**Informe del Deployment**

**Sistema de la máquina de café**

**Integrantes**

**Gabriel Suarez Baron**

**Alejandro Varela Franco**

**Alexander Sánchez Sánchez**

**Arquitectura de Software**

**ICE**

**28 de abril del 2023**

**Universidad ICESI**

Contenido

[Objetivo 3](#_Toc133533612)

[Componentes 3](#_Toc133533613)

[Integración continua 3](#_Toc133533614)

[Configuración de la base de datos 4](#_Toc133533615)

[Archivos de la base de datos 4](#_Toc133533616)

[Conexión a la base de datos 4](#_Toc133533617)

[Compilación archivos con Gradle 5](#_Toc133533618)

[Despliegue Servidor Central 7](#_Toc133533619)

[Despliegue Máquina de café 8](#_Toc133533620)

[Uso del nmap 11](#_Toc133533621)

[Diagrama de Deployment 13](#_Toc133533622)

# Objetivo

Poner en práctica el despliegue de un sistema de software distribuido sencillo, teniendo en cuenta las consideraciones y factores que ello implica, por ejemplo, la instalación y configuración de la base de datos.

# Componentes

Para realizar el despliegue del sistema de la máquina de café donde se busca maximizar ventas y disminuir costos se utilizaron los siguientes componentes.

|  |  |
| --- | --- |
| **Componente** | **Nombre de la PC remota** |
| Database | hgrid2 |
| Coffeemach | hgrid9 |
| Coffeemach | hgrid10 |
| CentralServer | hgrid11 |
| Cm\_logistics | No desplegado |
| Warehouse | No desplegado |

Para realizar este despliegue decidimos dividirnos el trabajo donde cada integrante del equipo desplego un componente. En este caso Alexander Sánchez fue el encargado de desplegar la base de datos, Alejandro Varela fue el encargado de desplegar el servidor central y Gabriel Suarez fue el encargado de desplegar las dos máquinas de café.

# Integración continua

Para integrar los diferentes conocimientos que hemos obtenidos en las diversos cursos que hemos tomado en la universidad, decidimos realizar un procesos integración continua, el cual, será el encargado de automatizar el despliegue de la máquina de café. Este es un procesos que aún esta en proceso y para saber cómo funciona dirigirse al siguiente repositorio de [GitHub](https://github.com/alejandro945/distributed-coffee-machine), donde estará explicado paso a paso este proceso. Para realizar esta integración se utilizó la tecnología de Jenkins usando los pipelines.

# Configuración de la base de datos

1. El primer paso para configurar la base de datos fue crear el usuario para el equipo. Este usuario lleva por nombre cofmachu\_ssv y además tiene la contraseña, la cual es cofmachpwd. (Ver linea 1)
2. Una vez creado el usuario, pasamos a crear nuestra base de datos, con nombre coffeemachine y le asignamos un propietario, el cual es el usuario creado anteriormente. (Ver linea 2)
3. Finalmente conectamos nuestra basa de datos. (Ver linea 3)

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

# Archivos de la base de datos

Para configurar nuestra base de datos, contamos con dos archivos .sql, los cuales son los encargados de crear las tablas y de realizar los insterts, a continuación, vamos a ver el contenido de cada uno de ellos.

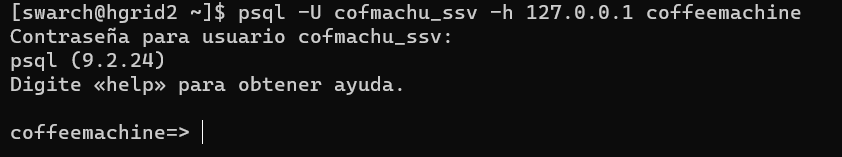
1. El archivo para crear las tablas lleva por nombre coffeemach-ddl.sql
2. El archivo para hacer los inserts lleva por nombre coffeemach-inserts.sql

# Conexión a la base de datos

Para comprobar la conexión a la base de datos, nos conectamos remotamente a la maquina que tiene desplegado la base de datos, en este caso a hgrid2 mediante SSH, una vez conectados procedemos a acceder a la base de datos de la siguiente manera

