SISTEMAS DISTRIBUIDOS

Práctica Extraordinaria

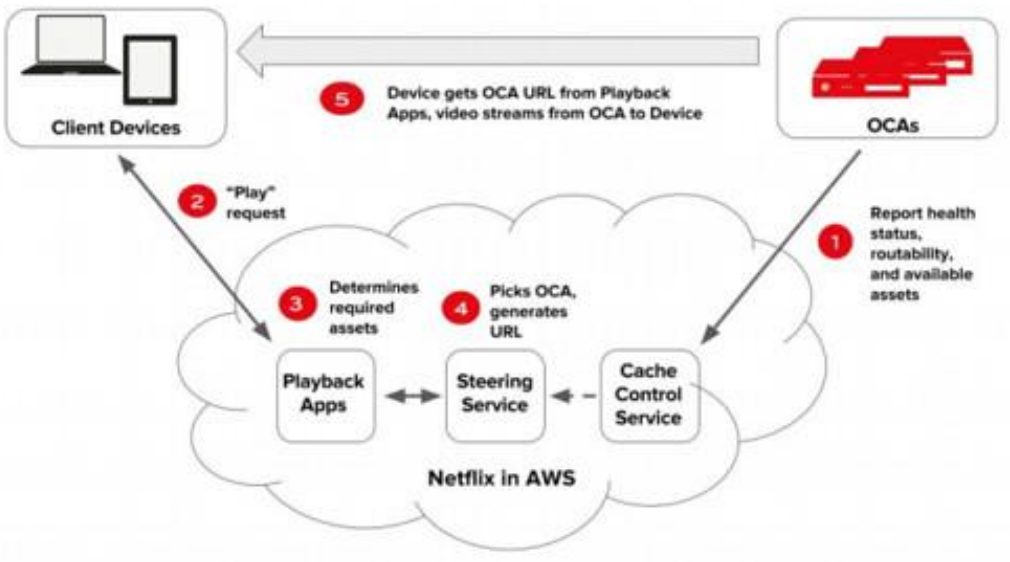
IceFlix: diseño de microservicios

El objetivo principal del proyecto es **desarrollar un sistema distribuido basado en microservicios.** Para ello se tomará como modelo la popular plataforma de *streaming NetFlix* para crear una pequeña plataforma de *streaming* bajo demanda, utilizando algunos de los servicios disponibles en **ZeroC ICE**. Los objetivos principales de la práctica son:

- ✓ Familiarizarse con la invocación a métodos remotos.
- ✓ Control de eventos.
- ✓ Diseñar algoritmos tolerantes a fallos comunes en sistemas distribuidos.
- ✓ Fomentar el trabajo en equipos multidisciplinares.

Introducción.

Conocida por todos es la plataforma de *streaming* bajo demanda *NetFlix,* lo que quizás no es tan popular es la gran competencia técnica que existe detrás, el increíble sistema distribuido que soporta el uso simultáneo de la plataforma por millones de usuarios en todo el mundo. No todos los detalles de su funcionamiento han sido publicados por la empresa, sin embargo, si los suficientes para poder hacernos una idea general de su funcionamiento.



Una colección de microservicios soporta el funcionamiento de la plataforma: autenticación, facturación, catalogado, recodificación, análisis y *profiling* de clientes (*big data*), etc. Estos microservicios se ejecutan en varias regiones de AWS. Por otro lado, Netflix dispone de sus propios servidores físicos (*OCA's*) que almacenan los archivos de vídeo y están diseñados específicamente para ofrecer un gran rendimiento en tareas de *streaming*.

Obviamente no pretenderemos crear una arquitectura con semejante complejidad, pero sí intentaremos implementar una pequeña maqueta con una serie de microservicios que cooperarán entre sí para ofrecer video bajo demanda, de una forma similar a como lo hace la conocida plataforma. No vamos a utilizar mecanismos de caché ni a implementar muchos de los servicios disponibles en la plataforma, nos centraremos en: autenticación, catalogado de medios y *streaming* RTSP.

