

Practica programación de Sockets Telemática

**Abraham Miguel Lora Vargas
Camilo Galvis Montoya**

**Universidad EAFIT
2020**

Diseño del Producto (Snake Protocol)

Especificación del servicio:

Para este proyecto se realizó el diseño e implementación de un protocolo el cual denominamos **Snake Protocol** que describe la comunicación entre una aplicación cliente y un servidor, mediante el lenguaje de programación **JAVA**. De acuerdo a esto, la aplicación cliente envía peticiones al servidor con el fin de ser procesadas. Una vez este servidor recibe las peticiones, estas son procesadas y respondidas de acuerdo a los diferentes tipos de solicitudes mediante comandos ya debidamente especificados.

La implementación de este protocolo corre sobre sockets directamente. De acuerdo a esto, la aplicación cliente opera de manera interactiva, por lo tanto, cuenta con las siguientes características:

- Crear y eliminar buckets
- Lista los diferentes buckets que existen en el directorio
- Carga diferentes archivos desde el cliente hasta la carpeta o bucket definido por el cliente.
- Listar los diferentes archivos dentro de los buckets
- Descargar archivos del servidor al cliente
- Eliminar archivos del bucket

Para la aplicación del servidor cuenta con las siguientes características:

- La aplicación servidor recibe parámetros donde se especifica el path donde deben ser creador y alojados los buckets.
- Cuenta con la capacidad de manejar múltiples clientes de manera concurrente. Por lo tanto, cada cliente puede conectarse y desconectarse del servidor en el momento que lo considere y el servidor podrá seguir operando sin ningún tipo de restricción de manera transparente.
- El servidor cuenta con la capacidad de recibir comandos del cliente así se encuentre enviando un archivo o respuesta de éste
- Puede recibir y enviar diferentes archivos por parte de los clientes conectados al servidor

El presente documento será dividido en dos apartados denominados Cliente y Servidor, donde se explicarán cosas relacionadas con cada uno.

Cliente

El cliente de Snake Protocol cuenta con un sistema de lectura de comandos por medio de un CLI, estos comandos son principalmente funciones del cliente y se usan para ejecutar las comunicaciones principales entre el servidor por medio del protocolo. El procesamiento de los comandos se ejecuta en base a la sintaxis definida por el operador de la línea de comandos quien puede consultar todos los comandos disponibles introduciendo el comando de ayuda **help**.

Durante el arranque del cliente, este intentará conectarse al servidor como primera acción, si no es posible establecer la conexión lo reintentará cada 5 segundos hasta que se finalice el proceso o consiga conectarse. A su vez una vez se logra la conexión, el cliente le indica al servidor que la conexión es de tipo **client**, de esta manera se produce un canal exclusivo para la lectura de comandos del protocolo, esta denominación cobra sentido durante la transferencia de archivos, ya que por cada archivo que se transfiere (tanto de descarga como de subida), se produce una conexión secundaria desde el cliente al servidor, donde se le indica al servidor que es de tipo **files**, logrando así producir un sub-canal exclusivo para la transferencia de un archivo específico donde una vez es transferido el sub-canal es cerrado y el proceso principal de tipo **client** se mantiene operando normalmente. Esta especificación de canales le permite al protocolo seguir ejecutando comandos de un cliente independientemente de que este se encuentre subiendo o bajando archivos. Finalmente, los clientes son concurrentes ya que se produce un hilo desde el lado del servidor por cada conexión de clientes (independientemente del tipo de conexión), permitiendo al sistema operar con múltiples clientes mientras estos suben o bajan múltiples archivos al mismo tiempo sin ningún tipo de problema.

Comandos del Cliente:

Para la ejecución de comandos desde el lado del cliente es obligatorio utilizar los prefijos de argumentos [-a, -b, -c...] Con la finalidad de darle argumentos validos que requiera un comando en específico.