1. Nos conectamos a la base de datos con el siguiente comando (Ingresamos la contraseña **cofmachpwd**):

psql -U cofmachu\_ssv -h 127.0.0.1 coffeemachine



1. Una vez en la base de datos, aplicamos el siguiente comando para ver las tablas creadas en esta base de datos con el comando:

**\dt;**

**Texto

Descripción generada automáticamente con confianza media**

# Compilación archivos con Gradle

A continuación, mostraremos el proceso realizado para lograr compilar el proyecto con el comando gradle build para pasar solo los .jar y no todo el proyecto para ser compilado en cada máquina.

1. Agregamos una tarea en nuestro archivo build.gradle llamada copyToLib, la cual se encargará de juntar todos los ,jar necesarios en la carpeta libs

Texto

Descripción generada automáticamente

1. Una vez agregada la tarea anterior, construiremos el proyecto con el comando gradle build, el cual generara los ,jar necesarios para el despliegue en cada máquina de café.

Texto

Descripción generada automáticamente

1. Con el proyecto construido, ya tendremos la carpeta con todos los .jar, cabe recalcar que todos los .jar necesarios para que el proyecto funcione estarán en la carpeta libs correspondientemente (Veremos el ejemplo para el servidor central)

Texto

Descripción generada automáticamente

1. Procederemos a transformar la carpeta libs a .zip para ser enviada por scp
2. Cuando la carpeta libs este en .zip será enviada remotamente al equipo donde será desplegada a través de scp.
3. Posteriormente se deberá de pasar archivos adicionales para cada maquina que será explicado más adelante.

# Despliegue Servidor Central

Para desplegar y ejecutar nuestro servidor central se deberán seguir los siguientes pasos.

1. Se deberá de configurar nuestro archivo .cfg el cual tendrá los datos de la maquina donde se desplegará nuestro servidor, el archivo fue configurado de la siguiente manera. La IP mostrada en el archivo hace referencia a la maquina hgrid3 y se desplegara a través del puerto 12345.

A close-up of numbers

Description automatically generated with low confidence

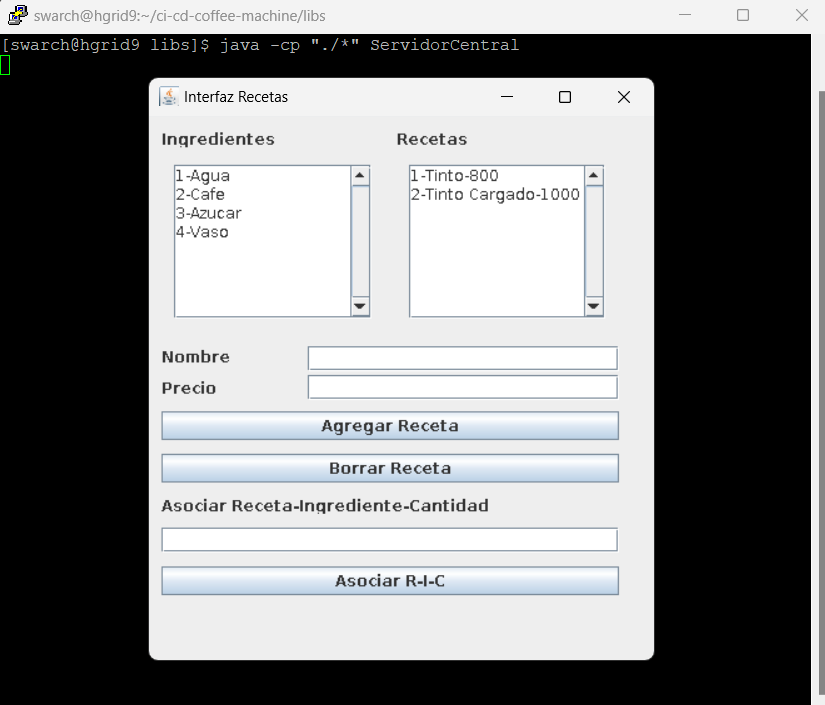
1. Se deberá de enviar el archivo libs.zip a través de scp a la ruta **ci-cd-coffee-machine** y posteriormente descomprimir la carpeta

