcd
```

Instalamos Node.

```
$ curl -o-
https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.5/install.
sh | bash

$ nvm install 18
$ nvm use 18
```

Instalamos Vitess.

```
$ version="v22.0"
$ url="$(curl -s
https://api.github.com/repos/vitessio/vitess/releases | jq
--arg version "${version}" -r '[.[] | select(.tag_name |
contains($version))] | sort_by(.created_at) | reverse |
.[0:1] | .[] | .assets[] | select(.content_type |
contains("application/gzip")) | .browser_download_url')"
$ file="${url##*/}"
$ curl -LO "${url}"


$ tar -xzf ${file}
$ cd ${file/.tar.gz/}
$ sudo mkdir -p /usr/local/vitess
$ sudo cp -r * /usr/local/vitess/
```

Añadimos esta línea a .bashrc.

```
export PATH=/usr/local/vitess/bin:${PATH}

$ source ~/.bashrc
```

Con esto ya estaría instalado Vitess.

3.2 Ejecución de cluster de ejemplo en Vitess

Vamos a ejecutar el cluster de ejemplo que encontramos en los ejemplos de Vitess. En primer lugar, copiaremos la carpeta de Vitess para poder hacer pruebas y cambios.

```
$ vitess_path=/usr/local/vitess
$ mkdir ~/Ivanr-vitess
$ cp -r ${vitess_path}/{examples,web} ~/Ivanr-vitess
$ cd ~/Ivanr-vitess/examples/local
```

Tras esto ejecutamos este script que analizaremos a continuación:

```
debian@IvanR-Vitess1:~/Ivanr-vitess/examples/local$
./101_initial_cluster.sh
Starting etcd...
add zone1 CellInfo
Created cell: zone1
etcd is running!
Starting vtctld...
Curling "http://192.168.1.176:15000/debug/status" to check if
vtctld is up
vtctld is running!
Successfully created keyspace commerce. Result:
{
  "name": "commerce",
  "keyspace": {
    "keyspace_type": "NORMAL",
    "base_keyspace": "",
    "snapshot_time": null,
    "durability_policy": "semi_sync",
    "throttler_config": null,
    "sidecar_db_name": "_vt"
  }
}
Starting MySQL for tablet zone1-0000000102...
Starting MySQL for tablet zone1-0000000100...
Starting MySQL for tablet zone1-0000000101...
Waiting for mysqlctls to start...
MySQL for tablet zone1-0000000100 is running!
MySQL for tablet zone1-0000000102 is running!
MySQL for tablet zone1-0000000101 is running!
mysqlctls are running!
```

```
Starting vttablet for zone1-0000000100...
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

vttablet for zone1-0000000100 is running!
Starting vttablet for zone1-0000000101...
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

vttablet for zone1-0000000101 is running!
Starting vttablet for zone1-0000000102...
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

vttablet for zone1-0000000102 is running!
Starting vtorc...
vtorc is running!
  - UI: http://localhost:16000
  - Logs:
/home/debian/Ivanr-vitess/examples/local/vtdataroot/tmp/vtorc
.out
  - PID: 5661


New VSchema object:
{
  "sharded": false,
  "vindexes": {},
  "tables": {
    "corder": {
      "type": "",
      "column_vindexes": [],
      "auto_increment": null,
      "columns": [],
      "pinned": "",
      "column_list_authoritative": false,
      "source": ""
    },
    "customer": {
      "type": "",
      "column_vindexes": [],
      "auto_increment": null,
      "columns": [],
```

```
      "pinned": "",
      "column_list_authoritative": false,
      "source": ""
    },
    "product": {
      "type": "",
      "column_vindexes": [],
      "auto_increment": null,
      "columns": [],
      "pinned": "",
      "column_list_authoritative": false,
      "source": ""
    }
  },
  "require_explicit_routing": false,
  "foreign_key_mode": "unspecified",
  "multi_tenant_spec": null
}
If this is not what you expected, check the input data (as
JSON parsing will skip unexpected fields).
Starting vtgate...
vtgate is up!
Access vtgate at http://192.168.1.176:15001/debug/status

vtadmin-api expects vtadmin-web at, and set http-origin to
"http://192.168.1.176:14201"
vtadmin-api is running!
  - API: http://192.168.1.176:14200
  - Logs:
/home/debian/Ivanr-vitess/examples/local/vtdataroot/tmp/vtadm
in-api.out
  - PID: 5927

Building vtadmin-web...

Installing nvm...

nvm is already installed!

Configuring Node.js 22.13.1

v22.13.1 is already installed.
Now using node v22.13.1 (npm v10.9.2)
```

```
Setting VITE_VTADMIN_API_ADDRESS to
"http://192.168.1.176:14200"

vtadmin-web is running!
  - Browser: http://192.168.1.176:14201
  - Logs:
/home/debian/Ivanr-vitess/examples/local/vtdataroot/tmp/vtadm
in-web.out
  - PID: 6396
```

1. Inicia el servidor de topología, que en este caso estamos usando etcd. Tras esto crea en etcd la celda “zone1”. Vitess organiza los datos por regiones o zonas llamadas.
2. Inicia vtctld, el servidor HTTP que nos permite ver la información del topo server. Nos muestra la URL para poder acceder a su interfaz web.
3. Crea el keyspace commerce en etcd.
4. Crea tres instancias de mysql, una para cada tablet del ejemplo.
5. Crea un vttablet para cada instancia de mysql y comprueba si tienen conexión.
6. Inicia vtorc, el sistema para la detección y reparación automática de fallos en Vitess y orquestador de réplicas MySQL. Nos da la URL para la UI web.
7. Crea el VSchema para el keyspace commerce, que es un archivo JSON que le indica a Vitess las tablas que tiene el keyspace y como está configurado el sharding para ese keyspace.
8. Inicia vtgate, el proxy de Vitess para las conexión con MySQL.
9. Inicia vtadmin-api, que es la API para vtadmin-web.
10. Inicia vtadmin-web, que es el dashboard web de Vitess que nos permite administrar Vitess de forma gráfica. Nos da la URL para acceder.

Podemos verificar que los procesos de etcd, MySQL y Vitess están ejecutandose.

```
debian@IvanR-Vitess1:~/Ivanr-vitess/examples/local$ pgrep -fl vitess
3742 etcd
3772 vtctld
4004 mysqld_safe
4045 mysqld_safe
4156 mysqld_safe
5355 mysqld
5407 mysqld
5440 mysqld
5574 vttablet
5607 vttablet
5634 vttablet
5661 vtorc
5891 vtgate
```

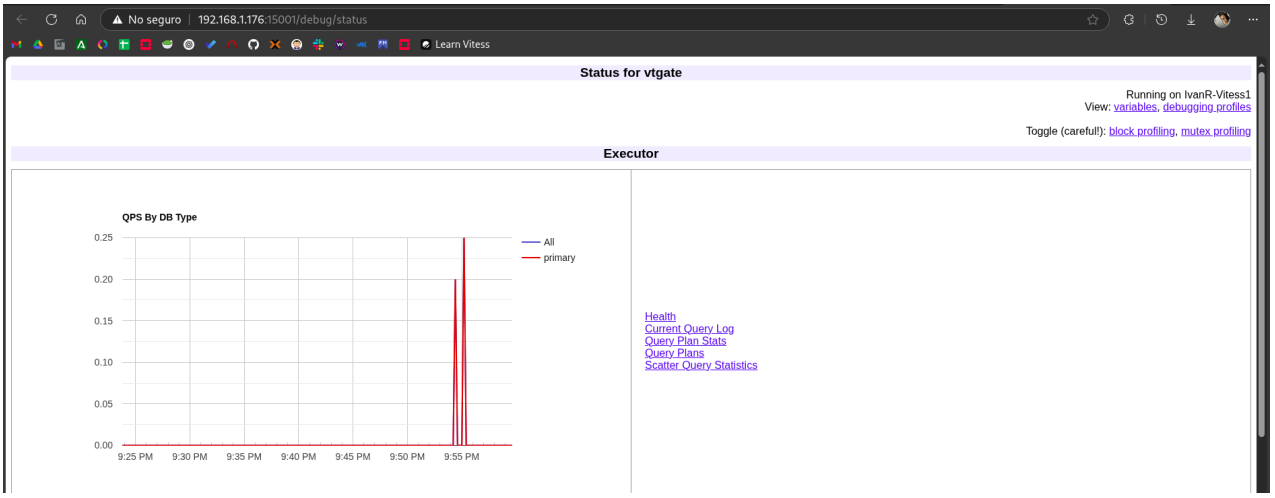
El cluster de ejemplo se ha creado y ejecutado correctamente. Veamos ahora algunos aspectos de este cluster. Para facilitar el uso de Vitess podemos cargar las variables de entorno de Vitess.

```
$ source ../common/env.sh
```

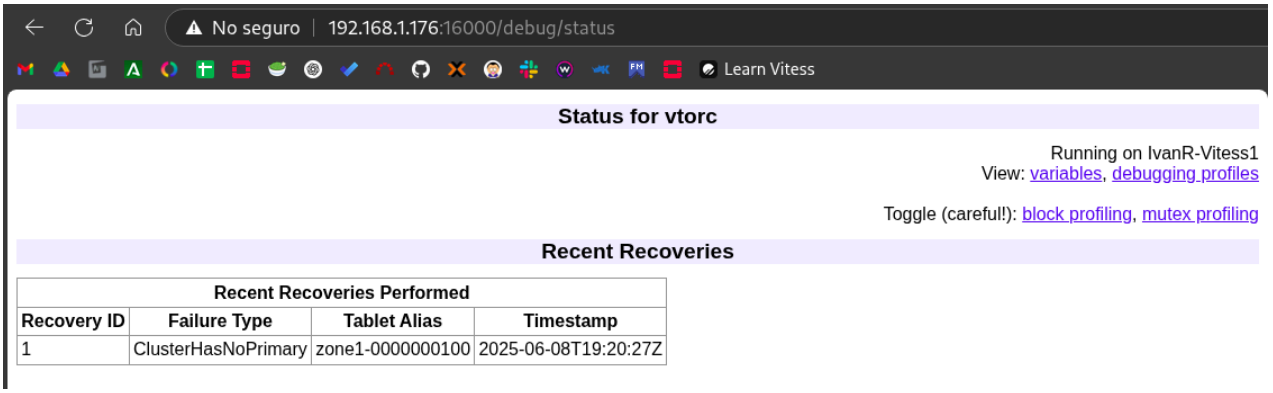
Podemos conectarnos a MySQL y comprobar que se han creado las tablas del keyspace commerce, que son pedidos, clientes y productos.