- **help** – Permite ver una lista con todos los comandos del cliente. Ejemplo de uso:
help
- **developers** – Permite ver una lista con los desarrolladores de Snake Protocol.
Ejemplo de uso: developers
- **load -p <ruta-de-archivo-local> -b <bucket-remoto> -f <nombre-de-archivo-remoto>**
- Permite subir un archivo local a un bucket del servidor indicando el bucket y el nombre del archivo así como también la ruta del archivo local. *Ejemplo de uso:*
load -p C:/imagen.jpg -b bucket01 -f imagen.jpg
- **download -d <ruta-local-de-destino> -b <bucket-remoto> -f <nombre-de-archivo-remoto>**
- Permite descargar un archivo remoto de un bucket indicando el bucket, el nombre del archivo y la ruta de destino donde se guardará el archivo (incluyendo el nombre). Ejemplo de uso: *download -d C:/imagen1.jpg -b bucket01 -f imagen.jpg*

- `newb -n <nombre-del-nuevo-bucket>` – Permite crear un nuevo bucket remoto. *Ejemplo de uso: `newb -n bucket01`*
- `deleteb -n <nombre-del-bucket-a-borrar>` - Permite borrar un bucket existente en el servidor con todos sus contenidos. *Ejemplo de uso: `deleteb -n bucket01`*
- `deletef -b <nombre-del-bucket> -n <nombre-del-archivo-a-borrar>` - Permite borrar un archivo dentro de un bucket. *Ejemplo de uso: `deletef -b bucket01 -n imagen.jpg`*
- `buckets` – Permite listar todas las buckets que fueron creadas en el servidor. *Ejemplo de uso: `buckets`*
- `files -b <nombre-del-bucket>` - Permite listar todos los archivos dentro de un bucket. *Ejemplo de uso: `files -b bucket01`*
- `uptime` – Permite saber cuanto tiempo el servidor lleva encendido. *Ejemplo de uso: `uptime`*
- `close` – Cierra la conexión (de manera correcta) con el servidor y a su vez finaliza el proceso del cliente. *Ejemplo de uso: `close`*

Los comandos que envíen y reciban datos del servidor siempre recibirán una respuesta predefinida la cual consta de un código de mensaje (número entero) y de un mensaje en formato UTF, estos mensajes son definidos en el apartado del servidor y cuando son recibidos por el cliente son imprimidos en la interfaz de comandos.

Ejecución de la Instancia en modo Cliente

Para ejecutar una instancia de Snake en modo cliente se debe tomar el JAR (Java Archive), con todo el código compilado y ejecutarlo vía consola introduciendo el siguiente comando:

```
java -jar snake-server-1.0-SNAPSHOT.jar
```

Si se desea se pueden especificar hasta 2 argumentos adicionales (totalmente opcionales) para el cliente de Snake los cuales son el puerto y el host del servidor, de esta manera se logra una conexión remota de manera correcta con el compilado. Ejemplo de uso con los 2 argumentos:

```
java -jar snake-server-1.0-SNAPSHOT.jar -p 4000 -h 51.15.209.191
```

Si no se especifica el puerto se tomará el puerto por defecto **4000** y si no se especifica host se tomará el host por defecto **localhost**.

Nota: Se debe realizar la ejecución utilizando Java 14 de lo contrario puede no funcionar.

Servidor

El servidor de Snake Protocol lee directamente mensajes de sockets de los clientes donde al conectarse un cliente se clasifican entre clientes de tipo **client** o clientes de tipo **files** (se menciona esto en el apartado del cliente dentro de este documento). Asimismo, cuando el servidor es inicializado este crea un servidor de sockets en el puerto especificado a la espera de nuevas conexiones donde cada conexión independientemente del tipo se ejecuta en un hilo aparte. El servidor solo entiende mensajes propios del protocolo y envía como respuesta a los clientes códigos y mensajes denominados “mensajes de respuesta”. A continuación, se especificará una breve descripción de ambos tipos de mensajes y sus acciones dentro del sistema servidor:

Mensajes del Protocolo (Clientes tipo CLIENT):

La especificación de estos mensajes se hace a manera de comunicación continua, es decir para cada tipo de mensaje se especifican los diferentes tipos de datos que se reciben por parte del cliente.