A picture containing text, font, screenshot

Description automatically generated

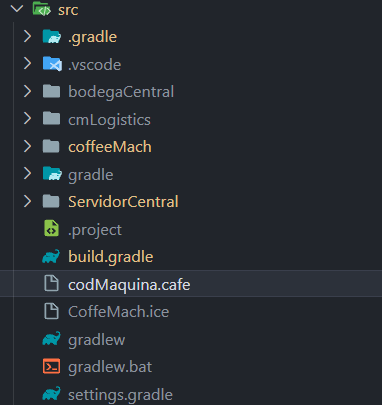
1. Una vez en la carpeta, se deberá de ejecutar el siguiente comando para lograr compilar y correr el archivo (Recordemos que deberá de ser ejecutado a través de PuTTY y con Xming para que se despliegue la interfaz gráfica).

java -cp "./\*" ServidorCentral



# Despliegue Máquina de café

El despliegue de esta máquina de café tiene una particularidad, debido a que se cuenta con un archivo .cafe, el cual contiene la identificación de la máquina frente al servidor central. Cada máquina deberá de tener un identificador único para ser ejecutada. El archivo es el siguiente (Para nuestra caso, el equipo hgrid10 tendrá como identificación en este archivo el ID 1 y el hgrid11 tendrá como identificación 2). Cabe recalcar que este archivo solo contiene un número.



Ahora sí, los pasos para desplegar las máquinas de café son los siguientes.

1. Se deberá de configurar nuestro archivo .cfg el cual tendrá los datos de la maquina donde se desplegará nuestras máquinas de café. Como se desplegarán dos máquinas, tendremos dos archivos .cfg con una configuración para cada maquina respectivamente.
   1. hgrid10

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

* 1. hgrid11

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

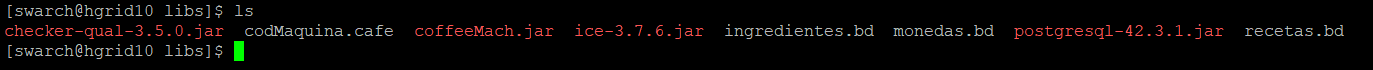
Interfaz de usuario gráfica, Texto

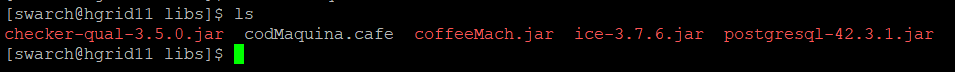
Descripción generada automáticamente

1. Se deberá de enviar el archivo libs.zip a través de scp a la ruta **ci-cd-coffee-machine** y posteriormente descomprimir la carpeta (Aplica para ambas hgrid)



