Sistemas Distribuidos Tarea # 1

Mauricio Solar msolar@inf.utfsm.cl
Ayudante: Humberto Farias Aroca humberto.farias@usm.cl

1. Objetivos

Familiarizarse con los problemas asociados a mantener un sistema distribuido. Usando para este fin librerías de Python como socket y entornos de virtualización de arquitecturas como Docker.

2. Generales

- El lenguaje a utilizar es Python.
- Se debe usar la librería socket de Python.
- Para cada actividad se deberá trabajar usando virtualización en base a containers, donde cada parte de la arquitectura, por ejemplo el servidor, deberá ser un container de Docker.
- En todas las actividades se deberá configurar un deployment automático de la arquitectura usando los archivos *docker-compose.yml* y *dockerfile*.

3. Actividades

Actividad 1: Cliente-servidor

En esta actividad se debe crear una aplicación cliente-servidor con las siguientes características:

Servidor:

- Se debe abrir un socket en el puerto 5000 y mantener el estado de espera de peticiones por parte del cliente.
- Cada vez que llega una petición debe registrarse el mensaje y IP esta en un *log.txt*. Además de enviar un mensaje al cliente que ha recibido correctamente la petición.

Cliente:

- Se debe crear una conexión al servidor (IP + Puerto).
- Se debe enviar un mensaje de saludos al servidor.
- El cliente deberá imprimir las respuestas que recibe del servidor y registrarlas en *respuestas.txt*

Se debe incluir en la carpeta de la actividad un archivo README con toda información que considere relevante, como por ejemplo las ruta a los archivos: *log.txt* y *respuestas.txt*.

Actividad 2: Simulando un sistema distribuido de datos

En sistemas de almacenamiento distribuido de datos como Hadoop, existe como parte de su arquitectura un nodo de coordinación (headnode, scheduler, master, etcc) y nodos de trabajo y/o almacenamiento (datanode, workers, slave, etc). En el headnode se mantiene un registro del status de cada datanode en base a consultar si esta operativo, este proceso en Hadoop se conoce como heartbeat. En esta actividad simularemos esta arquitectura de almacenamiento distribuido.

Características de la arquitectura

- Un container será el headnode, que tendrá como misión mantener actualizado el estatus del sistema de almacenamiento. Este proceso se debe realizar enviando un mensaje multicast a todos los datanodes preguntando si están operativos.
- La arquitectura tendrá 3 containers que serán del tipo datanode. Cada uno de estos deberá responder al headnode su estatus cuando éste les envíe el mensaje multicast.
- El headnode deberá llevar un registro de *hearbeat_server.txt* de los 3 datanode. Donde cada 5 segundos enviará el mensaje multicast para verificar si están operativos los datanode.

- Adicionalmente un container cliente debe poder enviar mensajes para que sean guardados al headnode. Este cliente sólo conocerá la IP y puerto del headnode.
- El Headnode seleccionará aleatoriamente un datanode y deberá enviar el mensaje a este datanode para ser guardado en el archivo *data.txt*.
- El datanode enviará un mensaje al headnode indicado que el registro fue correcto.
- El headnode deberá llevar el *registro_server.txt* en cual datanode guarda el mensaje del cliente. Y deberá informar al cliente que deberá llevar el *registro_cliente.txt* para registrar en que datanode se guardó su mensaje.

Consideraciones de la arquitectura

- Toda la arquitectura del sistema de almacenamiento se debe crear usando comando de docker a partir de los archivos de configuración: *docker-compose.yml* y *dockerfile*.
- El container del Cliente debe ser independiente de la arquitectura de almacenamiento y sólo conectarse al socket del headnode.
- Se debe incluir en la carpeta de la actividad un archivo *README* con toda información que considere relevante, como por ejemplo las ruta a los archivos: *hearbeat_server.txt*, *registro_server.txt* y *registro_cliente.txt*.

4. Consideraciones generales

- Se realizará una ayudantía con el fin de presentar y resolver dudas de la tarea.
- Consultas sobre la tarea se deben realizar en moodle o enviar un correo al ayudante (humberto.farias@usm.cl).

5. Reglas de entrega

- La tarea se realiza en **grupos de 2 personas**.
- La fecha de entrega es el día **2 de Octubre** donde se debe subir el link a un repositorio de GitHub con los archivos de las actividades.
- La corrección se realizará sobre los archivos presentes un repositorio de GitHub creado para este fin por cada estudiante.
- Se ejecutará el comando *docker-compose build* y *docker compose up*. Y la arquitectura cliente-servidor deberá desplegarse de forma autónoma.
- Cada día o fracción de atraso, se penalizará con un descuento de **10 puntos**.
- Copias se calificarán con **nota 0**.