Brainstorm

* Modelo predictivo para inversión en Forex
* Posibles ideas de monetización del modelo, a parte de la inversión por cuenta propia:
  + Un usuario pueda copiar las ordenes en su bróker a través de integraciones con nuestra API REST.
  + La plataforma de trading elegida permite varias opciones que permiten a los desarrolladores generar ingresos:
    - Subir el modelo a la plataforma de trading como AE (Bot automático) para que otros usuarios de la plataforma puedan copiarla comprando el modelo o alquilándolo.
    - Subir el modelo a la plataforma como indicador para que otros usuarios puedan ejecutar manualmente las señales.
  + Crear un servicio de señales en una web

NOTA: estas ideas son de cara al futuro, la arquitectura diseñada para la práctica comprende solo la arquitectura para implementar el modelo, generar señales y enviarlas al bróker con una cuenta personal.

Diseño del DAaaS

Definición la estrategia del DAaaS

*Definir el catálogo de servicios que proporcionará la plataforma DAaaS, que incluye incorporación de datos, limpieza de datos, transformación de datos, datapedias, bibliotecas de herramientas analíticas y otros.*

Servicio de predicción en tiempo real del precio de activos Forex, generación de señales de compra o venta en base al modelo predictivo y ejecución automatizada de dichas señales sin interacción humana en el menor tiempo posible.

Arquitectura DAaaS

*Definir la selección de componentes, la definición de procesos de ingeniería y el diseño de interfaces de usuario. Diseño y ejecución de Proofs-of-Concept (PoC) para demostrar la viabilidad del enfoque DAaaS.*

* Extracción de datos históricos a través de la API REST de la plataforma de trading para entrenar el modelo.
* Obtener datos en streaming para pasarlos al modelo y que nos genere una señal de compra o venta en tiempo real. Se hará a través de un websocket TCP con un cliente en el servidor de la plataforma de trading que enviará cada actualización a un servidor Python actuando como endpoint.
* Necesitamos un modelo predictivo que sea escalable para que construya las predicciones a una velocidad aceptable. Por ello se ha optado por Tensor Flow Serving alojado en Google Cloud ML Engine.
* Los datos históricos para entrenar al modelo se guardarán en Google Storage en formato CSV.
* Para la gestión de los datos en streaming se ha optado por una arquitectura dirigida por eventos. Por ello, un componente clave de la arquitectura es un cluster de Kafka en la nube con dos topics, uno para recibir los datos en streaming de la plataforma de trading y otro para las señales de salida que genera el modelo también en streaming.
* Para conectar todos los puntos de la arquitectura falta aún un componente clave, el encargado de hacer circular los datos entre todos los componentes. Para ello, se ha decidido utilizar una máquina virtual con varios microservicios Python que gestionen este flujo. En concreto cada servicio se encargará de:
  + Hacer una llamada a la API REST de la plataforma de trading para obtener datos históricos de los activos y guardalos en Google Storage en formato CSV.
  + Establecer una conexión TCP con el cliente y permanecer escuchando continuamente para recibir los datos en tiempo real. Cada mensaje que reciba se preprocesará para conseguir la calidad adecuada del dato y se enviará al topic de entrada de Kafka mediante el cliente de Kafka Streams para Python.
  + Consumer del topic de entrada que enviará por gRPC al modelo, recibirá la respuesta del modelo y la enviará de nuevo por medio de un producer al topic de salida de Kafka.
  + Consumer del topic de salida, que aplicará una serie de ultimas validaciones a la predicción recibida y hará http requests a la API del bróker para colocar las ordenes de compra o venta.

DAaaS Operating Model Design and Rollout

*Personalizar los modelos operativos DAaaS para cumplir con los procesos, la estructura organizacional, las reglas y el gobierno de los clientes individuales. Realizar seguimiento de consumo y mecanismos de informe.*

Hay que tener en cuenta que el mercado Forex opera 24 horas al día de lunes a viernes por lo que todos los servicios (menos Google storage) deben de estar levantados durante ese periodo.

Los viernes, una vez cerrado el mercardo se ejecutará primero un clud function que arrancará Google storage y luego, un crontab que arrancará el servicio de descarga de datos históricos a Google storage. El mismo código detendrá el Google storage cuando haya acabado.

Los viernes, a la hora que cierra el último mercado y después de haber ejecutado el punto anterior, se ejecutará un cloud function programado con un Scheduler que detendrá todos los servicios.

Los lunes a la hora de la apertura del primer mercado se ejecutará un cloud function para arrancar de nuevo todos los servicios. Además, en la maquina virtual habrá programado un crontab que al iniciar la maquina arrancará los microservicios alojados en la máquina.

Los modelos se reentrenarán manualmente los fines de semana, eso incluye levantar a mano el Google storage y entrenar los modelos con los nuevos datos y detener de nuevo el Google storage.

\*punto de mejora: TensorFlow model serving permite A/B testing, por lo que sería interesante en el futuro entrenar los modelos continuamente con datos en streaming provenientes de Kafka mientras que el modelo anterior sigue en funcionamiento. Por ejemplo con Ksql se pueden hacer querys directamene a los topics de Kafka e incluso utilizar Kafka como nuestra fuente de datos históricos a partir de un punto del tiempo en el que ya tenemos los datos pasados y prescindir así del proceso de descarga de datos históricos y archivado en csv.

Desarrollo de la plataforma DAaaS.

*Construcción iterativa de todas las capacidades de la plataforma, incluido el diseño, desarrollo e integración,* ***pruebas****, carga de datos, metadatos y población de catálogos, y despliegue.*

* Primero, crear la maquina virtual y descargar los históricos a Google storage.
* En segundo lugar, habría que construir un modelo que funcione bien. En principio sería un modelo por activo.
* Una vez que tenemos un modelo que funciona al menos para un activo, seguiríamos con obtener los datos en tiempo real del bróker. Hay que crear un EA en la plataforma con el código propio que utiliza esta plataforma (tengo ejemplo guardado), luego crear en la VM el servidor que reciba estos mensajes.
* Tras haber conseguido obtener datos en tiempo real, crearemos el cluster Kafka en la nube y le añadiremos los topics. En principio cada topic podría tener una partición por cada activo o modelos que tengamos. Configurar las particiones para que una vez obtenido un menaje por un consumer este se destruye si es el efecto que deseamos.
* Creamos el tensor Flow server y subimos le modelo y hacemos la configuraciones necesarias.
* Escribimos el código Python que actuará como consumer - gRPC requester – producer.
* Escribimos el código que consumirá el topic de salida, hará las peticiones a la API para lanzar las posiciones. Este código, o uno relacionado deberían de hacer ciertas validaciones antes de lanzar las ordenes , como que las predicciones no estén desactualizados en comparación con la hora real del mercado, gestión del riesgo de la cartera y demás criterios financieros útiles para proteger el capital del inversor.
* Probamos el producto con una cuenta demo y comparamos el modelo con cuenta demo y con datos históricos. Puede haber pequeñas direncias en los precios.
* Si todo cuadra, lanzamos el modelo en una cuenta real con un capital mínimo. Comparamos modelo con operativa real con modelo con datos históricos. Si toda parece ir bien por un tiempo, lanzamos el producto.

Link a Diagrama:

<https://github.com/xuteo/Practica-BigData-arquitectura/blob/3b71a81632d62ed57cca5dbdcc38c3eb5cfcc026/Diagrama.PNG>