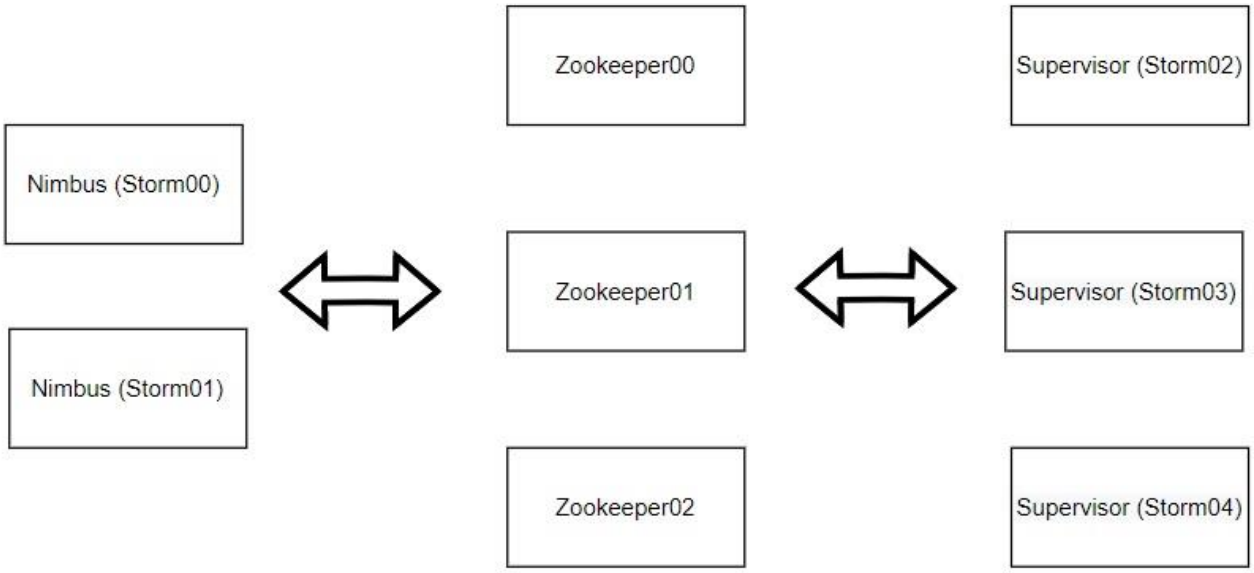
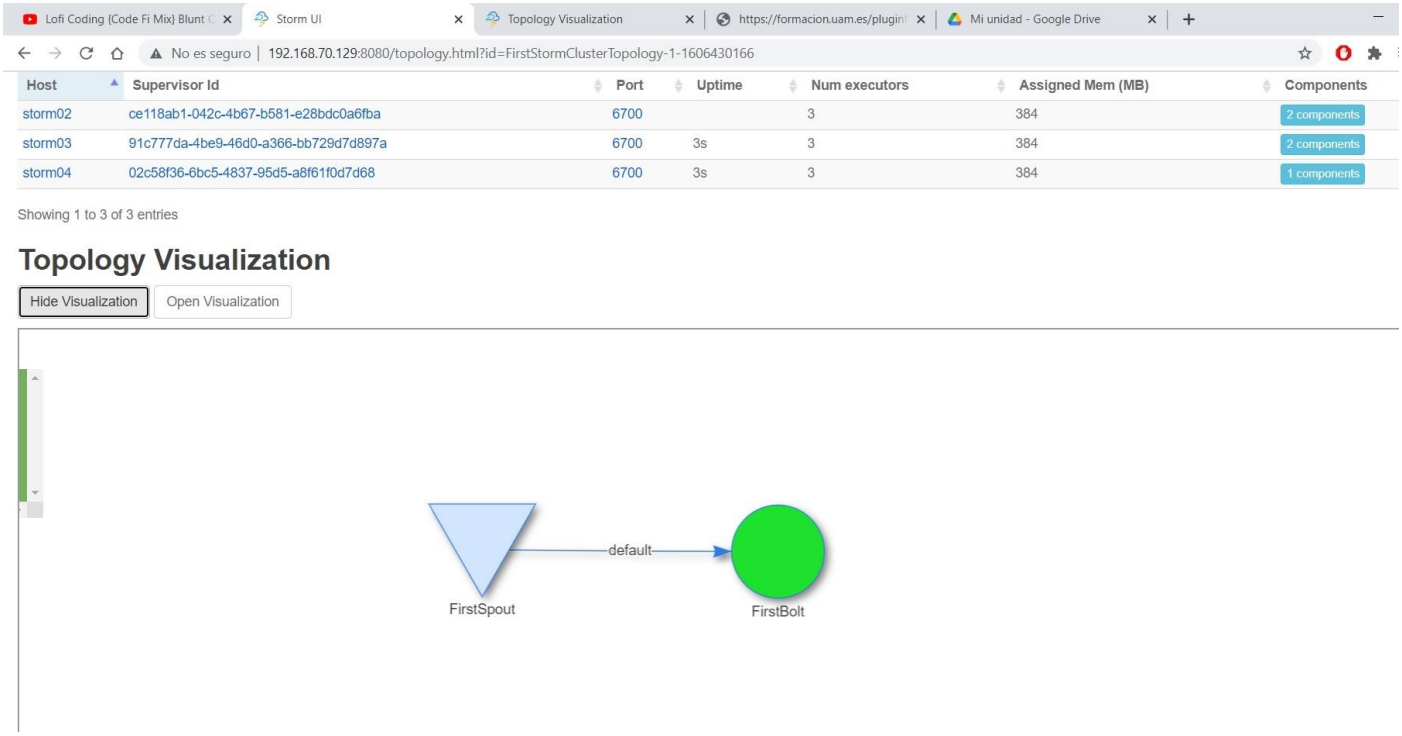
APACHE STORM