- **NEWB** – Crear un nuevo bucket.
 1. bucket (UTF) – Nombre del nuevo bucket.

Posibles respuestas:

 - **BUCKET_ALREADY_EXISTS** – Si el bucket ya existe.
 - **BUCKET_CREATED** – Si se logra crear el bucket correctamente.
 - **INTERNAL_ERROR** – En caso de que el servidor sea incapaz de crear la carpeta (por falta de permisos).
- **DELETEB** – Borrar un bucket existente.
 1. bucket (UTF) – Nombre del bucket.

Posibles respuestas:

 - **BUCKET_NOT_EXISTS** – Si el bucket no existe.
 - **BUCKET_DELETED** – Si se logra borrar el bucket correctamente.
 - **INTERNAL_ERROR** – Si ocurre un error inesperado.
- **DELETF** – Borrar un archivo en un bucket.
 1. bucket (UTF) – Nombre del bucket.
 2. file (UTF) – Nombre del archivo.

Posibles respuestas:

 - **BUCKET_NOT_EXISTS** – Si el bucket no existe.
 - **FILE_NOT_EXISTS** – Si el archivo no existe.
 - **FILE_DELETED** – Si se logra borrar el archivo correctamente.
 - **INTERNAL_ERROR** – En caso de que el servidor sea incapaz de borrar el archivo (por falta de permisos).
- **BUCKETS** – Lista todas las buckets.

Posibles respuestas:

 - **BUCKETS** – Mensaje de comprobación para luego enviar la lista de buckets.

- **BUCKETS[UTF]** – Mensaje con todas las buckets separadas por comas (,).
- **UPTIME** – Enviar el tiempo en el que el servidor lleva encendido.
Posibles respuestas:
 - **UPTIME** – Mensaje de comprobación para luego enviar el tiempo de encendido.
 - **UPTIME[UTF]** – Mensaje con el tiempo de encendido en formato HH:MM:SS.
- **CLOSE** – Cerrar una conexión con el cliente de manera correcta.
Posibles respuestas:
 - **CLOSE** – Mensaje indicando que la conexión socket será cerrada en breve desde el lado del servidor.
- **FILES** – Envía la lista de archivos de un bucket.
 1. bucket (UTF) – Nombre del bucket.Posibles respuestas:
 - **BUCKET_NOT_EXISTS** – Si el bucket no existe.
 - **FILES** – Mensaje de comprobación para luego enviar la lista de archivos.
 - **FILES[UTF]** – Mensaje con todos los archivos del bucket separados por comas (,).
- **LOAD** – Envía información indicando que el cliente quiere iniciar una carga de un archivo en un bucket.
 1. bucket (UTF) – Nombre del bucket.
 2. fileName (UTF) – Nombre del archivo.Posibles respuestas:
 - **BUCKET_NOT_EXISTS** – Si el bucket no existe.
 - **FILE_ALREADY_EXISTS** – Si el archivo que se desea subir ya se encuentra en el bucket.
 - **RECEIVING_FILE** – Este mensaje le indica al cliente que puede iniciar un sub-canal de tipo **files**, con subtipo 1 (Upload), para cargar el archivo en el servidor. (Véase mensajes del Protocolo con clientes tipo Files)
 - **INTERNAL_ERROR** – En caso de que el servidor sea incapaz de crear el archivo (por falta de permisos).
- **DOWNLOAD** – Envía información indicando que el cliente quiere iniciar una descarga de un archivo en un bucket.
 1. bucket (UTF) – Nombre del bucket.
 2. fileName (UTF) – Nombre del archivo.Posibles respuestas:
 - **BUCKET_NOT_EXISTS** – Si el bucket no existe.
 - **FILE_NOT_EXISTS** – Si el archivo que se desea descargar no existe en el bucket.
 - **FILE_FOUND** – Este mensaje le indica al cliente que puede iniciar un sub-canal de tipo **files**, con subtipo 0 (Download), para descargar el archivo del servidor. (Véase mensajes del Protocolo con clientes tipo Files)
 - **INTERNAL_ERROR** – En caso de que el servidor sea incapaz de crear el archivo (por falta de permisos).