```
$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 8.0.40-Vitess Version: 22.0.0
Copyright (c) 2000, 2025, Oracle and/or its affiliates.

mysql> show tables;
+--------------------+
| Tables_in_commerce |
+--------------------+
| corder             |
| customer           |
| product            |
+--------------------+
3 rows in set (0,01 sec)
```

Podemos acceder a la interfaz web de VTGate. Vemos en tiempo real el estado del servidor proxy y las consultas que se hacen.

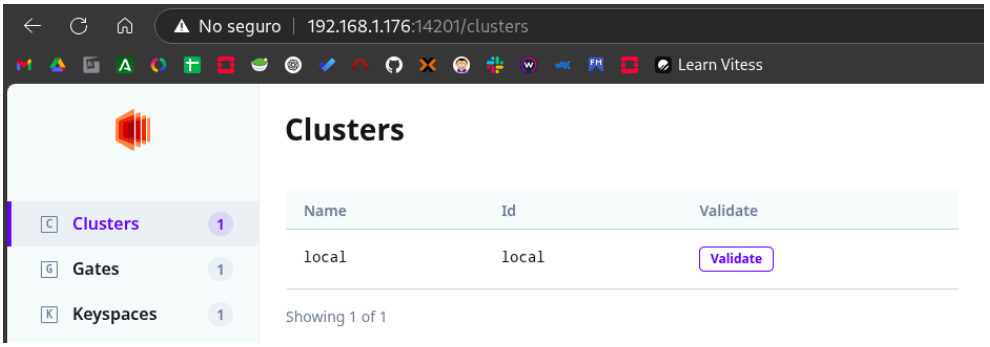


Podemos acceder a la interfaz web de VTOrc. Muestra errores que detecta.

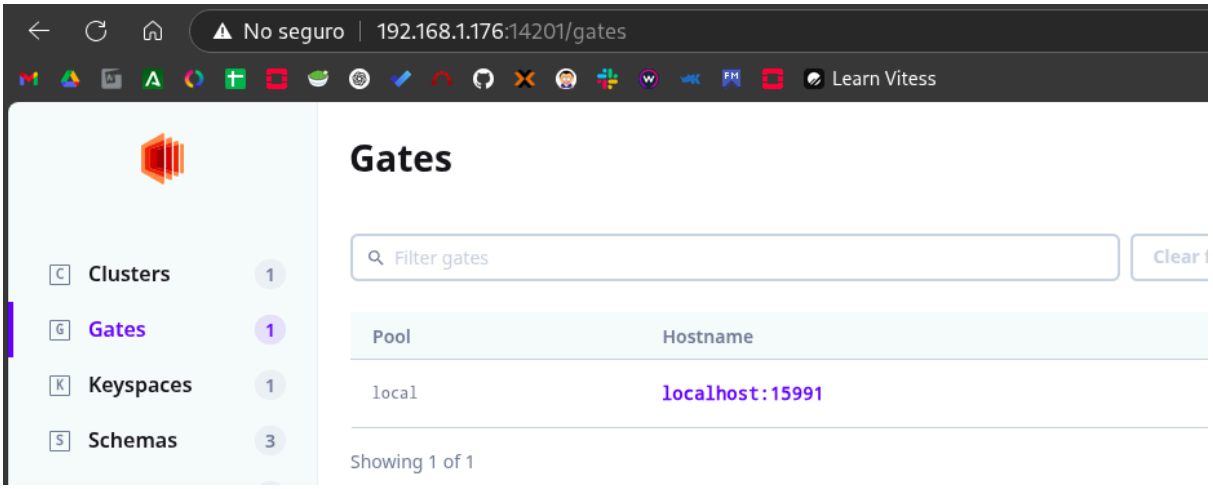


3.3 Interfaz web VTAdmin.

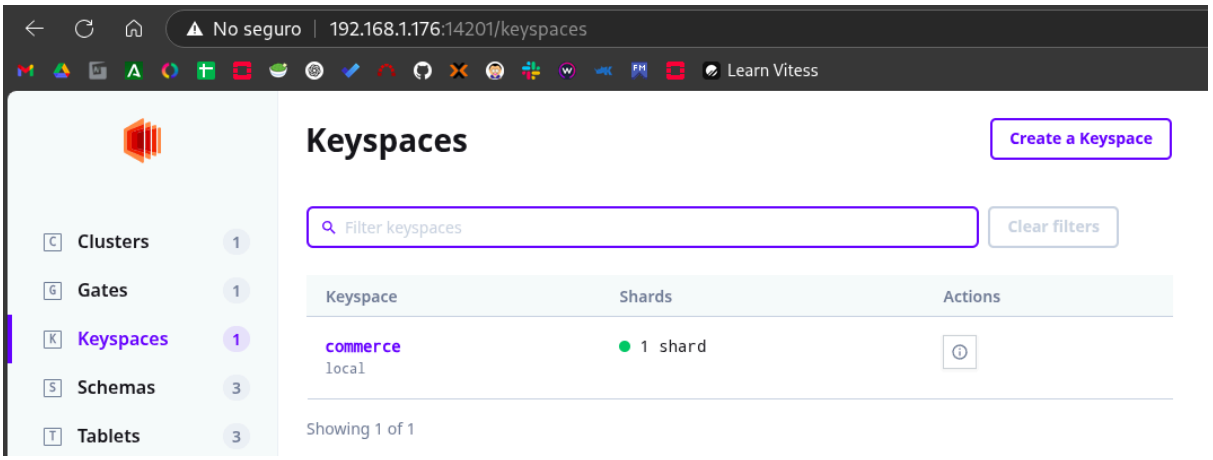
Podemos acceder a la interfaz web de VTAdmin. Veamos que nos ofrece.



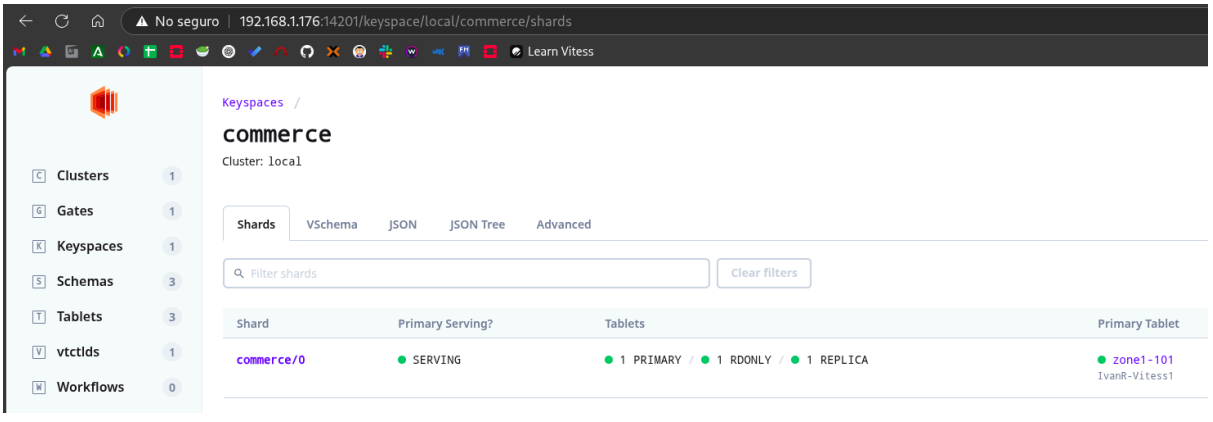
En **Clusters** podemos ver los clusters del servidor. El actual se llama local y nos permite validar que todos los nodos del cluster son accesibles y están activos.

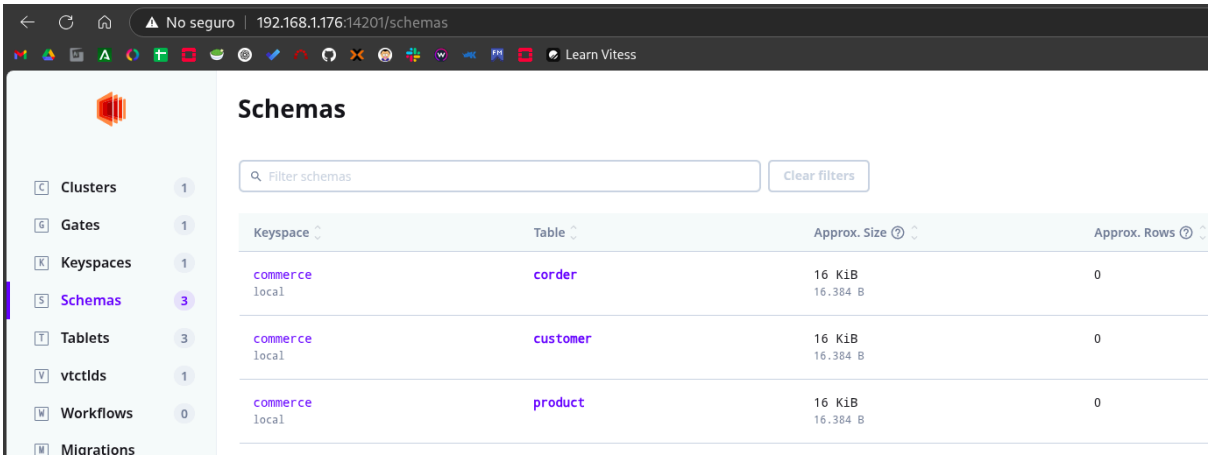


En **Gates** podemos ver los servidores VTGate disponibles, el cluster al que pertenecen y el hostname para acceder.

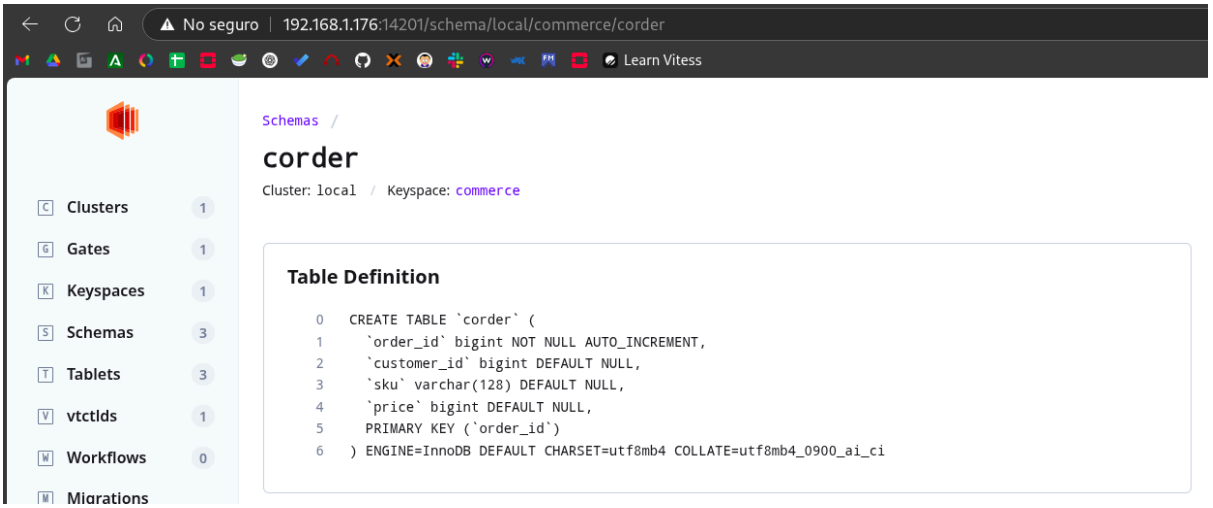


En **Keyspaces** podemos ver los keyspace del servidor, poder crearlos, validarlos y ver sus shards. Si clicamos en el keyspace nos enseñará la lista de shards, como está configurado el keyspace y algunas opciones avanzadas.

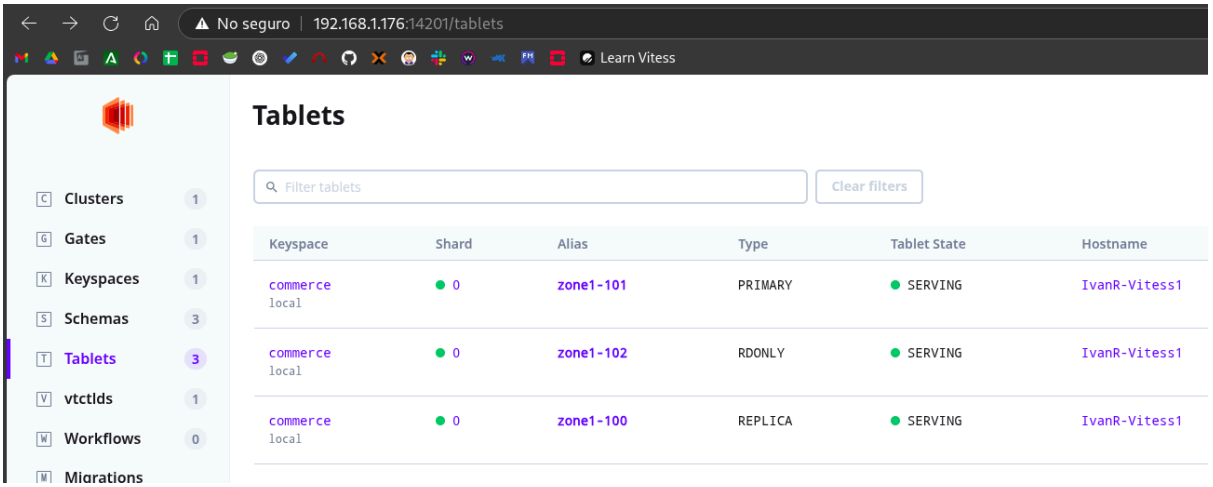




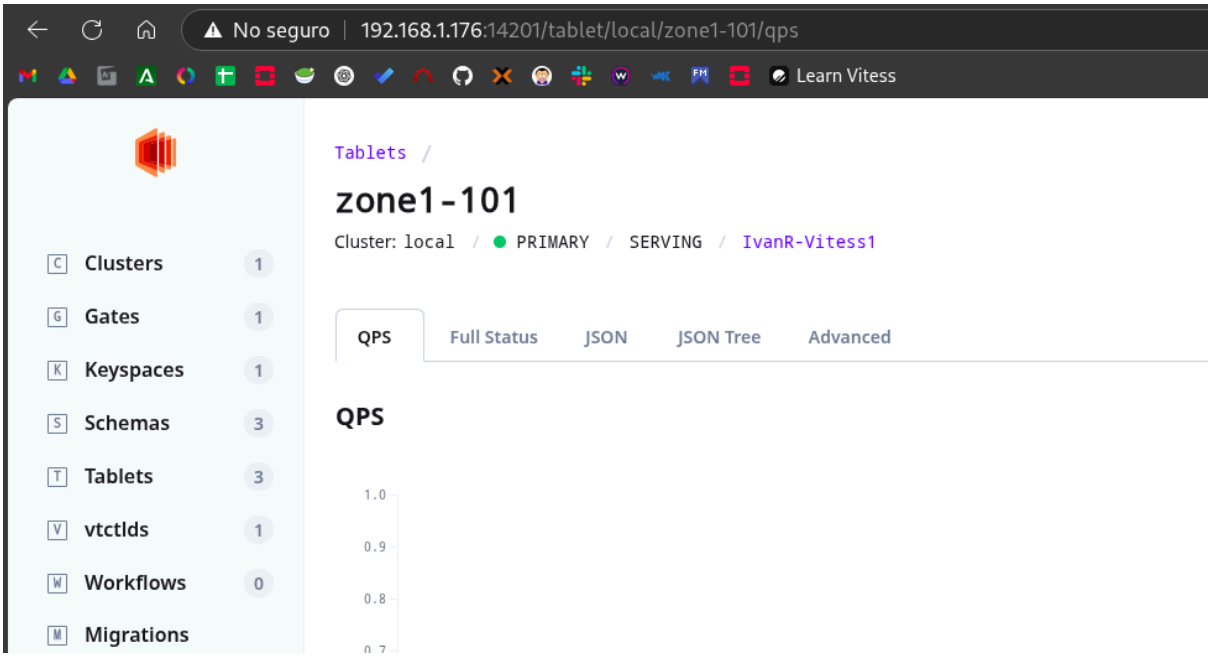
