Análisis de un modelo qCBR aplicado en AWS SageMaker

Antes de empezar a utilizar la filosofía de MLops, el usuario debe tener conocimientos sobre las herramientas brindado por los servicios de paga que desee aplicar, conocer la estructura del modelo, los parámetros, el diseño de entrada de los datos. Esto se debe a que al utilizar un servicio basado en la filosofía de MLops nos traerá costos y empresarialmente lo que buscamos es la optimización de los recursos en función al tiempo.

Teniendo en cuenta que los dispositivos de computación cuántica adolecen de limitaciones tecnológicas, como limitaciones de qubits, problemas de ruido (Noisy) o de decoherencia, en la actualidad el uso de estas computadoras cuánticas aplicado a machine learning ha ido en aumento, esto se debe a los aumentos en el control de la coherencia y los sistemas de corrección de errores, contando también con un aumento en la oferta de servidores. Se encontraron varias empresas proveedoras del servicio de computación cuántica, solo destacaremos 5 empresas; (I) IBM Q EXPERIENCE, (II) GOOGLE QUANTUM AI, (III) AMAZON BRAKET, (IV) XANADU QUANTUM CLOUD, (V) FOREST: RIGETTI COMPUTING.

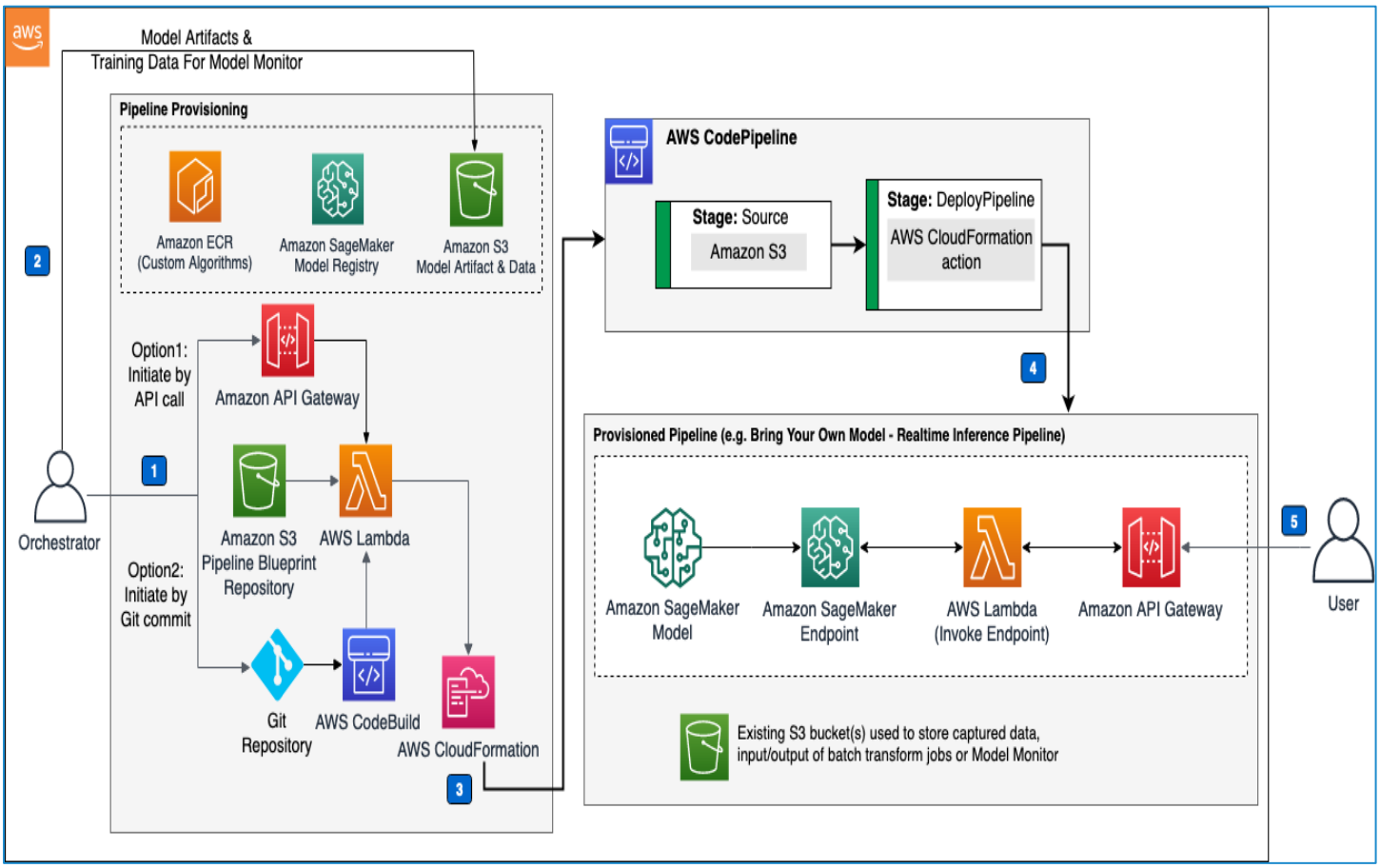
Debemos tener en cuenta que cuando trabajamos con computación cuántica la entrada de datos debe estar en un formato de datos cuánticos, en el día a día la información se procesa en ordenadores clásicos los cuales reciben y almacenan la información en bits, por ende, si deseamos trabajar con un modelo cuántico debemos llevar nuestros datos a un formato cuántico, es decir, con qubits. A partir de estos se construyen los estados cuánticos computacionales, los estados binarios se represan de la siguiente manera; $|\psi> = \alpha|0> + \beta|1>$.