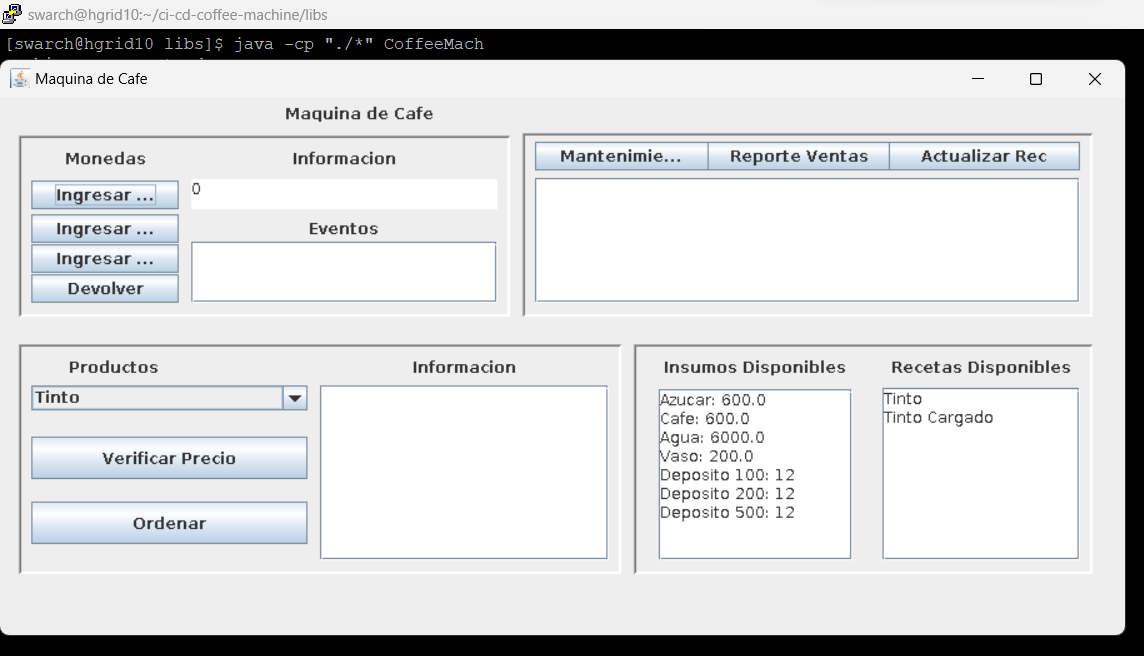
1. Una vez se tiene la carpeta descomprimida se deberá de entrar a la carpeta libs y copiar el archivo codMaquina.cafe para que la máquina pueda ser identificada. Aplica para ambas máquinas. (En la siguiente imagen se puede apreciar el archivo .cafe)





1. Con el archivo codMaquina.cafe en la carpeta libs, podemos ya ejecutar y desplegar nuestra máquina. Igualmente, que en el despliegue del servidor utilizaremos el mismo comando en PuTTY con Xming, con la diferencia del nombre de la clase, que en este caso sería CoffeeMach.

java -cp "./\*" CoffeeMach



Interfaz de usuario gráfica

Descripción generada automáticamente

# Despliegue Logística

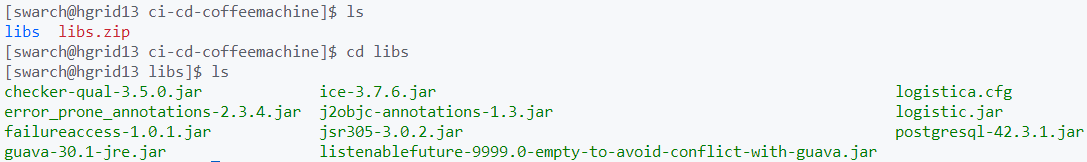
Para este caso en particular, debemos también desplegar en un nodo de procesamiento un componente que ejecute las tareas de la Logística para el sistema de la máquina de café. Es por esto por lo que:

1. Se debe configurar correctamente el archivo .cfg del componente de logística el cual contendrá la información necesaria para los servicios de intercomunicación. Por esto que la IP mostrada representa la máquina hgrid13 y se desplegará a través del puerto 12350.

A picture containing text, screenshot, font, line

Description automatically generated

1. Así mismo, deberá enviarse sus ejecutables comprimidos a través de scp a la ruta **ci-cd-coffee-machine** y después descomprimir.



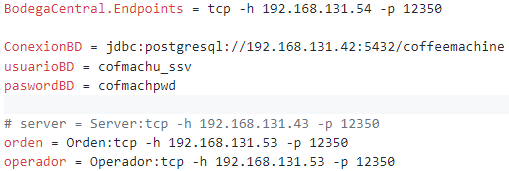
1. Una vez en la carpeta, se deberá de ejecutar el siguiente comando para lograr compilar y correr el archivo

java -cp "./\*" CmLogistics

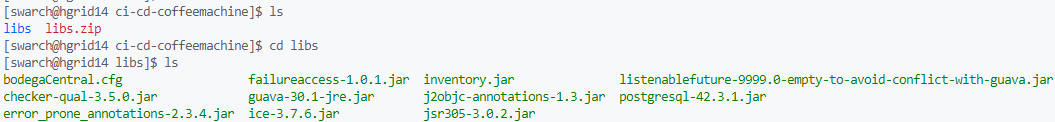
# Despliegue Bodega Central

Una vez ejecutado el componente de Logística, debe ejecutarse el componente de Bodega Central para que pueda usar el componente anteriormente descrito, Por tanto:

1. Configuramos el archivo .cfg de este componente ingresando la IP de la máquina en la que será desplegada para habilitar servicios de comunicación, en este caso como es la máquina hgrid14, su IP es 192.168.131.54 y el puerto por el que se desplegará es el 12350



1. Una vez compilado, comprimimos la carpeta /libs que se encuentra en /build y la enviamos a través de scp a la máquina hgrid14 al directorio **ci-cd-coffee-machine** y eventualmente la descomprimimos



1. Una vez en la carpeta, se deberá de ejecutar el siguiente comando para lograr correr los ejecutables

java -cp "./\*" BodegaCentral

# Implementación patrones de arquitectura

## **Despliegue Broker**

Para el despliegue del Broker se siguen los mismos pasos para los demás componentes, exceptuando que para este caso el Broker se desplegará en la máquina hgrid5 por medio del puerto 12346 y posteriormente se ejecuta a través del comando

java -cp "./\*" MessageBroker

## **Uso patrón broker**

El patrón broker nos permite implementar una capa extra de disponibilidad sin sobrecargar el servidor central gracias a que este ejecutará las peticiones más importantes de las alarmas sin que el servidor tenga que responder a todas individualmente, ya que todas estas peticiones pasarán primero por el broker ya que este está, por tanto, conectado al servidor central a través de una interfaz provista de Alarmas. Así mismo, gracias a este, podemos implementar otro patrón el cual se detallará a continuación.

## **Uso patrón reliable messaging con persistencia**

Gracias al broker se puede aprovechar este nodo de procesamiento para ejecutar tareas de envío de mensajes y de persistencia en local, de modo que, si una Alarma requiere ser enviada al servidor, pasará primero por el broker almacenándola en un archivo serializado que lleva por nombre **alarmas.bd** que será creada por la clase *AlarmaRepository.java* que implementa una interfaz *IRepositorio.java* e inmediatamente enviando la alarma al servidor como se puede apreciar aquí:

    @Override

    public void queueAlarma(Alarma am, CallbackPrx cb, Current current) {

        // Guarda en la capa de persistencia

        alarmaRepository

                .add(new model.Alarma(am.idAlarma, am.codMaquina, am.externalType, am.isTerminated, am.message, cb));

                sendNotifications(am, cb);

    }

Una vez el envío de la alarma tiene una respuesta por parte del servidor por medio de un “acknowledge”, se procede a eliminar la alarma almacenada en la capa de persistencia. Así, una vez la alarma es resuelta, se notifica al cliente por medio de un callback del estado de esta:

    @Override

    public boolean acknowledge(int code, int type, CallbackPrx cb, Current current) {

        model.Alarma am = alarmaRepository.getElement(code, type);

        if (am != null) {

            // Mark Delivered

            alarmaRepository.remove(am);

            cb.alarmConfirmation(code + " Fue confirmada y almacenada en la capa de persistencia");

            return true;

        }

        return false;

    }

## **Despliegue Proxy-cache**

Así como se ha venido mostrando a través de los anteriores despliegues, el proxy-cache tendrá su propio nodo de procesamiento a través del cual podrá entregar rápidamente las peticiones más frecuentes hechas por las máquinas de café. Por tanto, esta será desplegada en hgrid4 a través del puerto 12346. Tras haber compilado y enviado los artefactos a la máquina, procedemos a descomprimir y ejecutar a través del siguiente comando:

java -cp "./\*" ProxyCache

## **Uso patrón proxy-cache**

Este patrón permite un mayor rendimiento al momento de hacer consultas ya que implementa una estructura de Cache en HashMap que almacena en una clave (ingredientes, recetas y productos) el resultado retornado por las consultas previamente hechas por el cliente al servidor a través de este nodo de procesamiento de acuerdo con el tipo de consulta que se haga (ingredientes, recetas y productos) del siguiente modo:

    @Override

    public String[] consultarIngredientesProxy(Current current) {

        return consultar("ingredientes", "consultarIngredientes");

    }

    @Override

    public String[] consultarRecetasProxy(Current current) {

        return consultar("recetas", "consultarRecetas");

    }

    @Override

    public String[] consultarProductosProxy(Current current) {

        return consultar("productos", "consultarProductos");

    }

La máquina de café consulta al proxy por alguno de los tres (3) elementos previamente mostrados que requiera y el proxy cache procede a realizas la consulta en su implementación local de Cache y verifica que exista la consulta en memoria así:

    private String[] consultar(String key, String methodName) {

        if (cacheService.getElementFromCache(key) != null) {

            return cacheService.getElementFromCache(key);

        }

        try {

            Method method = recetaServicePrx.getClass().getMethod(methodName);

            String[] result = (String[]) method.invoke(recetaServicePrx);

            cacheService.putElementInCache(key, result);

            return cacheService.getElementFromCache(key);

        } catch (Exception e) {

            e.printStackTrace();

            return null;

        }

    }

De modo que si existe, retorna el valor almacenado en memoria y si no, pasa a realizar la consulta al servidor y vuelve y almacena en memoria para tenerlo a la mano para la siguiente consulta. Estos resultados los almacenará por 3 minutos de acuerdo con la implementación:

    private CacheService() {

        this.cache = CacheBuilder.newBuilder().maximumSize(10).expireAfterWrite(3, TimeUnit.MINUTES).build();

    }

## **Uso patrón observer**

El patrón observer permite asignar la responsabilidad de actualizar los datos requeridos del cliente al servidor adjuntando o suscribiendo el cliente a las recetas que el servidor administra a través del método *attach()* de modo que el cliente no deba estar constantemente consultando por los datos, consumiendo recursos que puede implementar en otras tareas:

    @Override

    public boolean attach(ObservablePrx proxy, Current current) {

        if (proxy == null)

            return false;

        observables.add(proxy);

        return true;

    }

Y a través del método *notifyAll()* actualiza a todos los proxys suscritos al servidor, y también, por medio del mismo método implementado en el proxy actualizará a todas las máquinas de café suscritas al proxy. Este método lo que hará es iterar a través de la lista de suscritos e irá invocando el método remoto *update()*:

    @Override

    public void \_notifyAll(Current current) {

        System.out.println("ObserverService.\_notifyAll - Notifying all proxy observers");

        for (int i = 0; i < observables.size(); i++) {

            observables.get(i).update(getUpdate("product", current));

        }

    }

Así el servidor, al invocar este método de cada uno de los proxy-cache desplegados, pasan a actualizar los datos que tienen en local así:

    @Override

    public void update(String[] data, Current current) {

        System.out.println("ObservableService.update - Server · Data: " + data);

        cacheService.putElementInCache("product", data);

        observerService.\_notifyAll(current);

    }

Y finalmente, el proxy-cache pasará a actualizar a todas las máquinas de café a través de sus propios métodos de actualización así:

    @Override

    public void update(String[] data, Current current) {

        System.out.println("ObservableService.update  - Proxy · Data: " + data);

        // controladorMQ.cargarRecetaMaquinas();

        controladorMQ.updateRecipes(data);

    }

Y almacena los datos en su persistencia local que lleva por nombre **recetas.bd**

# Uso del nmap

Finalmente, haremos el rastreo de los puertos para saber si se está ejecutando los device de manera correcta y como fueron configurados.

1. Base de datos (hgrid2)

Texto

Descripción generada automáticamente

Se puede observar que en el puerto 5432 se está ejecutando la base de datos postgres

1. Servidor Central (hgrid9)

Texto

Descripción generada automáticamente

Se puede observar que en el puerto 12345 esta corriendo nuestro servidor central.

1. Máquina de Café 1 (hgrid10)

Texto

Descripción generada automáticamente

1. Máquina de Café 2 (hgrid11)

Texto

Descripción generada automáticamente

# Diagrama de Deployment

