tags: iic2173_2024_2

2024-2 / IIC2173 - E0 | CoolGoat Async

Fecha de entrega: *02/09/2024 - 3.5 Semanas*

Introduccion al proyecto del curso

A nivel general el proyecto del curso busca dar a los alumnos una experiencia práctica del desarrollo de un sistema de software compuesto de diferentes componentes. Dicho sistema será desarrollado en grupos y la idea es que este desarrollo sea lo más parecido a un sistema real posible. Para partir con esto, la **entrega 0 es de carácter individual** y en futuras entregas se trabajará en grupo. Así el proyecto abarca temáticas como protocolos, integraciones, seguridad, deployment, entre otros. **El proyecto deberá siempre correr en un entorno Cloud para obtener los conocimientos prácticos del desarrollo y *deployment* de software en la nube.**

Enunciado

Dentro de los últimos años se han vuelto más populares las aplicaciones enfocadas en fútbol y las transacciones que vienen asociadas a estas, pero esto ha complicado a las personas en tener que estar muy atentas de los resultados y como les afectan en sus predicciones del futuro.

Dentro de los desafíos para este problema se pueden enumerar tanto:

- Obtener la información de los partidos fácilmente
- Poder buscar dentro de todas las opciones posibles de resultados las que sean mejores y más provechosas

Es por esto que la empresa LegitBusiness decidió optar por incentivar a pequeños grupos de desarrollo para poder entrar a este mundo mediante su apoyo con nuevas tecnologías, pero que primero se deberá crear una *prueba de concepto* para ver su funcionamiento.

Para la prueba de concepto, le han pedido que se conecte a un broker MQTT de eventos (servidor que acepta mensajes de un cliente publicador y los difunde entre los clientes suscritos, en este caso nosotros enviaremos los mensajes y ustedes deberán suscribirse), que emite la información de distintos partidos cada 5 minutos. Estos eventos tienen el siguiente schema JSON y se publican en el canal **fixtures/info**, en este caso los eventos

serán la información del partido del cual ustedes deberán vender su pasaje en futuras entregas.

```
{
    "fixtures":[
        "fixtures":
        {
        "id":<int>,
        "referee": <string|null>,
        "timezone": <string>,
        "date": <date>,
        "timestamp": <int>,
        "status": {
            "long":<string>,
            "short": <string>,
            "elapsed": <int|null>,
        },
    },
    "league": {
        "id": <int>,
        "name": <string>,
        "country": <string>,
        "logo": <string>,
        "flag": <string>,
        "season":<int>,
        "round": <string>,
    },
    "teams": {
        "home": {
            "id":<int>,
            "name":<string>,
            "logo": <string>,
            "winner": <bool|null>,
        },
        "away": {
            "id": <int>,
            "name": <string>,
            "logo": <string>,
            "winner": <bool|null>
        },
    },
    "goals": {
        "home":<int|null>,
        "away":<int|null>,
    },
    "odds": {
        "id":<int>,
        "name":<string>,
        "values": any[],
    },...
]
}
```

En este caso la información de los partidos viene como un JSON hecho string para manejar posibles inconsistencias en los datos y el tamaño de estos. Deberán obtener el valor,

parsearlo a un formato que puedan manejar (revisen la sección de enlaces útiles), y luego guardarlo como corresponda.

Debe crear una plataforma que pueda mostrar una lista de los vuelos que lleguen desde el broker desde el canal **fixtures/info**.

Especificaciones

Para esta tarea personal, "Entrega 0", la idea es que todos aprendan la base de una aplicación montada en la web que sea capaz de integrarse a un software externo, por lo que deberán crear y configurar un servicio web que implemente un **pequeño servicio que funcione como API**.

Pueden desarrollar su solución con el framework que deseen de esta lista:

- Servicio web:
  - Ruby
    - Rails (Version API)
  - Python
    - FastAPI
    - Django
  - Javascript
    - Koa
    - Express
    - NestJS
- Lenguajes para el servicio de conexión a broker:
  - Ruby
  - Python
  - Javascript
  - Go
  - C#
  - C/C++
  - etc..

Queda prohibido y no se revisarán aplicaciones hechas en PHP.