Mensajes del Protocolo (Clientes tipo FILES):

La especificación de estos mensajes se hace a manera de comunicación continua, es decir para cada tipo de mensaje se especifican los diferentes tipos de datos que se reciben por parte del cliente. Para el tipo de mensaje se utiliza el numero de Protocolo (1 para carga, 0 para descarga), este sistema no genera respuesta para cargas, ya que al ser un sub-canal solo se utiliza para la transferencia de archivos, sin embargo, para las descargas se reciben los datos del archivo y su tamaño, los mensajes de respuesta son enviados solo por el protocolo de clientes tipo **client** especificado arriba.

- **UPLOAD (1)** – Subir un nuevo archivo al servidor.
 1. bucket (UTF) – Nombre del nuevo bucket.
 2. fileName (UTF) – Nombre del archivo.
 3. size (long) – Tamaño del archivo a cargar.
 4. bytes (bytes[]) – Stream de bytes del archivo que se cargará.
- **DOWNLOAD (0)** – Descargar un archivo del servidor.
 1. bucket (UTF) – Nombre del nuevo bucket.
 2. fileName (UTF) – Nombre del archivo que se quiere descargar.

Posibles respuestas:

- **SIZE[LONG]** – Tamaño del archivo que se descargará.
- **BYTES[BYTES[]]** – Stream de bytes del archivo que se descargará.

Listado de Mensajes de Respuesta del Protocolo:

Para el listado de mensajes de respuesta se manejan intervalos, donde del 200 al 299 se encuentran todos los mensajes que indican un buen procedimiento (tanto del cliente como del servidor), desde el 800 al 899 se sitúan todos los posibles mensajes de error. Y luego los mensajes independientes ya sea de errores internos o mal procedimiento durante la ejecución de comandos del protocolo vienen del 900 en adelante.

- BUCKET_CREATED (200)
- BUCKET_DELETED (210)
- FILE_DELETED (220)
- BUCKETS (225)
- FILES (230)
- UPTIME (235)
- RECEIVING_FILE (240)
- FILE_FOUND (245)
- CLOSE (250)
- BUCKET_NOT_EXISTS (805)
- BUCKET_ALREADY_EXISTS (810)
- FILE_ALREADY_EXISTS (815)
- FILE_NOT_EXISTS (820)
- INTERNAL_ERROR (900)
- BAD_CMD (980)

Ejecución de la Instancia en modo Servidor

Para ejecutar una instancia de Snake en modo servidor se debe tomar el JAR (Java Archive), con todo el código compilado y ejecutarlo vía consola introduciendo el siguiente comando:

```
java -jar snake-server-1.0-SNAPSHOT.jar --server
```

Si se desea se pueden especificar hasta 2 argumentos adicionales (totalmente opcionales) para el servidor de Snake los cuales son el puerto y el path de archivos, de esta manera se logra ejecutar un servidor con diferentes opciones a las que trae éste por defecto. Ejemplo de uso con los 2 argumentos:

```
java -jar snake-server-1.0-SNAPSHOT.jar --server -p 4000 -f "C:/server_data"
```

Si no se especifica el puerto se tomará el puerto por defecto **4000** y si no se especifica la ruta de archivos se tomará el path relativo y una carpeta llamada server, es decir: **./server**.

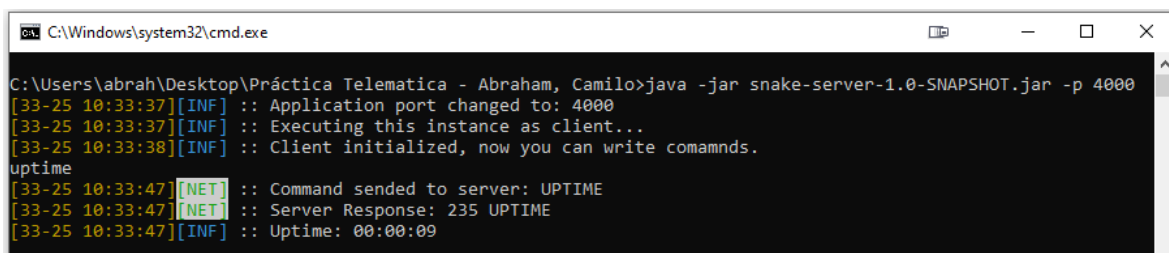
Nota: Se debe realizar la ejecución utilizando Java 14 de lo contrario puede no funcionar.

