**Propuesta de Arquitectura en la Nube.**

Para la solucion de este numeral se comienza con la hipótesis de que el modelo ya esta previamente entrenado y listo para inferir.

**Almacenamiento, S3.**

Para el almacenamiento de datos se usa el servicio de Amazon S3 esta proporciona una escalabilidad masiva y una alta disponibilidad, lo que permite almacenar grandes volúmenes de datos, acá estarán a disposición las bases listas para el procesamiento y trabajos ETL

**Procesamiento de datos y entrenamient, AWS glue**.

Para los trabajos ETL y entrenamiento del modelo se usará AWS glue el cual es un componente administrado que permite el escalado automático para procesar grandes volúmenes de datos lo que la hace altamente escalable, con este se construirán los pipelines necesarios de ETL de informacion que luego irán a la parte de la inferencia. También es posible entrenar el modelo en Amazon Sage Maker.

**Calificación y despliegue, Amazon SageMaker.**

Para la calificación del modelo se hace uso de SaegeMaker, este permite hacer inferencia por lotes lo que es adecuado para el proceso fuera de línea cuando existen grandes cantidades de datos disponibles por adelantado y no es necesario un punto de conexión persistente.

**Orquestacion, Airflow:** Servicio administrado con mucha versatilidad en su desarrollo ya que es Opensource por lo tanto hay mucha disponibilidad de informacion para ser consultada. Permite definir flujos de trabajo complejos que coordinan la ejecución de diferentes servicios de AWS repartidos en diferentes task, además de permitir la ejecución de flujos de trabajo de forma paralela lo que la hace una herramienta eficiente y optima.

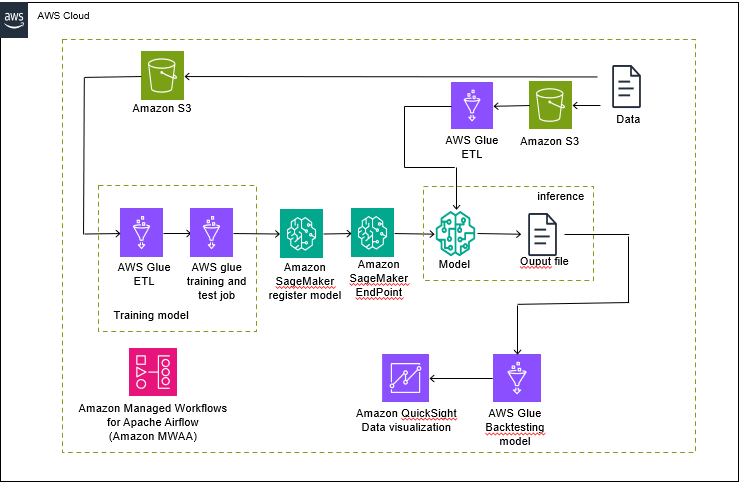
Esta arquitectura permite alta escalabilidad y confiaibilidad de varias maneras:

Utilizando servicios completamente administrados como Amazon glue, Sagemaker y Airflow, se elimina la necesidad de administrar la infraestructura ya que amazon escala automáticamente por nosotros dependiendo de la necesidad de los recursos para cada tarea.

Amazon S3 proporciona una alta disponibilidad y durabilidad para el almacenamiento de datos.

Airflow garantiza que todo el proceso quede orquestado y se ejecute de manera confiable.

**Step-by-Step**



\*Esta arquitectura se desarrolla teniendo como supuesto que el disparador de entrenamiento del modelo se da cuando hay nueva data de entrada disponible para entrenar.