Cada servidor tendrá que tener un dominio asignado. Los dominios TK, ML o GA son gratuitos, y pueden conseguir los ME fácilmente en Namecheap utilizando el github student

pack. También pueden usar otro proovedor de dominio si estiman conveniente. Finalmente, en el servidor deberán configurar un servicio proxy inverso con NGINX que esté escuchando en el puerto 443, asegurado con SSL.

Para esto, podrán usar Let's Encrypt con certbot, un servicio gratuito de implementación de HTTPS.

Finalmente, solo pueden usar los siguientes proveedores en la nube, y sus servicios IaaS:

- AWS

Los ayudantes podrán responder sus dudas en AWS. Además, las ayudantías se referirán a AWS. Las siguientes herramientas / plataformas están prohibidas y **no se corregirá nada que ocupe estos servicios:**

- Heroku
- Ciertos servicios de AWS
  - LightSail
  - Elastic Beanstalk
  - Amplify
  - Cognito (para esta entrega)
- Netlify
- Firebase (excepto para notificaciones móviles)

Y cualquier BaaS que implemente funcionalidad fuera de su código.

En caso de que no tengan tantos conocimientos de desarrollo web recomendamos que usen stacks tecnológicos que ya conozcan o que sean usados en las ayudantías. Por otro lado si quieren indagar en nuevos stacks tecnológicos pueden usar el proyecto del curso para explorar nuevas tecnologías. Recuerden que luego el proyecto se vuelve grupal asi que vean igual con sus compañeros que es lo que más les acomoda para que puedan reutilzar la mayor cantidad de código posible. Recuerden que **no hay mejores tecnologías sino que algunas son más adecuadas para algunos contextos o tareas**.

Puntaje

Esta entrega consiste en dos partes, la parte mínima (que todos deben lograr) que vale 75% de la nota final y una parte variable que también vale 25%. Sobre la parte variable, tendrán 2 opciones para trabajar, de las que deberán escoger 1. Cada una de las que escojan para evaluar vale 25% de la nota final, y **realizar la segunda parte puede dar hasta 5 décimas.**

Por otro lado, debido a que esta entrega presenta una buena cantidad de bonus, **la nota no sumará más de 8**, para que decidan bien que les gustaría aprovechar.

Los requisitos marcados como ***Esencial*** son obligatorios para que su tarea sea revisadas, esto pues o son fundamentales para el aprendizaje, o son necesarios para las siguientes entregas del proyecto. **De no cumplir con estos, su tarea no será revisada y será calificada con nota 1.**

Requisitos funcionales (10p)

Consideren que: no se les pide crear una interfaz para su aplicación (sin HTML), sino que debe ser una API que entregue la información en formato JSON.

- **RF1: (3p)** ***Esencial*** Debe poder ofrecer en una **API** la lista de los distintos partidos que se han encontrado en el broker a medida que se vayan recibiendo, de forma que muestren el detalle y cuando fue su última actualización. En las llamadas se pueden ir agregando nuevos partidoss, los cuales deben ser manejados y mostrados en este endpoint. Esta lista debe ser accedida a través de HTTP en la URI que ustedes estimen conveniente, por ejemplo: `{url}/fixtures`.
- **RF2: (1p)** ***Esencial*** Debe ofrecer un endpoint para mostrar el detalle de cada partido recibido desde el broker. Por ejemplo: `{url}/fixtures/{:identifier}`
- **RF3: (2p)** ***Esencial*** La lista de partidos debe estar paginada por default para que muestre cada 25 partidos y poder cambiar de pagina cambiando un *queryParam*. Por ejemplo: `{url}/fixtures?page=2&count=25`. Queda a criterio de ustedes si permiten traer más valores mediante otro número del count en *queryParams*.
- **RF4: (4p)** ***Esencial*** El endpoint `{url}/fixtures` debe permitir filtrar los partidos por el equipo home, visita y fecha del partido de la forma: `{url}/fixtures?home=LIS&visit=GRU&date=2024-03-14`. Acá *home* es el código del equipo home, *visit* es el código del equipo visita, y *date* es la fecha del partido. Para este filtro siempre deben de traer los partidos que todavía no se jugan, es decir, ignorar los pasados.
    - *Nota: Pueden hacer funcionar el filtro por partes, es decir, si solo se agrega el query param de fecha, pueden traer solo los partidos que cumplan ese filtro en vez de solicitar los otros.*

Nota: es importante que sea paginada pues el emisor de eventos enviará miles de mensajes de partidos durante el desarrollo y corrección de sus proyectos, por lo que si hacen una consulta a la base de datos para que traiga todos los eventos recibidos es probable que se caigan sus entregas.