Como trabajaremos con un modelo Case-Based Reasoning aplicado a métodos cuánticos (qCBR), debemos entender que el modelo estadístico Case-Based Reasoning es un enfoque basado en la toma decisiones según experiencias, este enfoque se desarrolló en el siglo 20’ en la universidad de Yale, en una clase de psicología. En síntesis, el modelo lo que hace es analizar todos los casos de éxito y sus soluciones para poder dar una predicción, pero al ser problemas extensos pueden traer consigo problemas computacionales, ordenadores clásicos tienen problemas al trabajar con enormes data sets y los ordenadores cuánticos pueden trabajar estos data sets sin problema alguno.

Sin embargo, como ya se conoce uno de los problemas actuales es que las empresas trabajan con ordenadores y datos clásicos. Como científicos de datos debemos buscar un equilibrio entre un modelo óptimo y los costos operacionales, ya que eso nos permitirá tener una ventaja competitiva en el mercado, trabajar con datos cuánticos implica el uso de otras plataformas de procesamiento y cloud aumentando los costos de manera significativa. Por eso el común denominador de los proyectos de MLops que trabajaremos serán con datos clásicos.

Los servicios de MLops buscan resolver la calidad del modelo, sesgo y explicabilidad del modelo con un monitoreo constante, esta solución aumenta la agilidad y la eficiencia en el equipo de data science y nos permite repetir procesos exitosos a escala, realizaremos esta herramienta que optimizará la productividad en nuestras instituciones. Por consiguiente, resalto que dentro del modelo de SageMaker que explicaré no tomaré en cuenta el tratamiento de los datos clásicos llevados a cuánticos y su desarrollo desde un quantum cloud.

Gráfico N.º 1. Template option 1: Single account deployment



Elaboración: Amazon SageMaker

Con el transcurso del aprendizaje a través de los módulos hemos podido aprender las distintas técnicas y pipelines que debemos seguir para poder desarrollar un algoritmo aplicado a un tema.

En el primer punto el **"Orchestrator"** o **"data scientist"** generará la estructura del siguiente pipeline de como trabajar un algoritmo utilizando SageMaker.

En el primer paso nosotros como científicos de datos tenemos dos opciones.

- La primera opción nos permite iniciar nuestro modelamiento de los datos mediante la llamada de un API, continuando con **AWS Lambda**
- La segunda opción nos permite iniciar mediante un commit de GIT, luego del commit del repositorio, continuamos con **AWS CodeBuild** este servicio de integración continua completamente administrado que compila código fuente, ejecuta pruebas y produce paquetes de software que están listos para implementar.

Ambos puntos nos llevan a utilizar **AWS Lambda**, un servicio que administrará nuestro modelo ofreciendo una combinación equilibrada de memoria, CPU, red y otros recursos para ejecutar nuestro modelo. Esto en términos de negocios es un servicio eficiente ya que nos permitirá tener un MAYOR control de nuestros gastos y uso de los servicios

de AWS. **Ambas opciones nos permiten subir nuestro modelo el cual hemos desarrollado en nuestro IDE (Integrated development environment) con nuestras propias características y parámetros.**

Mientras tanto el "orchestrator" deberá realizar las configuraciones pertinentes del modelo al utilizar el servicio de **Amazon ECR** obtendremos un contenedor ejecutable, independiente, ligero que integra todo lo necesario para ejecutar nuestro modelo, incluidas bibliotecas, herramientas del sistema, código y tiempo de ejecución. Seguido aplicamos **Model Registry** lo que nos permite especificar y versionar nuestro modelo y luego con **Amazon S3 Model Artifact & Data** que nos permite referenciar nuestra información relacionada a nuestro modelo, parámetros y lineamientos que deseemos que se cumplan. Este también proporciona acceso a pedido a los informes de cumplimiento y seguridad de AWS y acuerdos en línea con el proveedor del servicio.