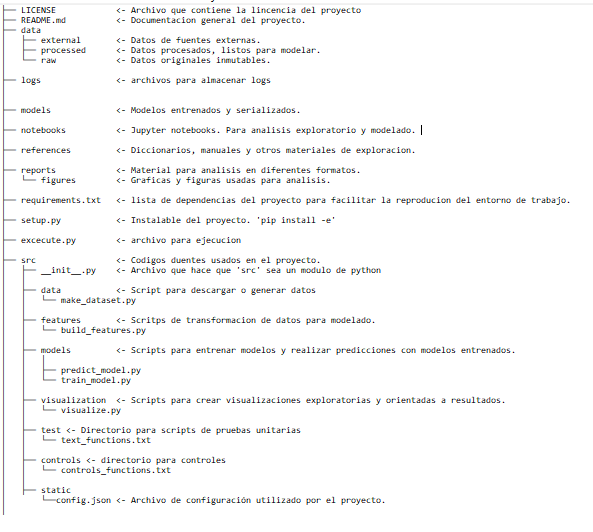
La componente de orquestación juega un papel muy importante en la ejecución del pipeline de ML al coordinar y gestionar los diferentes pasos del proceso.

* Orquesta la secuencia de pasos, asegurando que cada paso se ejecute en el orden correcto y que los datos se transmitan entre ellos de manera adecuada.
* Facilita la escalabilidad y la automatización del pipeline, asegurando que los recursos de AWS se utilicen de manera eficiente según sea necesario para cada tarea.
* Proporciona visibilidad y control sobre el estado y el progreso del pipeline, lo que facilita la monitorización y la resolución de problemas.

**Estructura de Directorios**

**\***La estructura que propone a continuación considera como supuesto un proyecto analítico que va desde, la preparación de los datos hasta la entrega de n modelos analíticos. Se hace el supuesto que la fase EDA ya termino.

La estructura que se propone a continuación toma se basa en la estructura de proyecto CookieCutter Data Science, con algunas modificaciones.



Con esta estructuración muy estandarizada de carpetas podemos comenzar a desarrollar nuestro proyecto de forma limpia y lógica, dando un lugar específico para cada uno de los archivos que se van generando a medida que el proyecto va creciendo. Esta estructuración estándar permite que los desarrolladores nuevos puedan entrar a comprender el proyector sin tener que profundizar en extensa documentación. También da la posibilidad al desarrollador de no tener que leer el código al 100% antes de saber buscar cada componente.

Para manejar el manejo de versiones en la nube del proyecto se usará un versionamiento semántico.

El versionamiento semántico está compuesto por 3 dígitos separados por puntos que representan la versión publicada. Por ejemplo: 1.1.2, 10.2.3, 20.0.0. El primer dígito se conoce como Mayor, el segundo como Minor y el tercero como Patch, y todos pueden tomar valores entre cero e infinito.

Patch

Se incremente cuando se hacen ajustes de bugs y pequeñas mejoras que no implican nuevas funcionalidades. El paquete es compatible en uso con la versión anterior (No hay impacto en la forma como el usuario final usa el paquete).

Minor

Se incremente cuando se agregan nuevas funcionalidades y cambios menores. Esta publicación puede contener ajuste de bugs. El paquete es compatible en uso con la versión anterior (No hay impacto en la forma como el usuario final usa el paquete). Cuando este componente se aumenta, se debe resetear el Patch a cero.

Major

Se incremente cuando se hacen cambios mayores que implican que el paquete no sea compatible en uso con la versión anterior (Cambia la forma como el usuario final usa el paquete). Esta publicación puede contener nuevas funcionalidades, cambios menores y ajustes de bugs.

Cuando este componente se aumenta, se debe resetear el Minor y el Patch a cero.

Observaciones:

* La primera versión estable (que se lleva a producción) es la 1.0.0
* Las versiones con Major = 0, serán versiones inestables, es decir, aún se encuentran en proceso de validación y no deben ser usadas en producción.
* Por lo general los modelos analíticos sólo aumentarán el componente Major por encima de 1 cuando haya cambios del esquema orquestador utilizado o se deba reentrenar el modelo para llevarlo a una versión más actualizada del algoritmo.

**Para el versionamiento de archivos, modelos, reportes se le concatena la fecha del día. Se hace una suposición de que el proceso no se ejecutara mas de un día para evitar errores en lecturas de datos.**

4. Datos

Exploracion de datos y preprocesamiento.