En **Schemas** vemos los esquemas que forman los keyspaces. En este caso vemos que hay tres esquemas que corresponden a las tres tablas que creamos. Vemos el keyspace al que pertenecen, el tamaño aproximado y el número de filas aproximado.



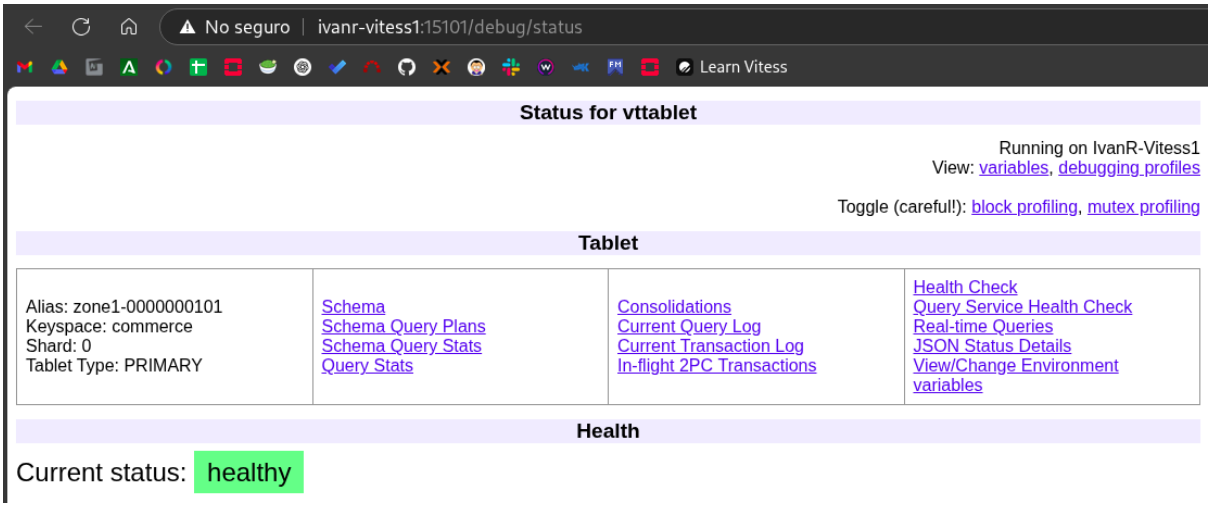
Si dentro de **Schemas** le damos a una tabla nos mostrará la definición de la tabla.

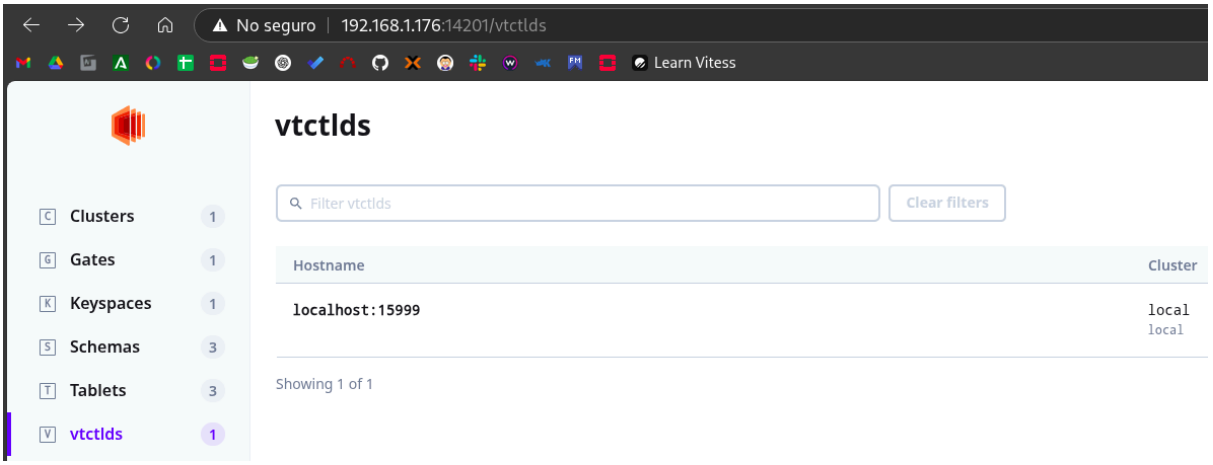


En **Tablets** podemos ver los tablets del servidor, el keyspace al que pertenecen, el nombre del shard donde están, el alias del tablet, el tipo, el estado y el servidor donde se encuentra. Podemos realizar acciones de validación para asegurarnos de que el tablet está operativo. Vemos que hay tres tablets en el keyspace, uno primario, otro de solo lectura y otro réplica. Si le damos al alias, podemos acceder a una pestaña que nos ofrece datos y opciones avanzadas del tablet.

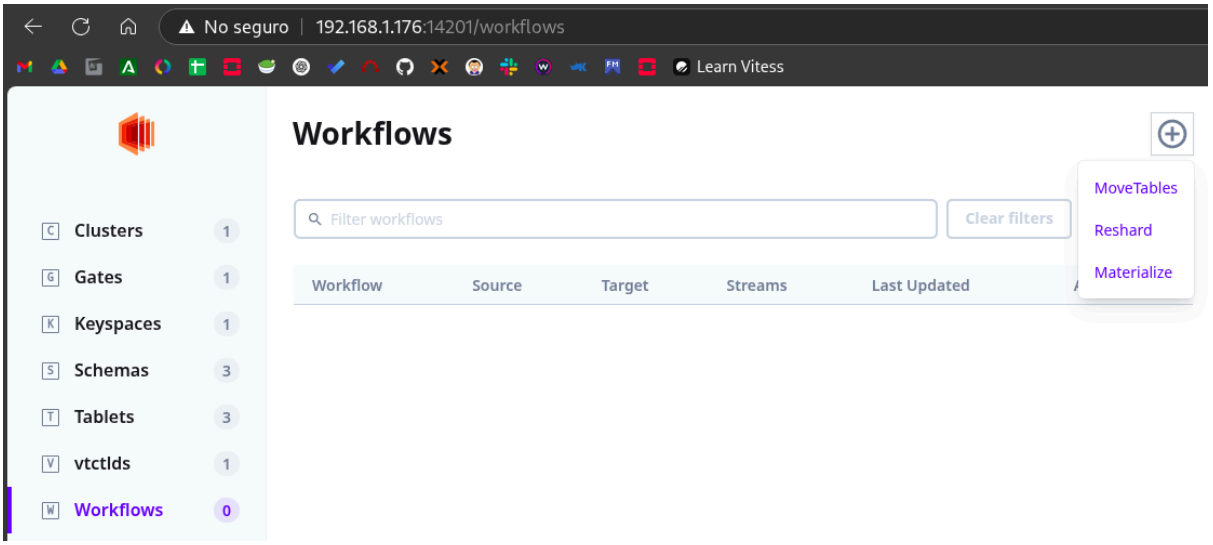


Si le damos al hostname nos mostrará una interfaz con el estado del tablet.



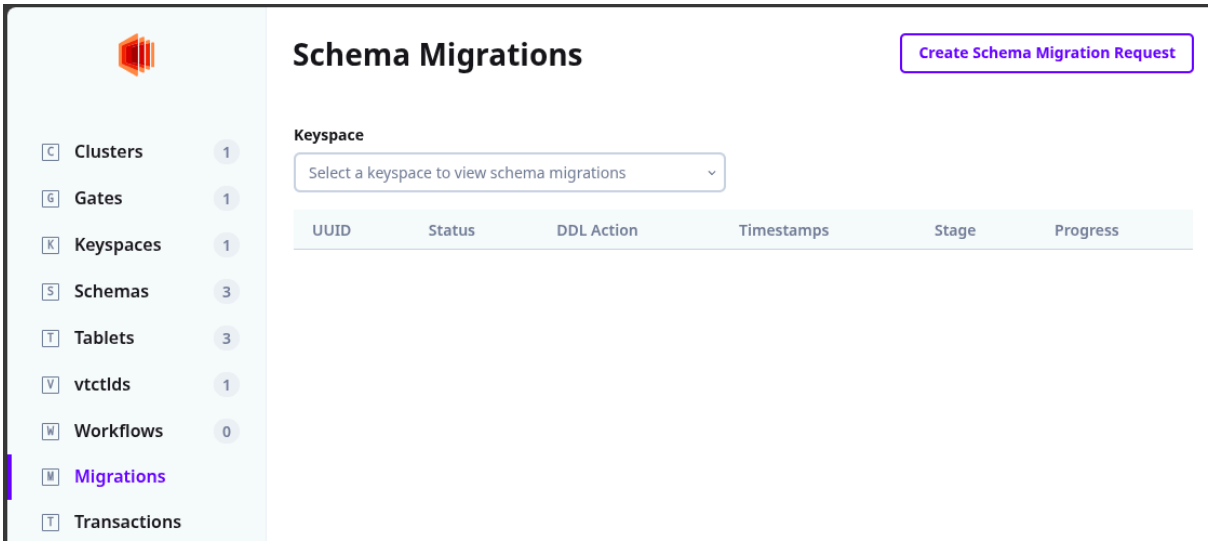


En **vtctlds** vemos los servidores vtctld actuales.

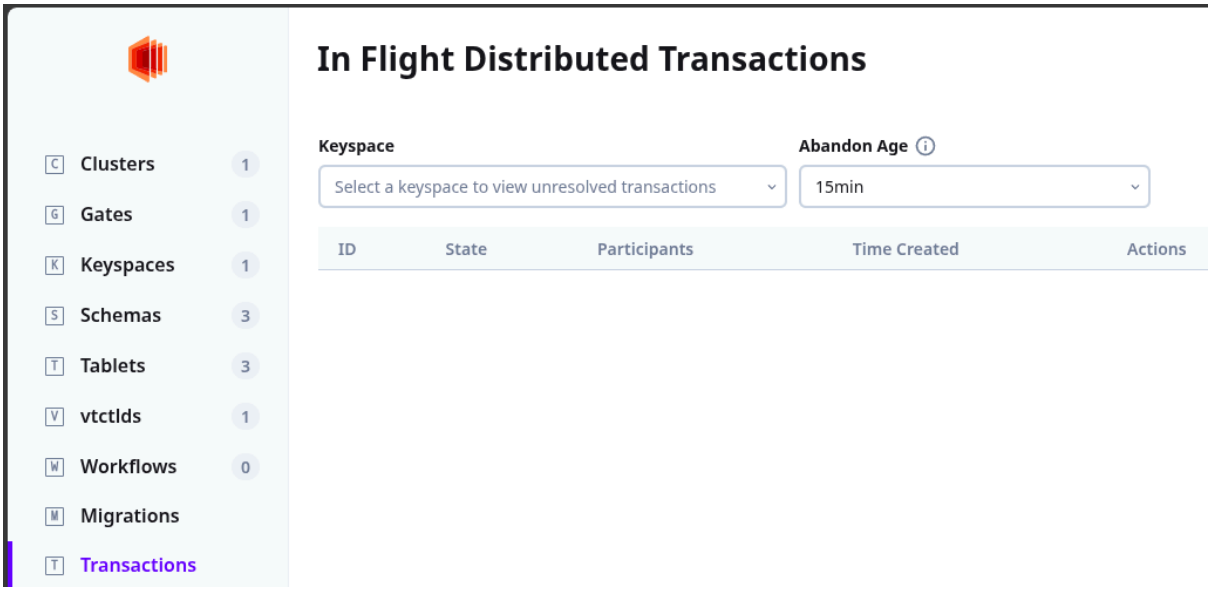


En **Workflows** podemos ver los flujos de trabajo actuales, que son tareas y operaciones avanzadas que podemos programar para realizar cambios en los

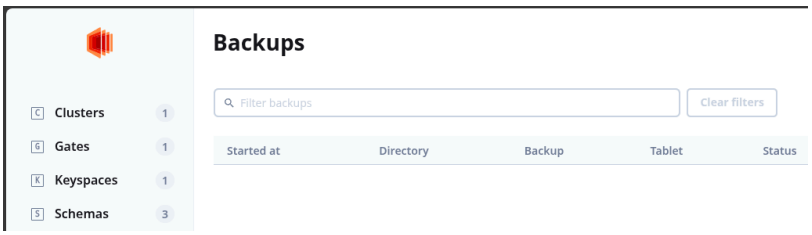
tablets y shards. Nos permite hacer MoveTables, Reshard y Materialize. Este último permite crear vistas materializadas en otro keyspace. Las dos primeras operaciones las probaremos y explicaremos más adelante.



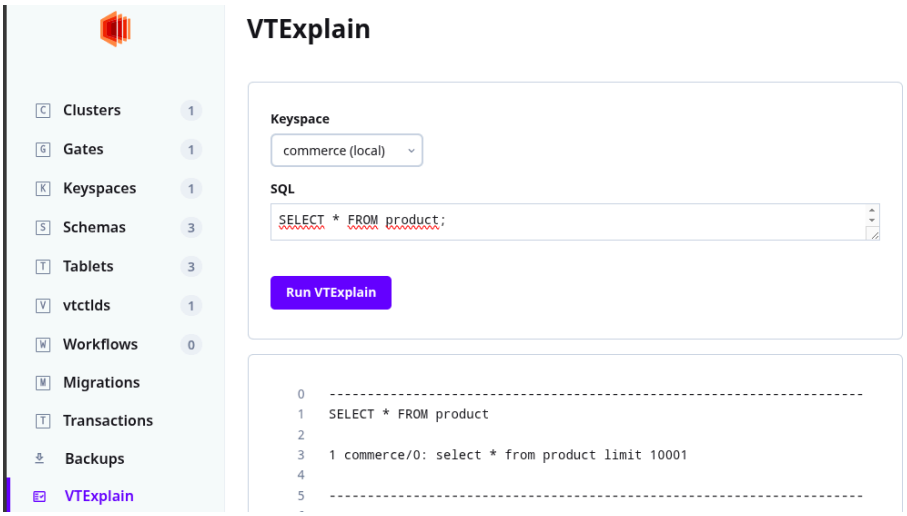
En **Migrations** podemos ver, controlar y supervisar las migraciones de esquemas (DDL) que se han solicitado en el servidor de Vitess.