Requisitos no funcionales (20p)

- **RNF1: (5p)** ***Esencial*** Debe poder conectarse al broker mediante el protocolo MQTT usando un proceso que corra de **forma constante e independiente de la aplicación web** (que corra como otro programa), los eventos recibidos deben ser persistidos con su sistema para que estos puedan ser mostrados (existen diferentes opciones). Para

esto debe usar las credenciales dentro del repositorio y conectarse al canal **fixtures/info**.

- **RNF2: (3p)** Debe haber un proxy inverso apuntando a su aplicación web (como Nginx o Traefik). *Todo lo que es Nginx es mejor configurarlo directamente en la instancia EC2 y no necesariamente con Docker.*
- **RNF3: (2p)** El servidor debe tener un nombre de dominio de primer nivel (tech, me, tk, ml, ga, com, cl, etc)
- **RNF4: (2p)** ***Esencial*** El servidor debe estar corriendo en EC2.
- **RNF5: (4p)** Debe haber una base de datos Postgres o Mongo externa asociada a la aplicación para guardar eventos y consultarlos. **Pueden usar Postgis por un bonus de 2 decimas para guardar información greográfica de cada partido.**
- **RNF6: (4p)** ***Esencial*** El servicio (API Web) debe estar dentro de un container Docker.

Docker-Compose (15p)

Componer servicios es esencial para obtener entornos de prueba confiables, especialmente en las máquinas de los desarrolladores. Además esta herramienta será necesaria durante el resto del desarrollo del proyecto para orquestar sus contenedores y servicios.

- **RNF1: (5p)** Lanzar su app web desde docker compose
- **RNF2: (5p)** Integrar su DB desde docker compose (Es decir la base de datos es un contenedor).
- **RNF3: (5p)** Lanzar su listener MQTT desde docker compose y conectarlo al contenedor de la app web (o base de datos si lo usara)

Variable

Deben elegir al menos uno de los dos grupos de requisitos siguientes.

HTTPS (25%) (15p)

La seguridad es esencial para sus usuarios, por ello los datos que viajan en su aplicación web deben encriptarse con los protocolos correspondientes.

- **RNF1: (7p)** El dominio debe estar asegurado por SSL con Let's Encrypt.
- **RNF2: (3p)** Debe poder redireccionar HTTP a HTTPS.
- **RNF3: (5p)** Se debe ejecutar el chequeo de expiración del certificado SSL de forma automática 2 veces al día (solo se actualiza realmente si está llegando a la fecha de expiración).

Algunos software de terceros requieren que sus aplicaciones funcionen con HTTPs, por lo que sin esta capa de seguridad no se puede integrar a dichos servicios. En futuras entregas

será necesario tener implementado HTTPs por lo antes mencionado, por lo que si lo logran mejor!.

Balanceo de Carga con Nginx (25%) (15p)

Para escalar el servicio de forma horizontal podemos replicar el contenedor de la aplicación web y utilizar NGINX como balanceador de carga, así podemos crear tantos servidores como nuestro servicio necesite. Esto es altamente necesario para aplicaciones que en ciertos momentos tendrán una carga importante y en otros no.

- **RF1: (5p)** Debe replicar al menos 2 contenedores de su aplicación web para que corran en paralelo.
- **RF2: (10p)** Debe configurar Nginx para que haga un balanceo de carga hacia los servidores levantados (Pueden encontrar la configuración en la documentación de NGINX).