Servidor en la Nube Funcional:

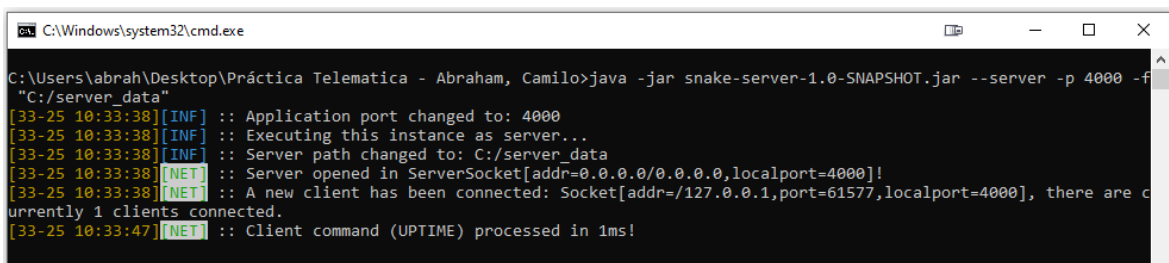
Para este proyecto hemos implementado un servidor en la nube el cual está corriendo en el puerto **4000** del servidor con IP **51.15.209.191** (el cual es un servidor de Linux con Ubuntu 18.04 hosteado 24/7 horas), y posee la última versión de Snake Protocol. Este servidor está diseñado para realizar pruebas con el sistema conectando un cliente local con dicho servidor remoto ejecutando el cliente con el comando de ejemplo que está en la sección Cliente. El servidor estará disponible durante varios días entregada esta práctica.

Imágenes del Sistema Funcionando:

Prueba Uptime:



```
C:\Windows\system32\cmd.exe
C:\Users\abrah\Desktop\Práctica Telemática - Abraham, Camilo>java -jar snake-server-1.0-SNAPSHOT.jar -p 4000
[33-25 10:33:37][INF] :: Application port changed to: 4000
[33-25 10:33:37][INF] :: Executing this instance as client...
[33-25 10:33:38][INF] :: Client initialized, now you can write comamnds.
uptime
[33-25 10:33:47][NET] :: Command sent to server: UPTIME
[33-25 10:33:47][NET] :: Server Response: 235 UPTIME
[33-25 10:33:47][INF] :: Uptime: 00:00:09
```



```
C:\Windows\system32\cmd.exe
C:\Users\abrah\Desktop\Práctica Telemática - Abraham, Camilo>java -jar snake-server-1.0-SNAPSHOT.jar --server -p 4000 -f
"C:/server_data"
[33-25 10:33:38][INF] :: Application port changed to: 4000
[33-25 10:33:38][INF] :: Executing this instance as server...
[33-25 10:33:38][INF] :: Server path changed to: C:/server_data
[33-25 10:33:38][NET] :: Server opened in ServerSocket[addr=0.0.0.0/0.0.0.0,localport=4000]!
[33-25 10:33:38][NET] :: A new client has been connected: Socket[addr=/127.0.0.1,port=61577,localport=4000], there are c
urrently 1 clients connected.
[33-25 10:33:47][NET] :: Client command (UPTIME) processed in 1ms!
```