1. Estructura general

1.1 El microservicio como base.

Nuestra unidad lógica básica con la que vamos a trabajar es el *microservicio*. Cada microservicio consistirá en un programa escrito en Python 3 que presentará una interfaz de objeto Ice con la que podrán interactuar otros microservicios y/o aplicaciones de cliente. Además, debe cumplir los siguientes requisitos:

- Dispone de un identificador único
- Debe anunciarse para que otros microservicios puedan interactuar con él
- Tiene que poder encontrar a los otros microservicios en ejecución
- Cuando un microservicio necesite ejecutar un método implementado en otro microservicio, deberá poder usar cualquiera de los que ha encontrado
- En caso de fallo en un microservicio, se eliminará como candidato a ser utilizado

El anuncio de los microservicios se realizará utilizando un canal de eventos de *IceStorm* exclusivo a tal fin. Cada microservicio debe implementar el mecanismo para comprobar el estado del resto de microservicios conocidos por él para así poder eliminar aquellos que hayan dejado de funcionar.

Cuando un microservicio deba realizar una llamada a otro, deberá intentarlo con todos aquellos objetos remotos disponibles hasta completar la invocación y sólo fallará cuando no disponga de más candidatos.

Cada servicio del sistema se implementará con un microservicio.

1.2 Servicio de autenticación.

Este servicio permite crear ***tokens* de autorización temporales** a los clientes para acceder a la plataforma. La lista de credenciales aceptadas por el servicio se almacenará en un fichero que sólo podrá ser modificado por un administrador desde fuera del sistema. La selección del fichero a utilizar se realizará mediante una propiedad que leerá el servicio al iniciarse.

La interfaz del servicio dispondrá de dos métodos: *refreshAuthorization()* que devuelve un token de autorización si las credenciales son válidas e *isAuthorized()* que indica si una autorización es válida actualmente o no.

Además de la interfaz, el servicio dispone de un canal de eventos donde se anunciarán los tokens que han quedado revocados (caducados). Obviamente ninguno de los servicios del sistema debe dejar operar a un usuario que esté utilizando un token caducado.

1.3 Servicio de catálogo

Para poder catalogar y buscar el contenido disponible en el sistema se creará este servicio. Dispondrá de una **base de datos persistente** cuya selección se realizará mediante una propiedad que indicará qué fichero utilizar.

Además de la base de datos, también debe indexar los títulos disponibles en la red, así como el servidor que lo almacena, para ello deberá suscribirse al canal de eventos donde el servicio de *streaming* anuncia la disponibilidad de medios. Este otro índice no será persistente puesto que estará cambiando según se inicien o paren el resto de servicios. Cada título de esta lista debe tener una entrada en la base de datos del servicio.

La interfaz dispone del método *getTile()* que permite recuperar toda la información sobre un medio y no requiere autorización. Para buscar por título ofrecerá el método *getTilesByName()* que devolverá una lista de identificadores de títulos que satisfagan el criterio de búsqueda.

Para facilitar la búsqueda en el catálogo se incluirá un sistema de *tagging*. Este sistema permitirá asociar a cada título una lista de *tags* (*strings*), además se dispondrá del método *getTilesByTags()* para buscar títulos que tengan uno, varios o todos lo *tags* de la búsqueda.

La modificación de la base de datos se realizará mediante tres métodos que si requieren autorización: *renameTile()*, *addTags()* y *removeTags()*.

1.4 Servicio de *streaming*

Es el servicio principal del sistema y el que realiza el *streaming*. Queda fuera del ámbito de la asignatura el implementar un protocolo de emisión de vídeo digital, por lo que nosotros vamos a utilizar **GStreamer** para hacerlo. Cuando el servicio se inicia, deberá leer una propiedad que indica el directorio donde se deben buscar los vídeos, los analizará todos y los anunciará utilizando un canal de eventos exclusivo.