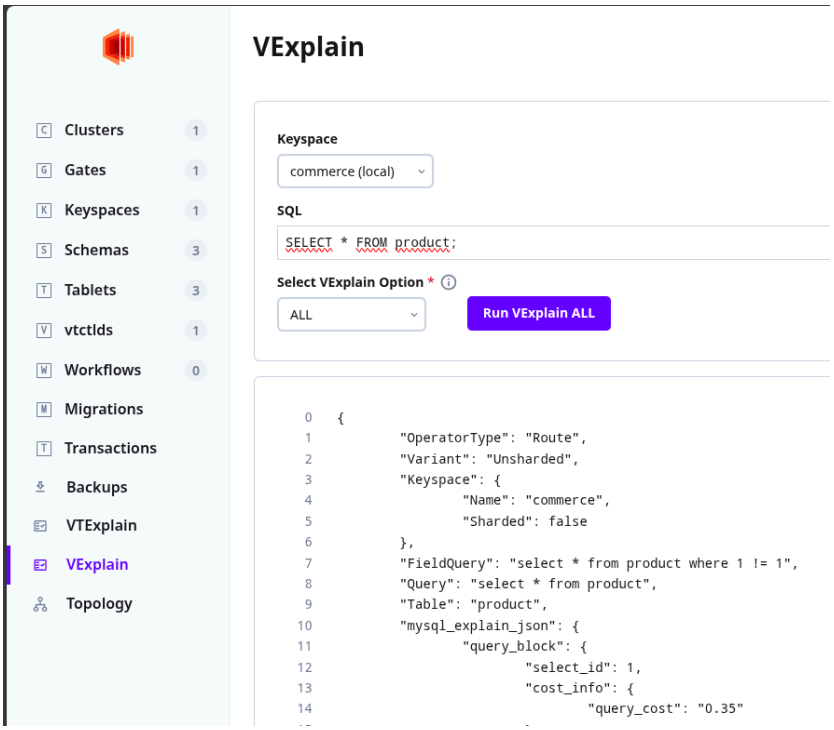
La pestaña **Transactions** te muestra las transacciones SQL activas que están ocurriendo en tu clúster de Vitess a través de VTGate. Una transacción es un bloque de operaciones SQL que se ejecutan juntas.



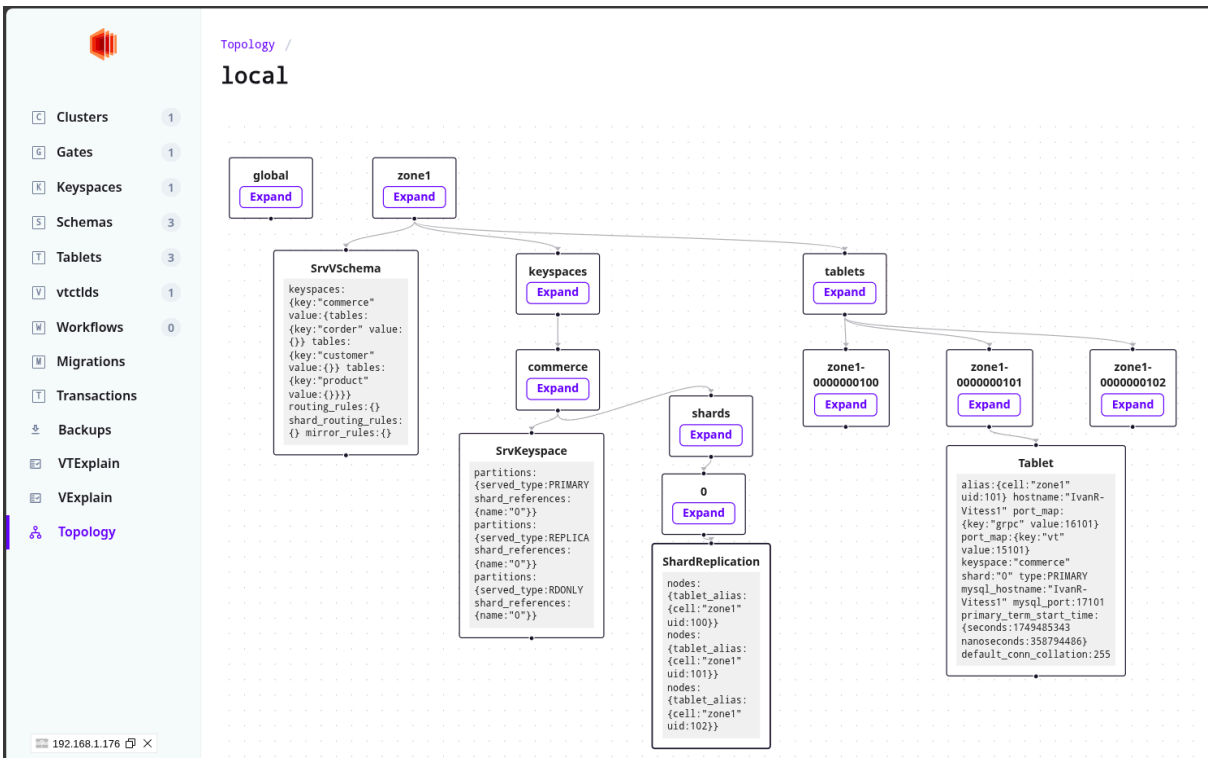
En **Backups** podemos ver, gestionar y restaurar copias de seguridad que se han realizado de los tablets. Más adelante las veremos mejor.



VTExplain es una herramienta que nos permite ver cómo Vitess plantea y ejecuta una consulta SQL a través de VTGate. Por ejemplo, vemos que Vitess ha añadido a la consulta un LIMIT 10001 para proteger el rendimiento del cluster.



VExplain es otra herramienta parecida pero más profunda. Simula el plan de ejecución lógico que seguirá el motor de Vitess para ejecutar una consulta. Como vemos muestra muchísima más información, como la información del keyspace, las tablas afectadas, el costo de ejecución en el servidor y las columnas de las tablas que se usarán.



En **Topology** podemos ver de forma gráfica y conceptual como están organizados los clusters y sus keyspaces. De forma gráfica nos ayuda a entender mejor cómo está configurado un cluster.

3.4 MoveTables

MoveTables es un tipo de WorkFlow que nos permite mover un set de tablas de un keyspace a otro sin tiempo de inactividad. En este apartado vamos a crear el keyspace **customer** y los tablets y después con un MoveTables moveremos las tablas **corder** y **customer** del keyspace **commerce** al keyspace **customer**.

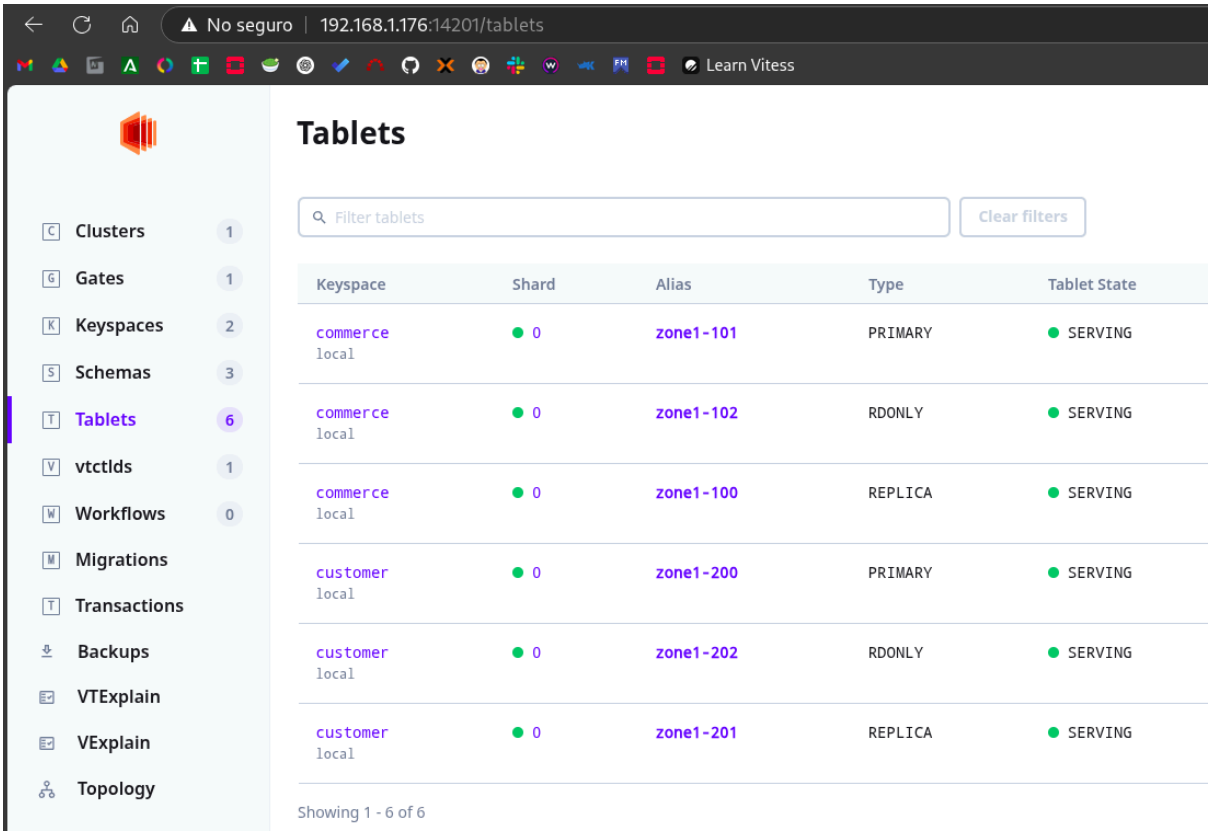
Primero vamos a insertar datos en las tablas.

