Pasos para crear arquitectura multitenant

Lo primero que se debe tener en cuenta es que debe existir un servidor para administrador y otro para los tenant.

**Servicios para administrador multitenant**:

Creación de máquina virtual para alojar la base de datos y demás para administrar los tenants que se pretendan integrar a la aplicación:

Para mayor eficiencia, estandarización y facilidad en replica, se usa una máquina virtual de linux, con el ubuntu server 20.04 LTS - Gen2 x64

Se habilita el puerto 22, de SSH para acceder al servidor y administrarlo desde consola bash.

1. Conectar con servidor via SSH:

Abrir una ventana de comandos y copiar lo siguiente

1. ssh username@server-ip

2. Copiar la contraseña de la máquina virtual

2. Instalar java17 y la base de datos SQL, en este caso será MySQL

1. Actualizar paquetes: sudo apt update

2. Instalar JDK de java 17: sudo apt install -y openjdk-17-jdk

La instalación del Apache tomcat no es necesaria, tampoco la de Apache2, ya que el paquete .jar del api tiene un servidor embebido.

3. Instalar Apache tomcat 10: basado en https://www.digitalocean.com/community/tutorials/how-to-install-apache-tomcat-10-on-ubuntu-20-04

- Agregar superusuario tomcat: sudo useradd -m -d /opt/tomcat -U -s /bin/false tomcat

- acceder a la carpeta tmp: cd /tmp/

- Descargar paquete tomcat version 10.1.19: sudo wget https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.19/bin/apache-tomcat-10.1.19.tar.gz

- Descomprimir paquete de tomcat en /opt/tomcat: sudo tar xzvf apache-tomcat-10\*tar.gz -C /opt/tomcat --strip-components=1

- Conceder permisos de owner al usuario tomcat: sudo chown -R tomcat:tomcat /opt/tomcat/ y sudo chmod -R u+x /opt/tomcat/bin

- Acceder a la configuracion de usuarios y roles: sudo nano /opt/tomcat/conf/tomcat-users.xml

- Agregar los roles y usuarios, en este caso seria: <role rolename="manager-gui" />

<user username="manager" password="manager\_password" roles="manager-gui" />

<role rolename="admin-gui" />

<user username="admin" password="admin\_password" roles="manager-gui,admin-gui" />

- Comando para iniciar tomcat desde el inicio del servidor: sudo systemctl enable tomcat

3. Copiar el archivo .jar del ejecutable de la API en el servidor por medio de ssh, lo cual se hace con el comando:

*scp /documentpath.extension username@IpServerAddress:/path/to/copy*

Si ocurre algún error en este caso, validar que el puerto 22 este abierto en la configuración de redes de la maquina virtual en Azure o el hosting que sea, y que también este permitido en el firewall del Ubuntu server, lo cual se hace por medio del comando:

*sudo ufw status*

En caso de que el puerto 22 no este habilitado en el firewall del servidor, usar el siguiente comando para habilitarlo:

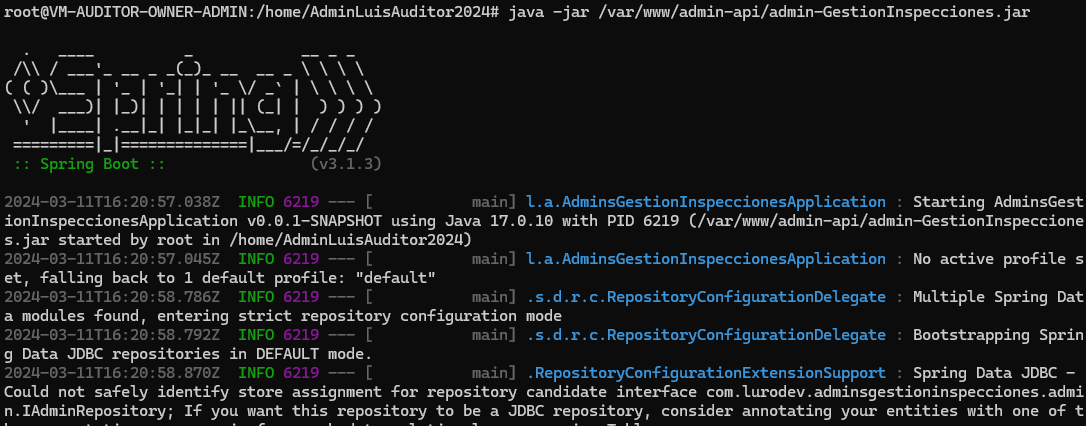
*sudo ufw allow 22*

Después de validar que el puerto 22 este habilitado por el firewall, copiar del archivo a la ruta deseada en el servidor.