LUCIAN IACOB

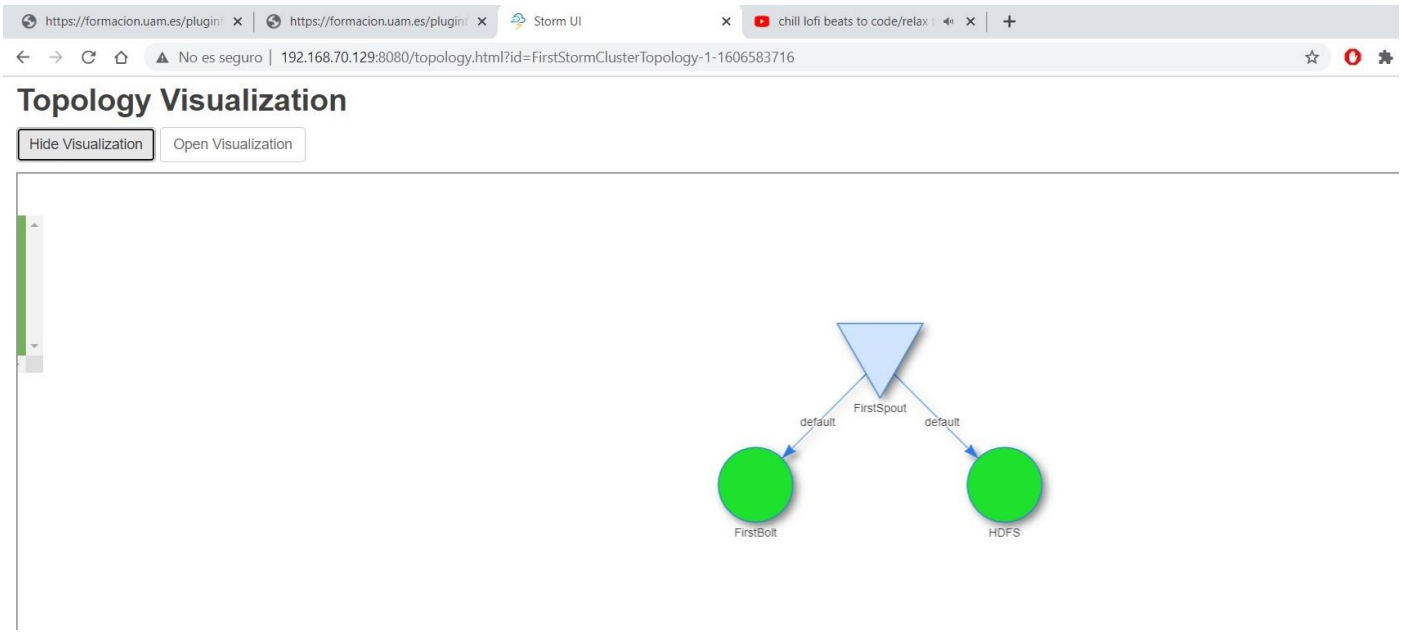
➔ **clusterStorm.jpg**



➔ **Lab2.jpg**



➔ **Lab3.jpg**



➔ **Questions**

1. **What is a Topology, Spout and Bolt in Apache Storm?**

   Una topología es una abstracción sobre la computación que tiene que hacer el cluster. Consiste en un grafo dirigido (los elementos se unen mediante flechas que indican direccionalidad) y acíclico (no hay caminos que te lleven al mismo punto desde el que partes).
   Es lo que lanzamos en este tipo de clústeres, al igual que en Hadoop lanzamos tareas MapReduce, en Apache Storm lanzado topologías.

   En Apache Storm tratamos streams de tuplas de datos mediante dos transformaciones primitivas: los spouts y los bolts:

   - Los spouts son el origen del stream, básicamente de donde salen los datos como por ejemplo conectarse a la API de Twitter y empezar a emitir los twits captados.

   - Los bolts básicamente reciben el stream, realizan alguna tarea de procesamiento (lanzan funciones, filtran tuplas, hacen joins, etc.) y, posiblemente, emiten nuevos streams.

   Estos dos son los elementos que vemos en las topologías que lanzamos en Apache Storm, además de las flechas que indiquen su relación así como el flujo de datos.

2. **What types of processing can I execute with Storm regarding message delivery (Guaranteed message processing)? Could you provide a briefly description and example for each of them?**

   Dentro de las llamadas de procesamiento remoto (RPC) encontramos 3 tipos de semánticas:

   - “ Tal vez ”

El procedimiento remoto puede ejecutarse una vez o ninguna, por lo que el cliente puede recibir una respuesta o ninguna.
Básicamente, el cliente envía una petición y se queda a la espera un tiempo determinado. Si en ese tiempo no llega ninguna respuesta se continúa la ejecución con la siguiente petición.