```
$ mysql < ../common/insert_commerce_data.sql
$ mysql --table < ../common/select_commerce_data.sql
Using commerce
Customer
+-------------+--------------------+
| customer_id | email              |
+-------------+--------------------+
|           1 | alice@domain.com   |
|           2 | bob@domain.com     |
|           3 | charlie@domain.com |
|           4 | dan@domain.com     |
|           5 | eve@domain.com     |
+-------------+--------------------+
Product
+----------+-------------+-------+
| sku      | description | price |
+----------+-------------+-------+
| SKU-1001 | Monitor     |   100 |
| SKU-1002 | Keyboard    |    30 |
+----------+-------------+-------+
COrder
+----------+-------------+----------+-------+
| order_id | customer_id | sku      | price |
+----------+-------------+----------+-------+
|        1 |           1 | SKU-1001 |   100 |
|        2 |           2 | SKU-1002 |    30 |
|        3 |           3 | SKU-1002 |    30 |
|        4 |           4 | SKU-1002 |    30 |
|        5 |           5 | SKU-1002 |    30 |
+----------+-------------+----------+-------+
```

Ahora ejecutamos el siguiente script que creará el keyspace **customer** y tres tablets, uno primario, uno de solo lectura y otro de réplica.

```
$ ./201_customer_tablets.sh
Starting MySQL for tablet zone1-0000000200...
MySQL for tablet zone1-0000000200 is running!
Starting vttablet for zone1-0000000200...
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

vttablet for zone1-0000000200 is running!
Starting MySQL for tablet zone1-0000000201...
MySQL for tablet zone1-0000000201 is running!
Starting vttablet for zone1-0000000201...
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

vttablet for zone1-0000000201 is running!
Starting MySQL for tablet zone1-0000000202...
MySQL for tablet zone1-0000000202 is running!
Starting vttablet for zone1-0000000202...
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

vttablet for zone1-0000000202 is running!
{
  "keyspace": {
    "keyspace_type": "NORMAL",
    "base_keyspace": "",
    "snapshot_time": null,
    "durability_policy": "semi_sync",
    "throttler_config": null,
    "sidecar_db_name": "_vt"
  }
}
```

En VTAdmin podemos ver que se han creado correctamente.

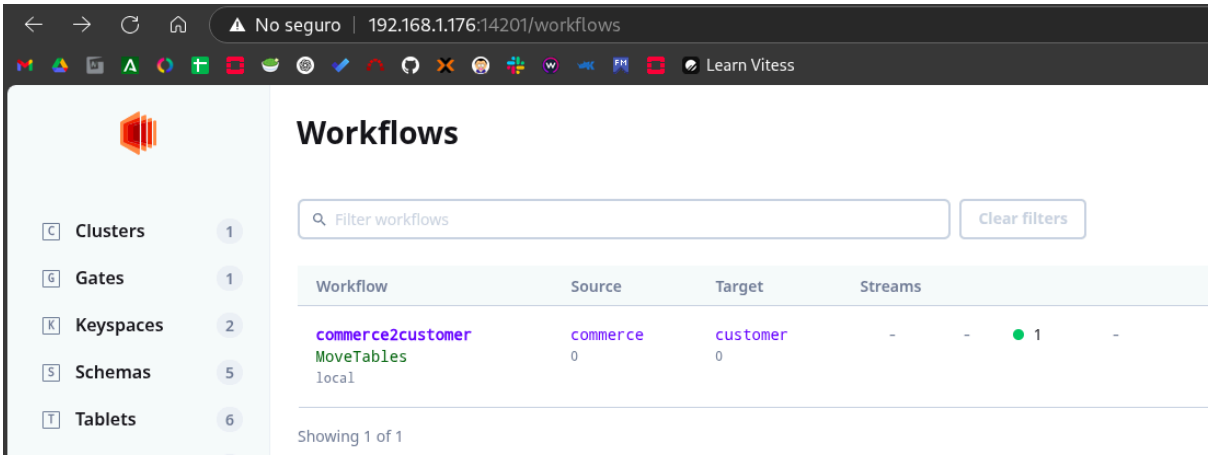


Ahora podemos crear el workflow que copiara las tablas del keyspace **commerce** a **customer**. Esto no interrumpirá ni bloqueará la actividad de la base de datos.

```
$ vtctldclient MoveTables --target-keyspace customer --workflow
commerce2customer create --source-keyspace commerce --tables
'customer,corder'

The following vreplication streams exist for workflow
customer.commerce2customer:

id=1 on customer/zone1-200: Status: Running. VStream has not
started.

Traffic State: Reads Not Switched. Writes Not Switched
```

Como vemos, la sintaxis del comando vtctldclient es muy sencilla para que podamos entenderla rápidamente. Este comando simplemente ha creado el workflow, los cambios no se han ejecutado todavía.

En VTAdmin podemos ver también el workflow creado.



Con este comando podemos ver el estado actual del workflow.