4. Después de copiado el archivo .jar en el servidor, validar que la API se pueda ejecutar, copiando el siguiente comando en el servidor:

java -jar /ruta/al/archivo/filename.jar

Deberá aparecer algo parecido a esto:



Al validar que la API se esta ejecutando, validar que el puerto en el cual se esta sirviendo la API este abierto por el firewall y en las configuraciones de la maquina virtual de Azure; luego de esto, acceder a la IP del servidor desde cualquier dispositivo por medio del navegador, copiando <http://server-ip:api-port>.

Apareciendo algo parecido a esto:



Al evidenciar que la API se esta ejecutando, finalizar la ejecución con el comando ctrl+c

5. Ejecutar la API en segundo plano en el servidor y forzar su ejecución cada vez que se inicie el servidor, para lo cual se debe registrar la API como servicio del servidor, con los siguientes pasos:

- Crear un archivo de servicio con la configuración para ejecutar la API, usando el siguiente comando: sudo nano /etc/systemd/system/api-start.service

Se ubica obligatoriamente el archivo en /etc/systemd/system/ porque en esta carpeta se ubican los servicios del sistema.

* Agregar lo siguiente al archivo api-start.service:
* [Unit]
* Description=My API Name
* After=network.target
* [Service]
* User=root
* Group=root
* ExecStart=/usr/bin/java -jar /ruta/a/tu/api.jar
* WorkingDirectory=/ruta/a/tu/directorio
* Restart=always
* [Install]

WantedBy=multi-user.target

Se asignan en User y Group a root, ya que es lo correspondiente para el administrador del servidor.

Después cerrar el archivo guardando cambios.

- Después de configurado el servicio, se procede con su habilitación e inicio, tener en cuenta que todos los servicios en Linux manejan su ciclo de vida con systemctl, siendo enable, disable, start, restart, stop; en este caso se usan los siguientes comandos ejemplificantes:

*sudo systemctl enable api-start*

*sudo systemctl start api-start*

Con estos comandos ya debería estar ejecutándose la API, lo cual se puede validar yendo al navegador, copiar <http://server-ip:api-port>

Otra forma de saber el estado del servicio es con el comando status de systemctl:

*sudo systemctl status api.start*

4. Configurar el firewall para permitir las conexiones, de acuerdo con el puerto: sudo ufw allow [portNumber]

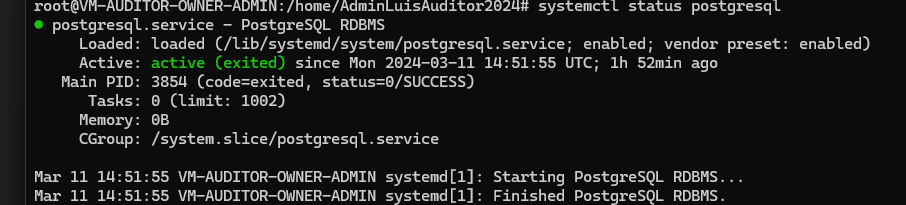
5. Instalar motor de base de datos postgresql:

- sudo apt update

- sudo apt install postgresql

- Despues de instalado copiar el siguiente comando para conocer el puerto por el cual se ejecuta la base de datos: *sudo ss -tlnp | grep postgres*

Tener presente que postgresql se estará ejecutando como un servicio por tanto está sujeto al ciclo de vida de los servicios, de start, stop, enable, restart…



5. Instalar base de datos MySQL server:

- sudo apt install -y mysql-server

5.1 Acceder a mysql como root: mysql -u root -p

6. Crear base de datos:

1. Crear el usuario administrador de la DB:

CREATE USER admin\_auditor WITH PASSWORD 'LuisDevMed@2022\*’;

2. Crear base de datos admin\_auditor\_db:

CREATE DATABASE admin\_auditor\_db WITH OWNER admin\_auditor;

3. Asignar rol de superusuario a admin\_auditor:

ALTER USER admin\_auditor WITH SUPERUSER;

Salir de la consola de postgresql, copiando **exit** 2 veces

**Configuración servidor multitenant**

Se crea el servidor en Ubuntu 20.04, con un balanceador de carga, ya que este servidor será el responsable de alojar las APIs que alimentan la aplicación y tendrán mayor trafico de datos de entrada y salida.