Se selecciono un set de datos de la plataforma Kaggler, el cual se llama 'Water in Australia', este cuenta con variables meteorológicas. Este conjunto de datos contiene aproximadamente 10 años de observaciones meteorológicas diarias desde muchos lugares de Australia.

RainTomorrow es la variable objetivo a predecir. Significa: ¿llovió al día siguiente, sí o no? Esta columna es Sí si la lluvia de ese día fue de 1 mm o más.

0. Carga de dataframe.

1. Se explorará las dimensiones de nuestro dataset y se harán unas previsualizaciones.

2. Se harán algunas validaciones estadísticas de nuestras variables disponibles.

3. Se hará un análisis univarido sobre nuestra variable respuesta para mirar su distribución.

4. Haremos un analisis de tipologia de datos de cada una de nuestras variables para determinar su naturaleza y validaremos nulidad.

5. Se realiza una validación de cardinalidad de las variables categoricas.

6. se determina la estrategia de codificación segun la naturaleza de la variable, ya sea en one hot encoding o un decode binario para asi eliminar nulidad.

7. Validacion de Outliers.

8. Validacion de imputaciones necesarias y técnica de imputacion.

9. Detección de Outliers.

10. Codificación de pipeline.

- El 10 se crearán en pipeline, para dejar la base lista para modelación.

**TODO ESTE PROCESAMIENTO SE ENCUENTRA EN EL NOTEBOOK LLAMADO ‘WATER\_AUS’**

**EL DICCIONARIO DE DATOS SE ENCUENTRA ADJUNTO EN EL ARCHIVO ‘METADATA.XLSX’**

**Trace el modelo de datos conceptual y explique la selección del mismo.**

Modelo de datos: se simulo la creación de la sabana con un modelo de datos tipo estrella, donde el insumo final se nutre de diferentes tablas y estas a su vez alimentada por la informacion de sensores de diferentes categorías. Este modelo captura los datos de los sensores en paralelo y al final alimenta una sola tabla con la informacion consolidada, se definió como Primary Key en todos los insumos el ‘Date’ y la ‘City’.

-Modelo muy usado en la construcción de datawarehousing

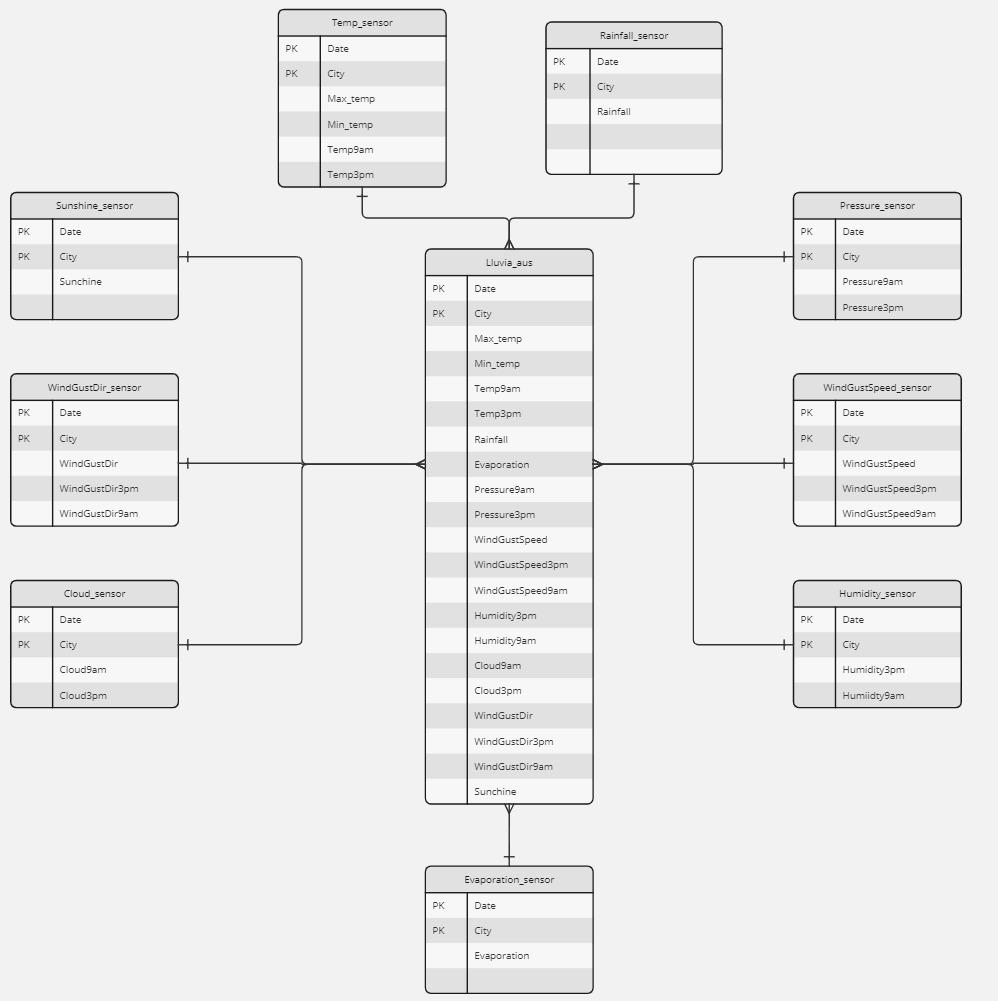
-Simplicidad y comprensión gracias a su estructura centralizada.