```
$ vtctldclient MoveTables --target-keyspace customer --workflow
commerce2customer status --format=json
{
  "table_copy_state": {},
  "shard_streams": {
    "customer/0": {
      "streams": [
        {
          "id": 1,
          "tablet": {
            "cell": "zone1",
            "uid": 200
          },
          "source_shard": "commerce/0",
          "position": "08e05fb8-454c-11f0-a4aa-525400e7e7e6:1-47",
          "status": "Running",
          "info": "VStream Lag: 0s"
        }
      ]
    }
  },
  "traffic_state": "Reads Not Switched. Writes Not Switched"
}
```

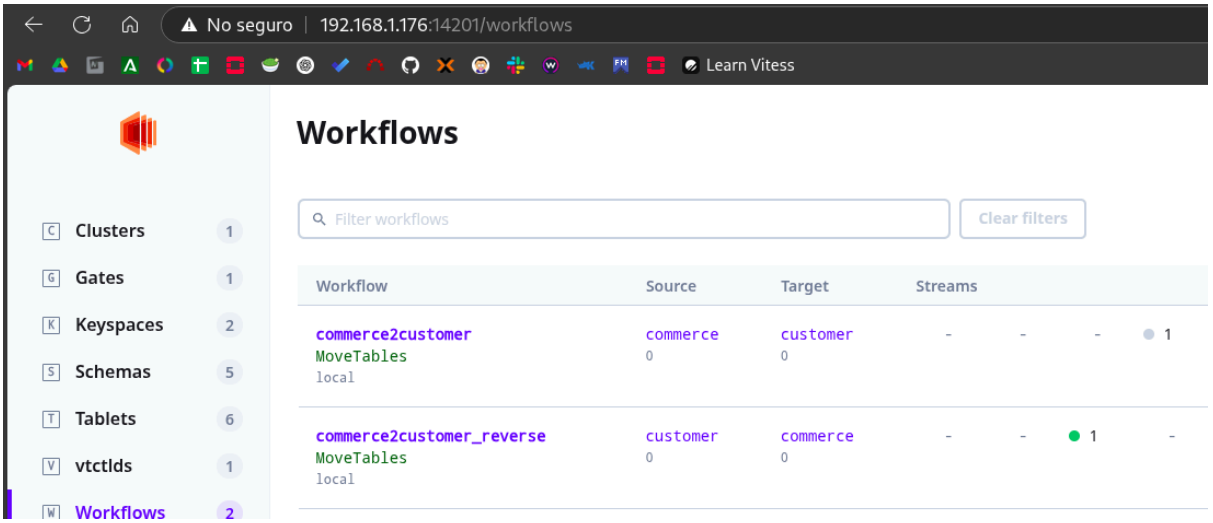
Tenemos la posibilidad de mover primero los tablets de lectura y luego los tablets de escritura.

```
# Mover tablets de lectura
$ vtctldclient MoveTables --workflow commerce2customer
--target-keyspace customer switchtraffic --tablet-types
"rdonly,replica"

# Mover tablets de escritura
$ vtctldclient MoveTables --workflow commerce2customer
--target-keyspace customer switchtraffic --tablet-types primary
```

La otra posibilidad es mover todos los tablets en un solo comando.

```
$ vtctldclient MoveTables --target-keyspace customer --workflow
commerce2customer SwitchTraffic
SwitchTraffic was successful for workflow customer.commerce2customer

Start State: Reads Not Switched. Writes Not Switched
Current State: All Reads Switched. Writes Switched
```

Una vez ejecutado vemos que se ha creado otro workflow.

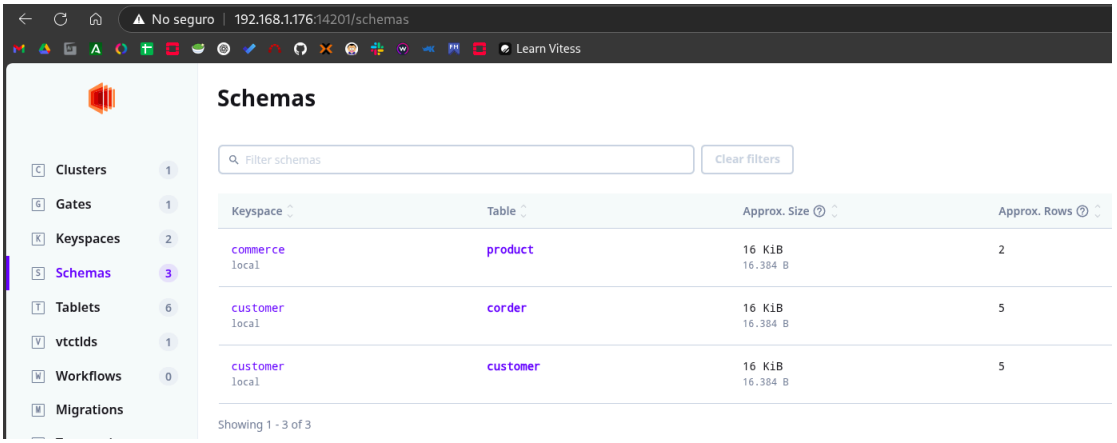


Este nuevo workflow nos permite revertir los cambios en caso de errores o fallos.

Tras esto ya podemos completar la migración de los datos con este comando.

```
$ vtctldclient MoveTables --target-keyspace customer --workflow
commerce2customer complete
```

Vemos que las tablas **corder** y **customer** ahora forman parte del keyspace **customer**.



3.5 Resharding

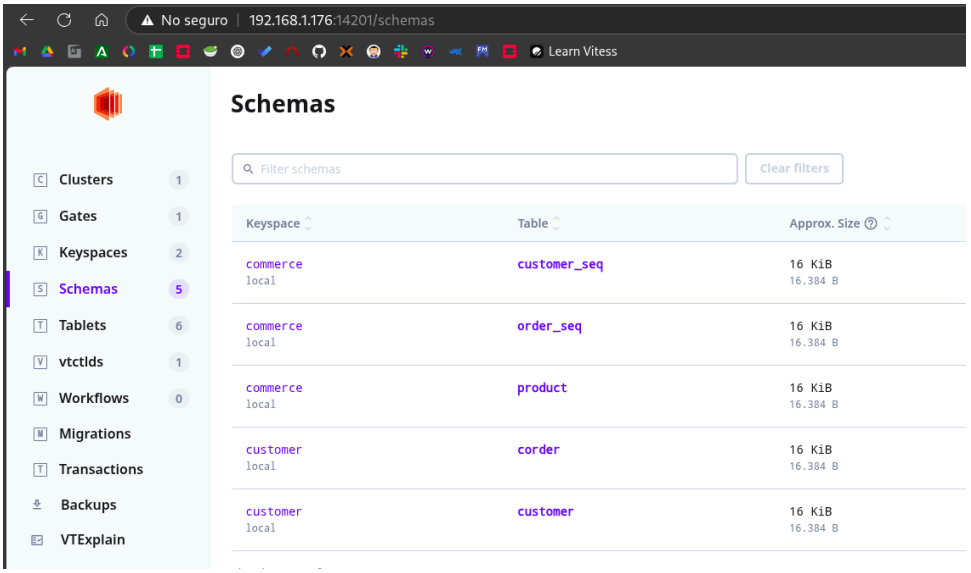
Sharding significa particionar la base de datos en partes más pequeñas llamadas shards, cada una con un subconjunto del total de datos. El resharding significa cambiar la forma en la que los datos están divididos por los shards. En este apartado vamos a dividir el shard del keyspace **costumer** en dos shards con los datos divididos.

El sharding y el campo autoincremental no se llevan bien. Por eso vamos a usar una herramienta que nos proporciona Vitess, las tablas de secuencias, que no están shardeadas. Vamos a crear las tablas de secuencia en el keyspace **commerce** (que no está shardeado) con un VSchema específico que hará las columnas autoincrementales. También modificaremos las tablas de **customer** para que asocien la columna autoincremental correctamente a las tablas en

commerce y los Vindexes para que el resharding se haga usando los campos clave correctos de cada tabla de **customer**.

```
$ vtctldclient ApplySchema --sql-file create_commerce_seq.sql commerce
$ vtctldclient ApplyVSchema --vschema-file vschema_commerce_seq.json
commerce
$ vtctldclient ApplyVSchema --vschema-file
vschema_customer_sharded.json customer
$ vtctldclient ApplySchema --sql-file create_customer_sharded.sql
customer
```

Vemos que se han creado las tablas de secuencia correctamente.

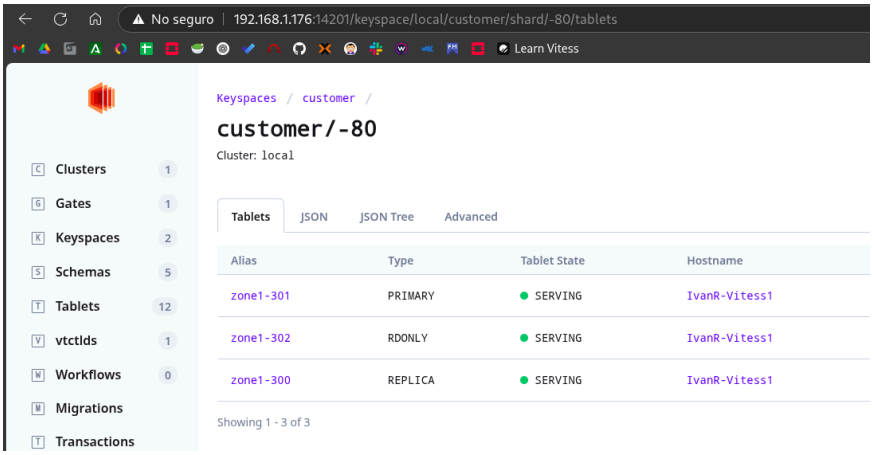


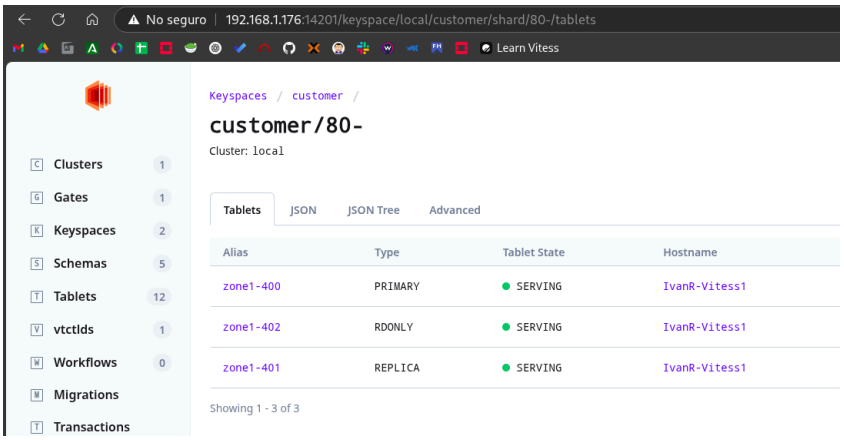
Ahora crearemos dos nuevos shards, 80- y -80. Cada uno de estos shards tendrá tres tablets asociados, uno primario, otro de solo lectura y otro de réplica. Esto lo haremos con el siguiente script.

```
$ ./302_new_shards.sh
Starting MySQL for tablet zone1-0000000300...
MySQL for tablet zone1-0000000300 is running!
Starting vttablet for zone1-0000000300...
vttablet for zone1-0000000300 is running!