Prueba Variada (deleteb, newb, load, download):

```
C:\Windows\system32\cmd.exe

C:\Users\abrah\Desktop\Práctica Telemática - Abraham, Camilo>java -jar snake-server-1.0-SNAPSHOT.jar -p 4000
[35-25 10:35:45][INF] :: Application port changed to: 4000
[35-25 10:35:45][INF] :: Executing this instance as client...
[35-25 10:35:45][INF] :: Client initialized, now you can write comamnds.
deleteb -n bucket01
[35-25 10:35:55][NET] :: Command sent to server: DELETEDB
[35-25 10:35:55][NET] :: Server Response: 210 BUCKET_DELETED
newb -n mibucket
[36-25 10:36:14][NET] :: Command sent to server: NEWB
[36-25 10:36:14][NET] :: Server Response: 200 BUCKET_CREATED
load -p C:/camilo.txt -b bucket01 -f camilo.txt
[36-25 10:36:22][INF] :: File Found: C:\camilo.txt [Size: 17 bytes]
[36-25 10:36:22][NET] :: Server Response: 805 BUCKET_NOT_EXISTS
load -p C:/camilo.txt -b bucket08 -f camilo.txt
[36-25 10:36:30][INF] :: File Found: C:\camilo.txt [Size: 17 bytes]
[36-25 10:36:30][NET] :: Server Response: 805 BUCKET_NOT_EXISTS
load -p C:/camilo.txt -b mibucket -f camilo.txt
[36-25 10:36:43][INF] :: File Found: C:\camilo.txt [Size: 17 bytes]
[36-25 10:36:43][NET] :: Command sent to server: LOAD
[36-25 10:36:43][INF] :: Sending file to the server...
[36-25 10:36:43][NET] :: File sent successfully!
download -d C:/camilo_otro.txt -b mibucket -f camilo.txt
[37-25 10:37:05][INF] :: Downloading file from the server...
[37-25 10:37:05][INF] :: Downloaded file: C:\camilo_otro.txt [Size: 17 bytes]
```

```
C:\Windows\system32\cmd.exe

C:\Users\abrah\Desktop\Práctica Telemática - Abraham, Camilo>java -jar snake-server-1.0-SNAPSHOT.jar --server -p 4000 -f
"C:/server_data"
[35-25 10:35:44][INF] :: Application port changed to: 4000
[35-25 10:35:44][INF] :: Executing this instance as server...
[35-25 10:35:44][INF] :: Server path changed to: C:/server_data
[35-25 10:35:44][NET] :: Server opened in ServerSocket[addr=0.0.0.0/0.0.0.0,localport=4000]!
[35-25 10:35:45][NET] :: A new client has been connected: Socket[addr=/127.0.0.1,port=61592,localport=4000], there are c
currently 1 clients connected.
[35-25 10:35:55][NET] :: Client command (DELETEDB) processed in 11ms!
[36-25 10:36:14][NET] :: Client command (NEWB) processed in 0ms!
[36-25 10:36:22][NET] :: Client command (LOAD) processed in 0ms!
[36-25 10:36:30][NET] :: Client command (LOAD) processed in 0ms!
[36-25 10:36:43][NET] :: Client command (LOAD) processed in 0ms!
[36-25 10:36:43][NET] :: A sub-channel was generated for file transfer of type (1), connection: Socket[addr=/127.0.0.1,p
ort=61594,localport=4000]
[36-25 10:36:43][INF] :: Sub-channel information [File: camilo.txt, Bucket: mibucket, Type: Upload]
[36-25 10:36:43][INF] :: New file uploaded from client: C:\server_data\mibucket\camilo.txt, closing sub-channel...
[37-25 10:37:05][NET] :: Client command (DOWNLOAD) processed in 0ms!
[37-25 10:37:05][NET] :: A sub-channel was generated for file transfer of type (0), connection: Socket[addr=/127.0.0.1,p
ort=61595,localport=4000]
[37-25 10:37:05][INF] :: Sub-channel information [File: camilo.txt, Bucket: mibucket, Type: Download]
[37-25 10:37:05][INF] :: File camilo.txt sended to client, closing sub-channel...
```

Diagrama de Secuencias:

Para ver el diagrama de secuencias lo puede encontrar en el repositorio de github como diagrama.png.

Link repositorio de github: <https://github.com/ToxicSSJ/snake-server>