En este caso, no hay ningún tipo de realimentación en caso de fallo, por lo que solo es admisible en situaciones en las que se toleren pérdidas o se permitan destiempos.

The message may be lost

Un ejemplo que se me ocurre es en laboratorios de observación astronómica como el LIGO, que están continuamente captando datos y procesándolos. En este caso, que ocurra un fallo por el que se pierda un dato y su procesamiento no debería ser tan importante puesto que es prácticamente imposible que en el paso de una ventana de tiempo a otra suceda un cambio inexplicable y se pase de un sistema físico a otro totalmente distinto puesto que estamos tratando con escalas de tiempo cosmológicas.

- " Al menos una vez"

Aquí la orden se ejecuta una o más veces, pero nunca cero. Lo que ocurre es que el cliente manda una petición y se queda a la espera. En caso de no recibir el acknowledge en el periodo de tiempo designado, se repite la orden sin ningún tipo de filtrado de si es la original o se ha mandado por haber fallado el primer intento. Por ello podemos tener situaciones en las que un procedimiento en remoto se ejecuta veces repetidas.
Esta situación solo puede ser aplicable para el caso de operaciones idempotentes (por muchas veces que ejecutemos la orden el resultado siempre va a ser el mismo, da igual una o diez)

The message may be duplicated

Se me ocurre con ejemplo una funcionalidad que cuente los diferentes usuarios que comentan en directo alguna retransmisión (no el número de personas comentando). En este caso que se registre más de una vez el mismo comentario da igual porque estamos seleccionando miembros únicos.

- " Exactamente una vez "

En este caso la orden se ejecuta una y solo una vez. Se asemeja a al menos una vez pero en esta caso se lleva un registro de qué orden ha habido que mandar más de una vez para desechar repeticiones en las peticiones.