Starting MySQL for tablet zone1-0000000301...
MySQL for tablet zone1-0000000301 is running!
Starting vttablet for zone1-0000000301...
vttablet for zone1-0000000301 is running!
```

```
Starting MySQL for tablet zone1-0000000302...
MySQL for tablet zone1-0000000302 is running!
Starting vttablet for zone1-0000000302...
vttablet for zone1-0000000302 is running!

Starting MySQL for tablet zone1-0000000400...
MySQL for tablet zone1-0000000400 is running!
Starting vttablet for zone1-0000000400...
vttablet for zone1-0000000400 is running!

Starting MySQL for tablet zone1-0000000401...
MySQL for tablet zone1-0000000401 is running!
Starting vttablet for zone1-0000000401...
vttablet for zone1-0000000401 is running!

Starting MySQL for tablet zone1-0000000402...
MySQL for tablet zone1-0000000402 is running!
Starting vttablet for zone1-0000000402...
vttablet for zone1-0000000402 is running!
```

Los tablets se han creado correctamente en sus shards correspondientes.





Ahora podemos hacer resharding moviendo los datos del shard 0 a los shards 80- y -80.

```
$ vtctldclient Reshard --target-keyspace customer --workflow
cust2cust create --source-shards '0' --target-shards '-80,80-'

The following vreplication streams exist for workflow
customer.cust2cust:

id=1 on customer/zone1-400: Status: Copying. VStream has not
started.
id=1 on customer/zone1-301: Status: Running. VStream has not
started.

Traffic State: Reads Not Switched. Writes Not Switched
```

Como vemos, la sintaxis del comando vuelve a ser muy sencilla. Podemos ver si el workflow se ha completado con estos comandos.

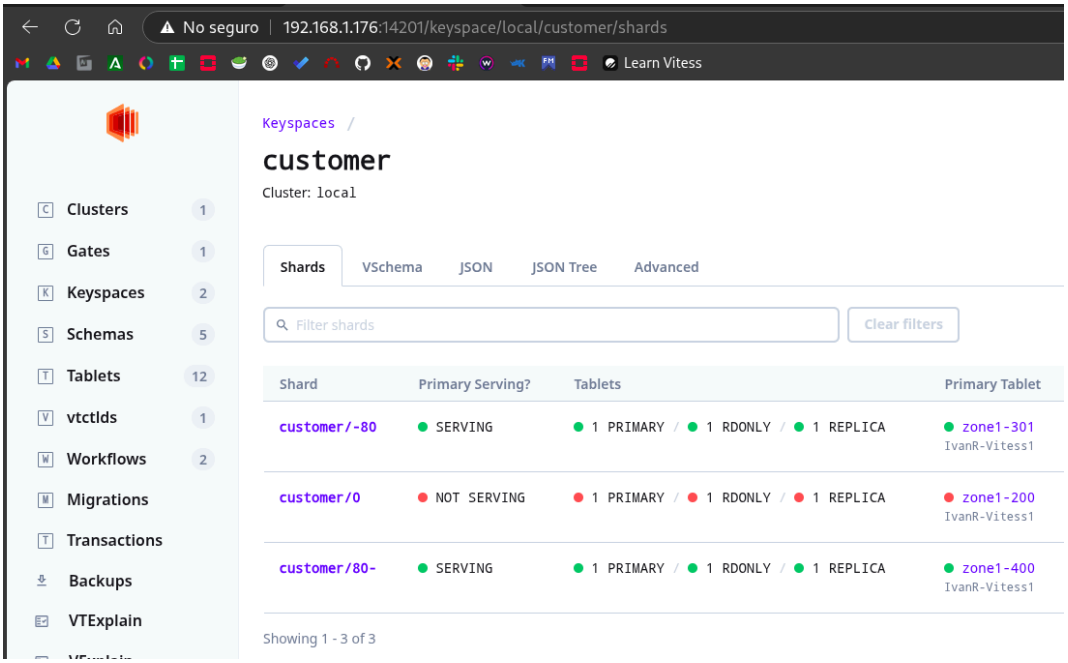
```
$ vtctldclient VDiff --target-keyspace customer --workflow cust2cust
create

VDiff 1094484a-2c36-42e9-8eff-e84cd8a70af5 scheduled on target
shards, use show to view progress

$ vtctldclient VDiff --target-keyspace customer --workflow cust2cust
show last

VDiff Summary for customer.cust2cust
(1094484a-2c36-42e9-8eff-e84cd8a70af5)
State:        completed
RowsCompared: 10
HasMismatch:  false
```

Ahora podemos pasar a cambiar las lecturas y escrituras de datos a los nuevos shards.

```
# Tablets de lecturas
$ vtctldclient Reshard --target-keyspace customer --workflow
cust2cust SwitchTraffic --tablet-types=rdonly,replica
SwitchTraffic was successful for workflow customer.cust2cust
```

```
Start State: Reads Not Switched. Writes Not Switched
Current State: All Reads Switched. Writes Not Switched
# Tablets de escrituras
$ vtctldclient Reshard --target-keyspace customer --workflow
cust2cust SwitchTraffic
SwitchTraffic was successful for workflow customer.cust2cust

Start State: All Reads Switched. Writes Not Switched
Current State: All Reads Switched. Writes Switched
```

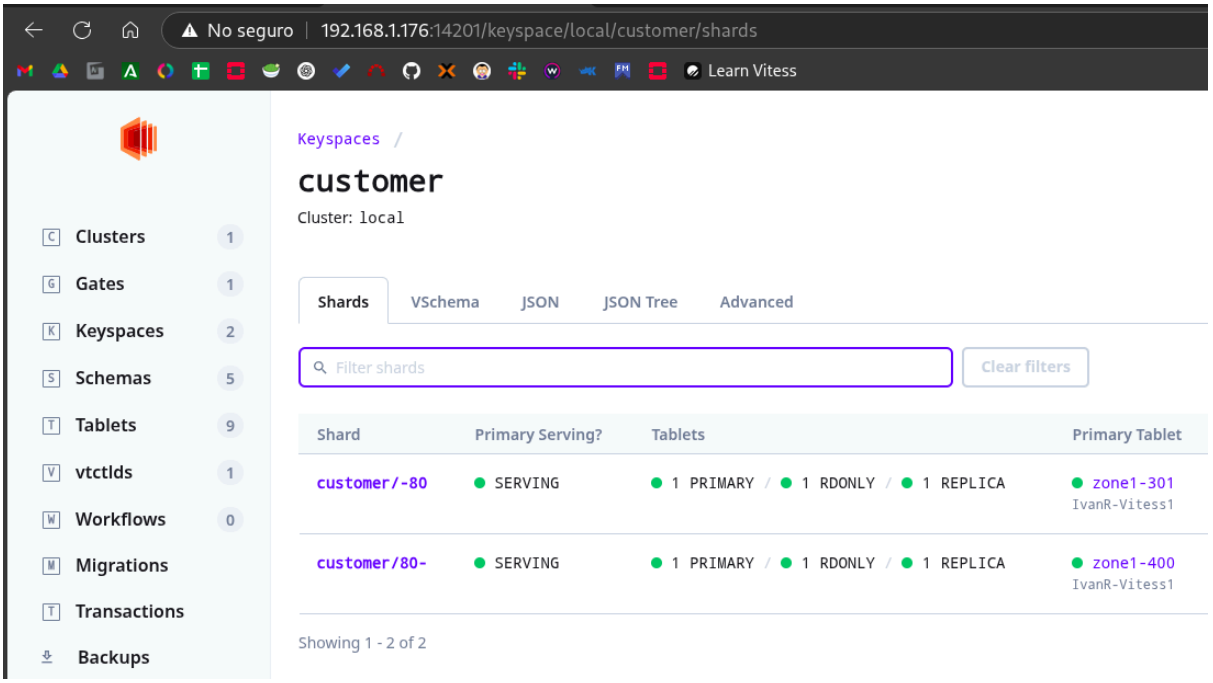
Vemos que ahora los shards activos son el -80 y el 80- y se desactivó el shard 0 en el keyspace **customer**.



Ahora vamos a marcar como completado el workflow de reshard, a parar los tablets del shard 0 y a eliminar el shard 0.

```
$ vtctldclient Reshard --target-keyspace customer --workflow
cust2cust complete

$ for i in 200 201 202; do
CELL=zone1 TABLET_UID=$i ../common/scripts/vttablet-down.sh
CELL=zone1 TABLET_UID=$i ../common/scripts/mysqlctl-down.sh
done

$ vtctldclient DeleteShards --force --recursive customer/0
$ rm -rf ${VTDATAROOT}/vt_000000020{0,1,2}/
```

Vemos que el shard 0 se ha borrado en **customer**.



Para probar que funciona el sharding he insertado estos datos.

```
$ mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 8.0.40-Vitess Version: 22.0.0
Copyright (c) 2000, 2025, Oracle and/or its affiliates.

mysql> use customer;
mysql> INSERT INTO customer (customer_id, email) VALUES
    -> (6, 'frank@domain.com'),
    -> (7, 'grace@domain.com'),
    -> (8, 'heidi@domain.com'),
    -> (9, 'ivan@domain.com'),
    -> (10, 'judy@domain.com');
Query OK, 5 rows affected (0,03 sec)