1. Instalar los programas necesarios:

Los programas necesarios son un servidor de base de datos SQL MySQL, Docker (microservices), y jdk 17 de java

* Actualizar paquetes inicialmente: sudo apt update
* Instalar paquetes: sudo apt install -y mariadb-server openjdk-17-jdk

1. Crear el script en bash (para linux) que contendrá los comandos para crear la base de datos de cada tenant:

* **Crear el archivo con extensión sh**:

*sudo nano createdb.sh*

* **Script para crear base de datos**:

Este script va a recibir el nombre de la base de datos y el usuario que se asignara como **superusuario** de esa base de datos. Antes de ejecutar este Script la API tuvo que haber hecho una validación de que la empresa no exista y que el número de identificación “NIT” y nombre de la empresa sea único.

Para crear la base de datos del tenant, se toma el nombre de la empresa y su NIT.

Nombre de base de datos: db\_[nombreEmpresa][NIT]

Administrador base de datos: admindb\_[nombreEmpresa][NIT]

El script .sh es el siguiente:

#!/bin/bash

# Verificar si se proporcionan suficientes argumentos

if [ $# -ne 2 ]; then

echo "Uso: $0 nombre\_basedatos usuario"

exit 1

fi

# Asignar argumentos a variables

DB\_NAME="$1"

DB\_USER="$2"

ROOT\_PASS=LuisDevMed@2022\*

# Comando para crear la base de datos

mysql -u root -p"$ROOT\_PASS" -e "CREATE DATABASE IF NOT EXISTS $DB\_NAME DEFAULT CHARACTER SET utf8 COLLATE utf8\_unicode>

# Comando para verificar si la base de datos se creó exitosamente

DB\_EXIST=$(mysql -u root -p"$ROOT\_PASS" -se "SELECT SCHEMA\_NAME FROM INFORMATION\_SCHEMA.SCHEMATA WHERE SCHEMA\_NAME='$DB>

if [ "$DB\_EXIST" = "$DB\_NAME" ]; then

# Comando para crear un usuario y concederle permisos sobre la base de datos

mysql -u root -p"$ROOT\_PASS" -e "CREATE USER '$DB\_USER'@'localhost' IDENTIFIED BY '$ROOT\_PASS';"

mysql -u root -p"$ROOT\_PASS" -e "GRANT ALL PRIVILEGES ON $DB\_NAME.\* TO '$DB\_USER'@'localhost';"

mysql -u root -p"$ROOT\_PASS" -e "FLUSH PRIVILEGES;"

echo "true"

else

echo "false"

fi

El llamado a este script se hace simplemente copiando lo siguiente en la ventana de comandos:

Ubicarse en la carpeta donde está el archivo: cd /ubicación/carpeta

Ejecutar el script: ./script.sh

Tener en cuenta que al crearse una base de datos para un tenant (empresa), también se debe crear una API que se conecte con esta nueva base de datos; es decir, para cada base de datos, debe haber un nuevo conjunto de APIs (replicas).

1. Crear el Script para replicar la API, lo cual incluye; crear nuevo directorio con el nombre y ID del tenant, éste tendrá dentro:

* El **application.properties**, el cual configura la api java, este documento debe tener información del “server.servlet.context-path” para mostrar la API en una URL base correspondiente al ID del tenant. El nombre de la base de datos y usuario administrador del tenant.
* El .jar de las APIs java que se vayan a usar.
* Archivo “.service” para convertir las APIs en servicios ejecutables al inicio del servidor.

1. Crear script para ejecutar los servicios correspondientes a las APIs. Si una API no funciona, validar que el puerto que usa este permitido, y ejecutar el comando:

*systemctl status serviceapi*

1. Crear la API y el llamado desde la misma para ejecutar los scripts.
2. Replicar las APIs que se van a conectar a la base de datos del tenant nuevo creado

Para replicar las APIs, se va a crear un directorio nuevo para cada tenant, el cual contenga todas las APIs a ejecutar, el archivo de configuración de las APIs (el mismo para todas) y el script del servicio para convertir las APIs en servicios que se inician con el servidor.

Al replicar las APIs se debe tener en cuenta que se deberá tener el mismo puerto para el microservicio, pero se debe asignar una url base diferente, la cual indique el tenant al cual va a servir. Porque no es correcto ni sostenible asignar un puerto a cada servicio, sin saber cuanto vaya a crecer la cantidad de tenants.

Para lo anterior, se tendrá un script .sh, el cual será llamado después de crear satisfactoriamente la base de datos.