Recomendaciones

- Lo más importante no es que la aplicación esté funcionando al 100%, sino que el servidor exista y se pueda acceder a la aplicación correctamente.
- Para esta entrega basta con que se pueda llevar a cabo las funcionalidades solicitadas manteniendo el estandar de una API. Fíjense en la proporcion entre RNF y RF. **No será necesario hacer una interfaz visual.**

Roadmap sugerido

Para simplificar esta primera entrega, le sugerimos seguir los siguientes pasos

- Ejecute pruebas de concepto para conectarse al cliente MQTT
- Levante un servidor web que pueda leer los eventos recibidos.
- Ponga su servicio en un container Docker
- Levante una máquina en AWS EC2 y abra los puertos
- Instale docker en la máquina
- Copie su aplicacion y construya el container
- Corra su servicio y termine su configuración
- Todo lo que es Nginx es mejor configurarlo directamente en la instancia EC2

Recuerden que la entrega debe estar corriendo en el EC2 en cloud, no se corregirán entregas locales, por lo que avancen incrementalmente en el cloud. Recuerden, funcionalidad que no esta en producción no está realmente terminada, **no subestimen la dificultad del deployment**, pues es mucho más manual que en otros cursos como ingeniería de software o desarrollo web.

A continuación se muestra a modo de guía un diagrama UML de componentes una posible solución de la entrega

En caso de que se implemente el bonus de NGINX con balanceador de carga se tienen multiples *CoolGoat API* que están replicados, el resto se mantiene constante.

Entrega

Se les proporcionará un repositorio de Github Classroom donde pueden subir su código e ir registrando sus commits.

Deben subir el código de su solución junto al archivo de configuración de Nginx (o Traefik u otro) en el repositorio que se les asignará vía github classroom.
También deben entregar el archivo .pem asociado al servidor EC2 para tener los respectivos accesos y poder realizar una buena corrección.
Además, para poder facilitar la corrección deben generar un README.md que señale:

- Consideraciones generales
- Nombre del dominio
- Método de acceso al servidor con archivo .pem y ssh (no publicar estas credenciales en el repositorio).
- Puntos logrados o no logrados y comentarios si son necesarios para cada aspecto a evaluar en la Parte mínima y en la Parte variable.
- De realizar un tercer requisito variable también explicitar en el readme.

Pueden sobrescribir este README sin problemas o cambiarle el nombre

Además, y como algo muy importante: **ESTÁ ABSOLUTAMENTE PROHIBIDO SUBIR SU ARCHIVO .PEM A SU REPOSITORIO DE GITHUB.** Si hacen esto se les calificará con nota 1. Para esto se les habilitará un buzón de canvas para que nos lo compartan.

Esta entrega es estrictamente individual y será revisada para casos de copia.

Atraso

Para esta entrega se les descontará 0.5 puntos en la nota máxima por horas Fibonacci con F1 = 6 y F2 = 6.

Se considerará como atraso cualquier modificación en features o implementación que tenga que ver solo con lo que se pide en esta entrega.

Fibonacci	Hora	Nota maxima

6	0:01 - 5:59	6.5
6	6:00 - 11:59	6
12	12:00 - 23:59	5
18	24:00 - 41:59	4.5
30	42:00 - 71:59	4
...	72:00 en adelante	1

Enlaces relevantes

- JSON strings parsing
  - Python
  - JS
  - Ruby
- MQTT
  - JS - MQTT.js tutorial
  - Python - Pago-MQTT
  - Ruby - Ruby-MQTT
- Github Student Pack
- Página AWS
- Docker - Instalación
- NGINX - Reverse Proxy
- Namecheap - Github Student Pack
- https://www.tutorialspoint.com/docker/docker_compose.htm
- https://phoenixnap.com/kb/ssh-to-connect-to-remote-server-linux-or-windows
- https://www.digitalocean.com/community/tags/deployment?type=tutorials
- https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-20-04-es
- https://www.nginx.com/blog/using-free-ssltls-certificates-from-lets-encrypt-with-nginx/
- Certbot
- Documentación AWS SES
- Documentación de OpenStreetMap

Ayudantías Útiles

- **Ayudantía 1** - Cloud 1: AWS, Linux y EC2 - 08/03/24
- **Ayudantía 2** - Cloud 2: Docker, Docker-Compose y alternativas - 15/03/24
- **Ayudantía 3** - Cloud 3: Deployment básico - 22/03/24

Apoyo

Pueden usar el Slack del curso en el canal #E0 para dudas más rápidas.