mysql> INSERT INTO corder (order_id, customer_id, sku, price) VALUES
    -> (6, 1, 'SKU-2001', 50),
    -> (7, 2, 'SKU-2002', 75),
    -> (8, 6, 'SKU-3001', 150),
    -> (9, 7, 'SKU-3001', 150),
    -> (10, 8, 'SKU-1002', 30),
    -> (11, 9, 'SKU-4000', 200),
    -> (12, 10, 'SKU-4000', 200);
Query OK, 7 rows affected (0,02 sec)
```

Vemos que los datos se dividen entre los dos shard correctamente.

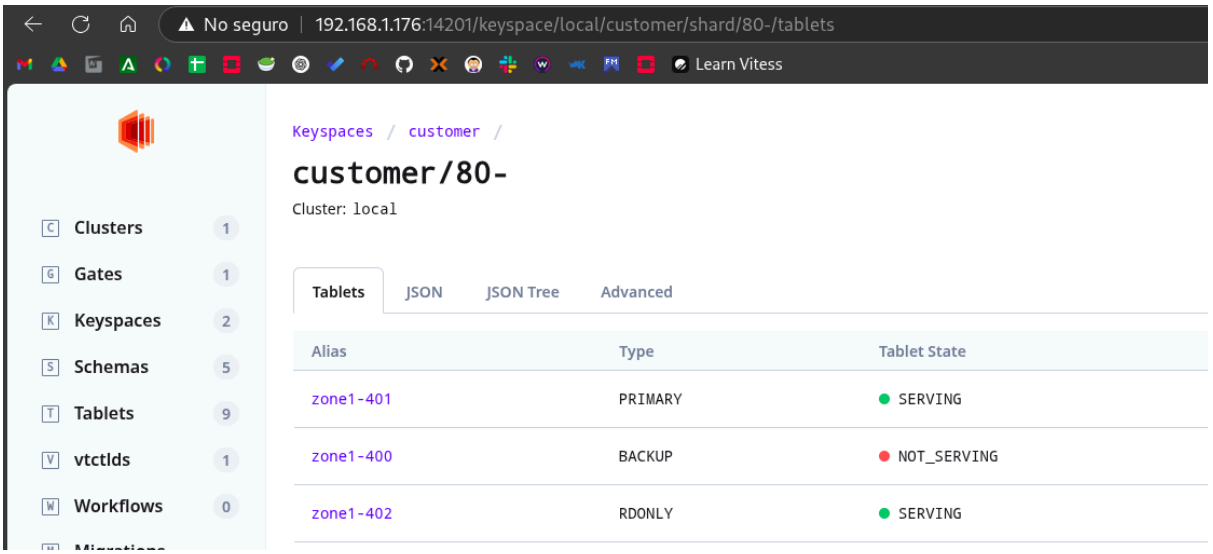
```
$ mysql --table < ../common/select_customer80-_data.sql
Using customer/80-
+-------------+--------------------+
| customer_id | email              |
+-------------+--------------------+
|           1 | alice@domain.com   |
|           2 | bob@domain.com     |
|           3 | charlie@domain.com |
|           4 | dan@domain.com     |
|           5 | eve@domain.com     |
|           6 | frank@domain.com   |
|           7 | grace@domain.com   |
|           9 | ivan@domain.com    |
+-------------+--------------------+
+----------+-------------+----------+-------+
| order_id | customer_id | sku      | price |
+----------+-------------+----------+-------+
|        1 |           1 | SKU-1001 |   100 |
|        2 |           2 | SKU-1002 |    30 |
|        3 |           3 | SKU-1002 |    30 |
|        4 |           4 | SKU-1002 |    30 |
|        5 |           5 | SKU-1002 |    30 |
|        6 |           1 | SKU-2001 |    50 |
|        7 |           2 | SKU-2002 |    75 |
|        8 |           6 | SKU-3001 |   150 |
|        9 |           7 | SKU-3001 |   150 |
|       11 |           9 | SKU-4000 |   200 |
+----------+-------------+----------+-------+
$ mysql --table < ../common/select_customer-80_data.sql
Using customer/-80
+-------------+------------------+
| customer_id | email            |
+-------------+------------------+
|           8 | heidi@domain.com |
|          10 | judy@domain.com  |
+-------------+------------------+
+----------+-------------+----------+-------+
| order_id | customer_id | sku      | price |
+----------+-------------+----------+-------+
|       10 |           8 | SKU-1002 |    30 |
|       12 |          10 | SKU-4000 |   200 |
+----------+-------------+----------+-------+
```

3.6 Copias de seguridad y restauración

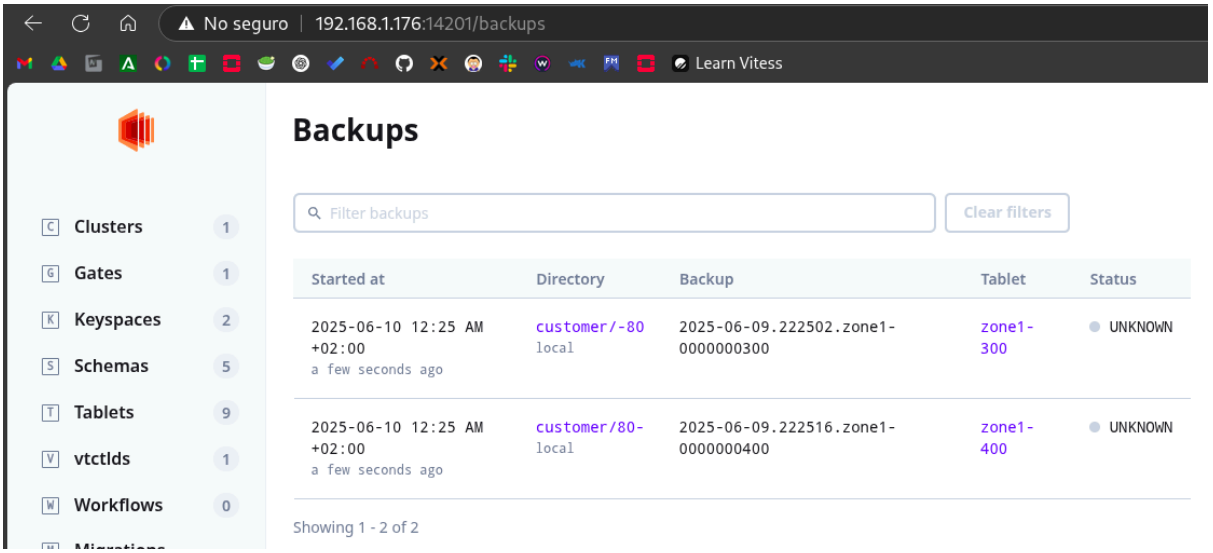
Podemos hacer copias de seguridad de shards de un keyspace con este comando.

```
$ vtctldclient BackupShard "customer/-80"
$ vtctldclient BackupShard "customer/80-"
```

Durante la copia de seguridad, como vimos en la teoría, el tablet **replica** deja de replicar para transformarse en un tablet de tipo **backup** y hacer la copia de seguridad .



Vemos que las copias de seguridad se han realizado con éxito.



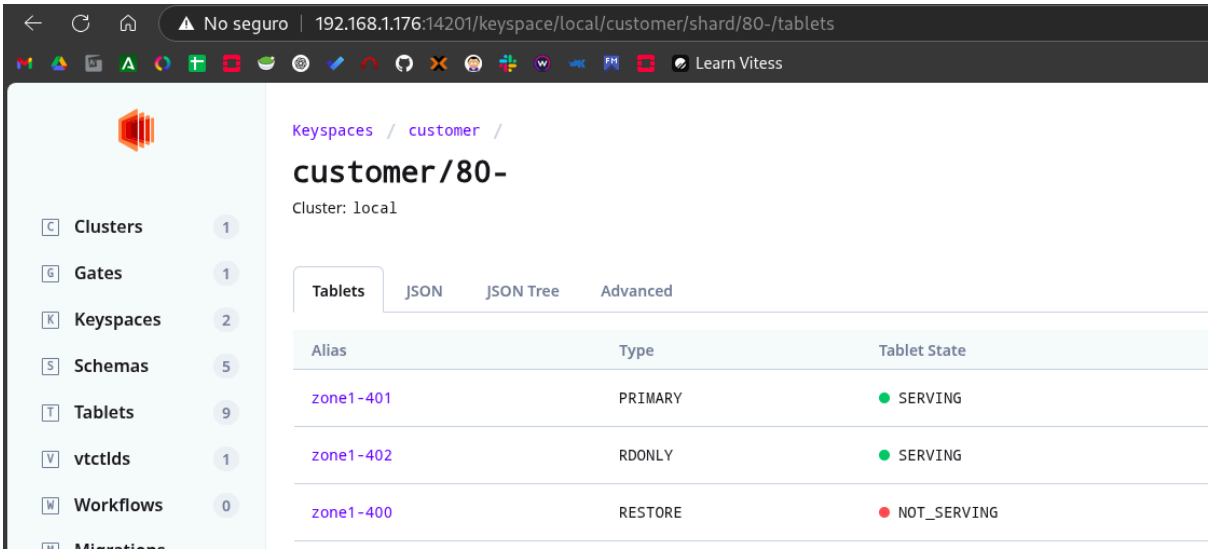
Así también podemos listar las copias de seguridad.

```
$ vtctldclient GetBackups "customer/-80"
2025-06-09.222502.zone1-0000000300

$ vtctldclient GetBackups "customer/80-"
2025-06-09.222516.zone1-0000000400
```

Para restaurar se tiene que el tablet de tipo réplica del shard.

```
$ vtctldclient RestoreFromBackup "zone1-0000000300"
$ vtctldclient RestoreFromBackup "zone1-0000000400"
```

Como vimos en la teoría, durante el proceso de restauración el tablet seleccionado se transforma en un tablet de tipo **restore**.



4. Conclusiones y propuestas para Vitess

Vitess es el futuro de las bases de datos MySQL de alta demanda. Me he dado cuenta que tiene muchísimas funciones que simplifican en muchos aspectos el manejo de MySQL y una gran cantidad de mejoras que nos da con simplemente instalarlo, colocándose por encima de sus programas competidores. Es la opción de software libre con más ventajas que las empresas pueden elegir. Lo único que mejoraría es una mejor guía por parte de Vitess para los usuarios que no tengan experiencia en los sistemas distribuidos.

Este proyecto ha sido una simple introducción y prueba de algunas de las funciones clave de Vitess, pero me he quedado con ganas de más. Una propuesta para poder seguir expandiendo los conocimientos en Vitess sería conseguir separar los componentes claves de Vitess, como los shards, vtgate o vtadmin en distintas máquinas virtuales para poder tener el sistema de base de datos mejor distribuido y aprovechar más recursos. Esta propuesta es posible pero hace falta profundizar más en el servidor de topología (que controla el acceso entre máquinas) y también tener equipos con más recursos; aunque se puede probar en Kubernetes, que es otra posible propuesta.

5. Dificultades que se han encontrado

En el proyecto solo he encontrado dos dificultades a la hora de ejecutar Vitess por primera vez:

- El script que inicia vtadmin fuerza a usar node 22 y lo instala usando una carpeta de módulos propia del repositorio de github de Vitess. Esta carpeta llamada **/repositorio-vitess/web/vtadmin/node_modules** tiene algún error en una de las dependencias y no permite iniciar vtadmin correctamente. La solución fue borrar la carpeta, forzando al script a descargar node 22 y sus dependencias de forma online y correcta.

- Vitess por defecto plantea llamar a las máquinas que lo forman por su FQDN y usarlo para comunicarse entre las máquinas. Si no tenemos registrados estos nombres en el servidor DNS de la red, Vitess fallará en varios componentes como en la web de VTAdmin, que no podrá conectarse a la API de VTAdmin y mostrar los datos de las bases de datos. La solución es reemplazar en dos variables el FQDN por la dirección IP del equipo.

```
$ grep hostname env.sh
hostname=$(hostname -f)

$ grep hostname env.sh
hostname="192.168.1.176"

$ grep hostname vtadmin-up.sh
case_insensitive_hostname=$(echo "$hostname" | tr '[:upper:]'
'[:lower:]')

$ grep hostname vtadmin-up.sh
case_insensitive_hostname="192.168.1.176"
```

6. Repositorio y vídeos

Repositorio de GitHub donde se encuentra la carpeta IvanR-Vitess1 de las pruebas:
https://github.com/IvanRuiperezB/Ivanr-Vitess

Vídeos sin sonido que usaré de respaldo en caso de que algo falle durante las demos el día de la presentación:

- Iniciando Vitess
- MoveTables
- Resharding
- Backups

7. Bibliografía

Documentación teórica oficial de Vitess 22: https://vitess.io/docs/22.0/overview/

Curso de Vitess de PlanetScale: https://planetscale.com/learn/courses/vitess

Instalación de Vitess 22.0: https://vitess.io/docs/22.0/get-started/local/

Instalación en Debian de Vitess: Enlace

Conceptos de Vitess: https://vitess.io/docs/22.0/concepts/

Guías de usuario de Vitess: https://vitess.io/docs/22.0/user-guides/