El *streaming* será controlado por un objeto remoto que implementará la interfaz *StreamController()*. Este objeto se creará bajo demanda por una factoría disponible en este servicio llamada *getStream()* y que requerirá de una autorización válida.

Además de la factoría dispondrá de otros dos métodos, uno para indicar si cierto título existe o no en el servidor, llamado *isAvailable()* y *reannounceMedia()* que fuerza al servicio a anunciar de nuevo todos sus títulos disponibles.

1.5 Servicio principal

Finalmente, para facilitar la labor de los clientes, se creará un servicio que permita obtener un proxy válido del servicio de autenticación y de catalogado. Para ello se implementarán los métodos *getAuthenticator()* y *getCatalogService()* respectivamente.

Cualquier cliente que quiera interactuar con el sistema, deberá acceder a través de este servicio.

2. Casos de uso

A continuación, se describen una serie de casos de uso que deben soportar los diferentes microservicios.

2.1 Servicio principal

Es el servicio más sencillo, pero también el que sirve de entrada a los clientes de la plataforma. Por eso es importante que sea robusto. Como el resto de microservicios, debe suscribirse al *topic* **ServiceAvailability**, donde se emiten los eventos de *IceFlix::ServiceAvailability()*. Deberá almacenar en memoria todos los servicios anunciados y eliminar aquellos que dejen de funcionar. Además de ello también debe atender las siguientes peticiones de los clientes:

- Main::getAuthenticator(): devolverá alguno de los servicios de autenticación que se hayan anunciado. Si no se ha anunciado ninguno elevará la excepción IceFlix::TemporaryUnavailable().
- Main::getCatalogService(): devolverá alguno de los servicios de catálogo anunciados. Si no se ha anunciado ninguno elevará la excepción IceFlix::TemporaryUnavailable().

2.2 Servicio de autenticación

El funcionamiento de este servicio es muy similar al que ya se implementó en L1 y L2 de la convocatoria anterior. Sin embargo, existen unas diferencias La principal es que en lo relativo al descubrimiento y mantenimiento de los otros microservicios debe comportarse igual que el servicio anterior: descubrir todos los activos y eliminar aquellos que ya no estén disponibles. Otra diferencia importante es la validez de los tokens emitidos: cada token emitido debe caducar a los 30 segundos. A ese respecto, el servicio publicará en el *topic* **AuthenticationStatus** los eventos de la clase *IceFlix::TokenRevocation()* que informan de la caducidad de un token concreto. Además de esa actividad automática, debe servir los siguientes métodos:

- Authenticator::refreshAuthorization(): si las credenciales enviadas en la invocación son correctas, devolverá un nuevo token que deberá caducar a los 30 segundos. En caso de credenciales inválidas, elevará la excepción *IceFlix::Unauthorized()*.
- Authenticator::isAuthorized(): retorna si un token dado está actualmente autorizado o no.

2.3 Servicio de catálogo

Es un servicio muy simple pero que ofrece gran versatilidad al sistema. Permite la búsqueda de los medios disponibles en todo el sistema. Debe suscribirse al *topic* **MediaAnnouncements**, donde se publican los eventos de la clase *IceFlix::StreamAnnounces()*. El evento *newMedia()* es publicado por los servidores de streaming cada vez que disponen de un nuevo medio. Cuando se recibe el aviso de un nuevo stream, este servicio creará una nueva entrada en su base de datos con el identificador del medio y los datos necesarios para completar una estructura *IceFlix::Media*. Uno de esos datos es el proxy del servicio de streaming que dispone del archivo, el servicio de catálogo siempre retornará el último proxy válido conocido. La otra responsabilidad de este servicio es atender la interfaz con los siguientes métodos:

- MediaCatalog::getTile(): retorna un objeto de tipo *IceFlix::Media* dado su identificador. Este método eleva *IceFlix::WrongMediaId()* si el identificador es desconocido y *IceFlix::TemporaryUnavailable()* si el identificador está en la base de datos, pero no se han recibido anuncios de ese stream.
- MediaCatalog::getTilesByName(): busca todos los títulos que cumplan los criterios de búsqueda y lo retorna en forma de lista de identificadores. Si la búsqueda es exacta, la lista contendrá todos los títulos con el mismo nombre que el término de búsqueda; en caso contrario aquellos títulos que incluyan el término de búsqueda en su nombre. Las búsquedas no diferenciaran entre mayúsculas y minúsculas. Si no se encuentra ningún medio, retornará una lista vacía.
- MediaCatalog::getTilesByTag(): retorna una lista de identificadores de medios que cumplan con los criterios de búsqueda. Se pueden buscar todos los medios que tengan la lista de *tags* empleada en la búsqueda o aquellos que sólo coincidan en uno o más *tags*. Si ningún medio cumple los criterios de búsqueda, se retornará una lista vacía.
- MediaCatalog::renameTile(): si la autorización es válida, establece el nombre de stream seleccionado. Lanzará la excepción *IceFlix::Unauthorized()* si la autorización es incorrecta o *IceFlix::WrongMediaId()* si el identificador no es válido.
- MediaCatalog::addTags(): añade la lista de tags a un medio determinado, sólo si la autorización está actualmente vigente. En caso de autorización errónea disparará *IceFlix::Unauthorized()* y si el identificador de medio no es correcto, *IceFlix::WrongMediaId()*.
- MediaCatalog::removeTags(): dada una lista de tags, eliminará dichos tags del medio seleccionado. Si alguno de los tags de la lista no está en el medio, se ignorará. En caso de identificador de stream incorrecto, se elevará la excepción *IceFlix::WrongMediaId()* y si la autorización no es válida, *IceFlix::Unauthorized()*.

2.4 Servicio de *streaming*

Es el servicio más importante de la plataforma, además del más complejo. Se implementa mediante dos clases: IceFlix::StreamProvider() que es la clase principal, y IceFlix::StreamController() cuyos objetos se instancian bajo demanda mediante una factoría getStream() en la clase principal. Cuando el servicio arranca, leerá todos los ficheros de vídeo de un directorio dado para calcular su identificador. Este identificador consistirá en la representación hexadecimal de la suma **SHA256** del fichero. Publicará un evento newMedia() en el *topic* **MediaAnnouncements**, que utiliza la interfaz IceFlix::StreamAnnounces(), por cada archivo de video disponible. Además de esto, también implementa la interfaz con los siguientes métodos:

- StreamProvider::getStream(): es la factoría principal del servicio, construye un objeto de tipo IceFlix::StreamController() y retorna su referencia. Si el identificador del medio es erróneo provocará la excepción IceFlix::WrongMediaId() y si la autorización no es correcta IceFlix::Unauthorized().
- StreamProvider::isAvailable(): retorna si el identificador de medio es válido o no.
- StreamProvider::reannounceMedia(): vuelve a anunciar todos los medios disponibles en el servidor (no debería ser necesario recalcular todos los **SHA256**).

2.4.1 Control de *streaming*

Es el objeto que controla una emisión de streaming en el sistema. Cuando se crea un *StreamController()* se asocia a un determinado medio y esta relación no puede cambiarse. Inicialmente el objeto creará su propio *topic* con el que enviar notificaciones al cliente. Además, se

suscribirá al *topic AuthenticationStatus* para detectar cuando caduca el token con el que se creó el objeto; cuando lo detecte, emitirá una notificación *requestAuthentication()* al cliente e iniciará un proceso de gracia de 10 segundos, si en ese tiempo el cliente no envía la nueva autorización al controlador, éste se interrumpirá y terminará. Para realizar correctamente el control de flujo, también debe implementar:

- StreamController::getSDP(): retorna la configuración del flujo *RTSP* que utilizará el cliente para reproducir el vídeo. Para ello el cliente debe proporcionar una autorización válida (en caso contrario se debe producir la excepción *IceFlix::Unauthorized()*) y el puerto *UDP* donde espera recibir el flujo *RTSP*.
- StreamController::getSyncTopic(): devuelve el nombre del *topic* donde el *StreamController()* notifica al cliente.
- StreamController::refreshAuthentication(): cuando se solicita renovar la autorización al cliente, éste es el método con el que enviará el nuevo token. Si no es válido, se provocará la excepción *IceFlix::Unauthorized()*.
- StreamController::stop(): detiene el flujo de vídeo y elimina el objeto *StreamController()* asociado.