The message is delivered exactly once

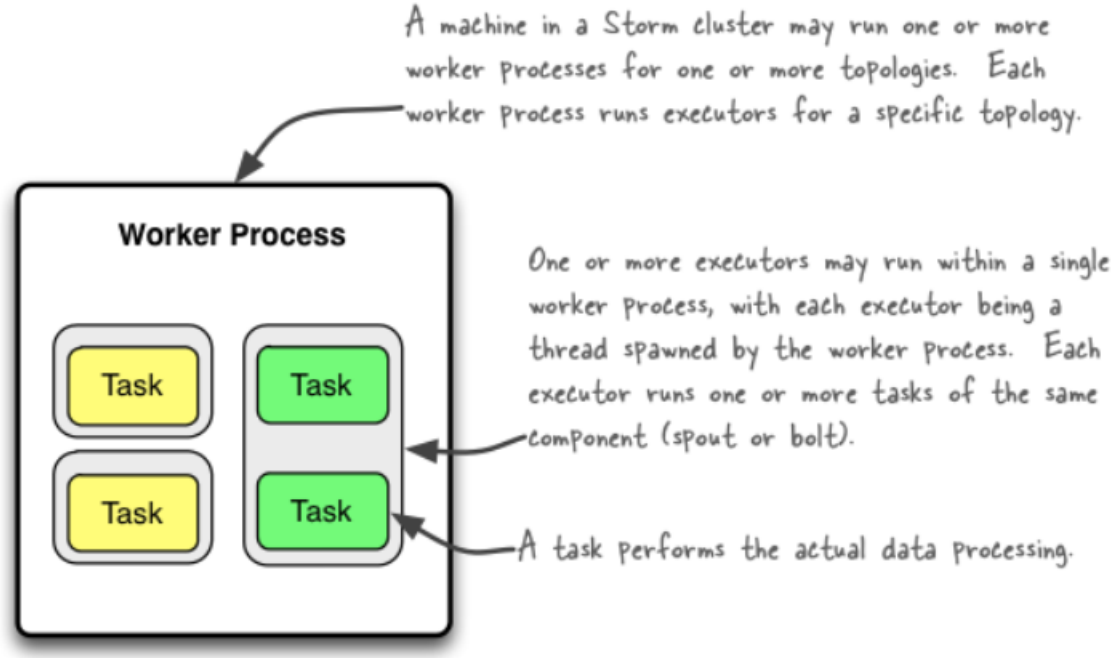
Como ejemplo propondría el proceso por el que un banco recibe órdenes de pago de manera instantánea (Bizum por ejemplo). Lo que se busca es garantía de que el pago se realiza una y solo una vez.

3. **Which sentence is the right one: -> A task runs executors or A executor runs tasks?**

   La correcta es “A executor runs tasks “.

   En Apache Storm tenemos los worker process, que son procesos lanzados por las máquinas. Estos realizan una parte de la topología que se le ha asignado, para ello crean hilos de ejecución que son los executors. Pero estos no son los que hacen la tarea directamente sino que lanzan una o más tasks (para un mismo componente) y ya la task es la que realiza el data processing.

   Adjunto una imagen obtenida de la propia documentación de Apache Storm (http://storm.apache.org/releases/current/Understanding-the-parallelism-of-a-Storm-topology.html) que lo explica todo mucho mejor.



4. **Which command should I execute to change the parallelism of a running topology?**

   Para cambiar el paralelismo de una topología en marcha debemos de recurrir al rebalanceo. El comando utilizado sería:

   $ storm rebalance mi_topología –n X –e bolt_cambiado=Y -e spout_cambiado=Z

   En el cual usamos la bandera n para cambiar el número de procesos worker lanzados a X y cambiamos el número de ejecutores usado por el bolt a Y, y los utilizados por el spout a Z

5. **If I have a single high-performance node in my cluster, which scheduler I should use?**

   Si solo tengo una máquina yo usaría un ResourceAwareScheduler. Nuestros medios son limitados aunque el ordenador sea high-performance pues solo es una máquina. Veo más conveniente este tipo de scheduler porque así será como podamos exprimir al máximo el potencial que tenga nuestro nodo.
   Otra opción viable supongo que sería un custom scheduler en el que poder especificar todo a nuestro gusto ya pero sin duda isolation y multitenant quedan descartados porque están orientados a clústeres de mayor tamaño.