Después de tener en claro los pasos del primer bloque seguimos con **AWS CloudFormation** el cual nos permitirá implementar y actualizar la informática, bases de datos y otros recursos con un estilo simple y enunciativo que abstrae la complejidad de las API de recursos específicos y continuar con el tercer bloque en el cual tomaremos el servicio de **AWS CodePipeline** que es un servicio de entrega continua completamente administrado que lo ayuda a automatizar sus canalizaciones de lanzamiento para actualizaciones de infraestructura y aplicaciones rápidas y confiables. CodePipeline automatiza las fases de creación, prueba e implementación de su proceso de lanzamiento cada vez que hay un cambio de código, según el modelo de lanzamiento que defina.

Lo que nos lleva al bloque final, la parte final de este proceso el cual nos permitirá seleccionar o subir el modelo desarrollado, luego a través de SageMaker Endpoint podremos realizar inferencias en tiempo real a través de una API REST, la cual pasa previamente por el servicio de AWS Lambda, todo este pipeline o bloque final almacena los resultados en un S3 bucket servicio también proporcionado por AWS.

Según lo descrito anteriormente, podemos canalizar la implementación de un modelo clasificador de la siguiente manera:

1. Antes de desplegar un modelo debemos tener en cuenta el ciclo de vida de un modelo
    1.1. Encontramos la necesidad de negocio o problema a solucionar.
        1.1.1. Buscamos la base del problema a solucionar.
        1.1.2. Identificamos los riesgos con respecto al modelo.
        1.1.3. Definimos el ROI en función al problema que soluciona la necesidad de negocio.
    1.2. Preparamos y estructuramos los datos.
        1.2.1. Analizamos e identificamos la fuente de los datos.
        1.2.2. Configuramos el acceso a los datos.
        1.2.3. Verificamos el sesgo y su ajuste.
        1.2.4. Preprocesado de los datos para la entrada del modelo.
    1.3. Desarrollamos el modelo.

1.3.1. Seleccionamos los modelos cuya estructura matemática se ajuste a nuestros datos y necesidad de negocio.

1.3.2. Separamos los datos para entrenamiento y test de los datos.

1.3.3. Revisar los parámetros del modelo.

1.3.4. Subir a un repositorio o generar un api de nuestro modelo.

1.4. Validamos el modelo-

1.4.1. Documentación de los riesgos en el modelo.

1.4.2. Test y validación de la entrada de datos del modelo para poder validar el modelo y su precisión.

1.5. Depuración del modelo

1.5.1. Validamos la calidad del código del modelo, así como los estándares de seguridad.

1.6. Monitoreo del modelo

1.6.1. Drift and Model Decay : Análisis periódico de los resultados del modelo y ajuste de este ante nuevos datos de entrada.

1.6.2. Automated monitoring para modelos machine learning que permita automatizar ajustes del drift.

1.6.3. Análisis del ROI.

2. Desarrollamos el modelo clasificador (Versión 1)

2.1. Si lo almacenamos y versionamos en GIT, generamos un commit .

2.2. Si el modelo tiene una API se utiliza la API.

Luego, utilizamos el servicio de AWS Lambda para así buscar administrar los recursos de manera eficiente, en función a la estructuración de nuestro modelo clasificador.

3. Una vez seleccionado el servicio de contenedor de nuestros datos, librerías, entre otros y de haber registrado el modelo para su ejecución, referenciamos nuestros parámetros de información del modelo Versión 1. Cada vez que hagamos una mejora en nuestro modelo este se versionará a través de S3 Artifics & Data, este servicio también nos permitirá mantener la seguridad del funcionamiento de nuestro modelo según los parámetros de cumplimiento.
4. Luego para el versionamiento de nuestro modelo ya que el entrenamiento de los modelos es continuo, usaremos el servicio de AWS CloudFormation lo cual nos permitirá versionar nuestro modelo, la entrada de los datos, entre otras opciones para que cualquier modificación que se de en nuestro modelo de manera “sencilla”.
5. Para poder mantener las actualizaciones del versionamiento de nuestro modelo se selecciona el servicio AWS Codepipeline este servicio nos permitirá automatizar las fases de creación de nuestro modelo, los test y los cambios según el versionamiento de nuestro modelo.

6. El último paso es cuando ponemos en producción nuestro modelo, en el cual el usuario introducirá los datos mediante el API de acceso estos datos irán directamente a nuestro AWS Lambda el cual está administrando nuestros recursos, luego de pasar por AWS Lambda y activar nuestros recursos para correr nuestro modelo, pasamos a ejecutar los parámetros de nuestro modelo con los datos de entrada para poder así generar una predicción y almacenarla en un S3 bucket, a la vez dar respuesta de los datos introducidos en función de nuestro modelo