3. El cliente IceFlix

A continuación, se especificarán las características que debe implementar el cliente.

3.1 Conexión y autenticación

El cliente debe permitir especificar el proxy del servicio *IceFlix::Main()* que va a utilizar. En caso de capturar un error *IceFlix::TemporaryUnavailable()*, podría informar al usuario. Pero debe reintentar automáticamente la conexión hasta un máximo determinado de veces que podrá definir el usuario y que por defecto serán 3 intentos, con una pausa de 10 segundos entre intentos.

Cuando la conexión esté establecida (se haya comprobado que el proxy es válido) se debe permitir al usuario autenticarse en el sistema. Esta autenticación no debe ser obligatoria puesto que se permite realizar búsquedas en el catálogo de forma anónima. Una vez se haya autenticado satisfactoriamente, el cliente deberá suscribirse al *topic* de *AuthenticationStatus* para detectar cuando caduca el token obtenido y renovarlo automáticamente cuando reciba el evento adecuado.

En ningún caso el cliente debe mandar la contraseña introducida por el usuario (que tampoco se debe mostrar nunca en el cliente) sino que mandará la representación hexadecimal de la suma **SHA256** del *password*.

También debe permitir cerrar la sesión del usuario (y dejar de renovar sus tokens) para poder cambiar las credenciales si el usuario lo desea.

Por último, el cliente mantendrá informado al usuario en todo momento de que la conexión está establecida y de si existe una sesión iniciada.

3.2 Búsqueda en el catálogo

Cuando el cliente se encuentre *on-line*, permitirá al usuario buscar títulos por nombre o por *tags*. Las búsquedas por nombre se pueden realizar por término exacto o simplemente que el título incluya la

palabra de búsqueda. La búsqueda por *tags* funcionará de manera similar: cuando el usuario introduzca la lista de *tags*, podrá indicar si quiere los títulos cuyos tags tengan alguno en común o tengan que estar todos.

Una vez realizada la búsqueda, si se obtienen identificadores de vuelta, el cliente mostrará el nombre y los *tags* del *stream* referenciado por cada identificador.

El último resultado de búsqueda que haya obtenido títulos se almacenará en memoria de manera que el usuario pueda seleccionar alguno de los títulos obtenidos en cualquier momento. Cuando el usuario seleccione un título, el programa cliente deberá informar al usuario en todo momento de su selección activa.

3.3 Edición del catálogo

Las opciones de edición del catálogo sólo podrán utilizarse si el cliente tiene una sesión activa en ese momento, en caso contrario se informará al usuario que debe autenticarse. Además de tener un token válido, el usuario deberá haber seleccionado alguno de los títulos que haya encontrado en una búsqueda anterior; si no tiene seleccionado ningún título, el error deberá informar al usuario de que necesita uno.

Una vez se disponga de autorización y del título que se desea editar, el cliente debe permitir renombrar el título, añadir tags (uno o varios) o eliminarlos.

3.4 Reproducción de medios

Si el cliente dispone de conexión, autorización y título seleccionado, debe permitir al usuario iniciar la reproducción del flujo. Una vez iniciado una reproducción, el cliente seguirá funcionando con normalidad, además debe permitir interrumpir el flujo en cualquier momento. Cuando se esté reproduciendo un vídeo, el cliente debe informar al usuario de este hecho de forma continua.

3.5 Programas auxiliares

De los microservicios aquí descritos, dos de ellos no pueden realimentarse desde el propio sistema, sino que debe hacerse de manera externa por un administrador. Estos dos servicios son el de autenticación y el de *streaming*. Esto quiere decir que desde el sistema *IceFlix* no se pueden añadir usuarios ni vídeos nuevos.

Para poder operar más fácilmente con el sistema, se deben crear también dos herramientas que permitan añadir usuarios y vídeos **sin necesidad de parar los servicios afectados**. Para implementar esto se puede tomar como referencia el servidor de autenticación que se entregó para resolver la entrega L2 de la convocatoria anterior. Dicho servidor disponía de un manejador de señal *SIGUSR1* de manera que, si el proceso recibía dicha señal, se volvía a cargar el archivo *users.json*. De esta forma, para añadir usuarios sin parar el servicio, se modificaba el fichero con el nuevo usuario y se mandaba la señal *SIGUSR1* al proceso.

4. La entrega

Se entregará un único fichero comprimido en **zip** o **tar.gz**, la estructura interna del archivo queda a decisión de los estudiantes excepto por los siguientes ficheros que **no deben estar en ningún subdirectorio** en el paquete y **deben disponer de los permisos adecuados** para poder ejecutarlos:

- run_icestorm: inicia *IceBox* para ejecutar *IceStorm*. El proxy del *TopicManager* que debe crear es: **IceStorm/TopicManager:tcp -p 10000**
- run_iceflix: ejecutará una instancia de todos y cada uno de los microservicios que forman el sistema.
- run_client: lanzará el programa cliente
- add_user <usuario> <contraseña>: añadirá dicho usuario con la contraseña indicada al archivo de credenciales del microservicio de autenticación, además debe provocar que se recargue la lista de usuarios en el servicio. Cabe recordar que en el archivo se almacena el ***hash* SHA256** del *password*.
- reload_media: provocará que el servidor de *streaming* vuelva a escanear el directorio de medios **anunciando los nuevos**.
- client_help.txt: un pequeño documento que explicará paso por paso cómo reproducir un medio y cómo editar su nombre y *tags*. Ejemplo de formato del fichero podría ser:
    1. Seleccionar <conectar> e introducir el proxy del servicio *IceFlix::Main()*
    2. Seleccionar <autenticar> e introducir usuario y contraseña
    3. Seleccionar <buscar> e introducir “ejemplo”
    4. Seleccionar primer resultado de búsqueda pulsando “1”
    5. Seleccionar <reproducir>
- README.md: fichero con la lista de alumnos del grupo y URL al repositorio en GitHub.

Para poder probar la práctica de manera directa, se puede incluir un archivo de vídeo de no más de 5MB a modo de ejemplo.

Es muy importante seguir las indicaciones sobre la entrega de manera exacta ya que gran parte del proceso de evaluación se realiza de forma automática, por eso los nombres de fichero deben ser exactamente los indicados anteriormente.

Adicionalmente a la evaluación automática también, realizará un análisis estático de código con **pylint**. Se considera que el estilo es correcto si obtiene como mínimo un **9.0** de puntuación global.

ATENCIÓN: Si tenéis miembros en el equipo que utilicen Windows, tened en cuenta que **los retornos de línea** deben ser **TIPO UNIX**. Si se utilizan los retornos tipo DOS la práctica no funcionará en el entorno de pruebas.

4.1 Documentación de entrega

No se solicitará memoria de prácticas ni documentación adicional en la entrega. La evolución del proyecto se consultará con el historial de *commits* de GIT. Por tanto, **no se recomienda subir la práctica en uno o dos commits**. El repositorio debe tener **commits de todos los miembros del equipo**.

4.2 Defensa de prácticas

Sólo se solicitará una defensa de práctica en casos especiales, por ejemplo:

- Repositorio con muy pocos *commits* o sin *commits* de algún miembro.

- El buscador de plagios alerte sobre dos o más entregas.

Una vez publicadas las notas, los alumnos que así lo deseen, podrán defender su práctica de manera "manual" para demostrar que cumple los criterios utilizados en la evaluación.