-Rendimiento y consulta eficiente debido a su diseño simplificado

-Flexible y escalable, es relativamente sencillo agregar otro sensor o dimensión como insumo sin afectar los demás.

-Se adapta muy bien a las necesidades del BI para reporteria y analisis.

**EL MODELO DE DATOS SE ADJUNTA EN LA IMAGEN LLAMADA ‘ENTITY RELATIONSHIP DIAGRAM.JPG’ PARA MAS CLARIDAD**



Bonus track.

Se generan un archivo .py llamado: pipeline\_preprocesado, que contiene el pipeline de transformación de datos, se hice uso de los métodos proporcionados por sklearn para la creación de las tuberías. Se definieron seis steps para todo el preprocesado, cinco de estos métodos son funciones personalizadas y convertidas a trasnsformers, el método que permite crear las variables one hot y el metodo de imputación es proporcionado por sklearn-learn,

Se creo un archivo llamado controles.py el cual contiene todos los controles hechos al dataframe resultante luego de pasar por el pipeline de transformación.

Se genera un tercer archivo .py llamado test\_rain\_aus\_fun, esta contine las pruebas unitarias creadas para testear cada una de las funciones que se usaron en el pipeline de transformación, para esto se uso la librería unittest.

Todas las pruebas necesarias para crear este archivo crearon en un notebook llamado pipeline\_de\_procesamiento.ipynb.

Entrenar el modelo de machine learning.

Se entreno el modelo siguiendo los siguientes pasos.

* Procesamiento de los datos.
* Separación en bases de test y de train a una relación 80 para train y 20 para test.
* Se creo un torneo de modelos donde se evaluó el comportamiento de un RF contra una RL, ambos modelos obtuvieron métricas similares, pero por velocidad de procesamiento se escoge la RL.
* Se procedió a crear una malla de hiperparametros para validar cuales eran los propicios para entrenar, se seleciono un max\_iter = 500, como solver = liblinear y random\_state = 0
* Se validan métricas tales como, precisión, recall, f1-score.
* Se evidencia que el modelo tiene métricas aceptables y no presenta overfiting.

Precision model\_test: 0.8528

Recall\_test: 0.5093561687281953

Precision model\_train: 0.8485

Recall\_train: 0.5019357866332955

Auc model\_train: 0.7244

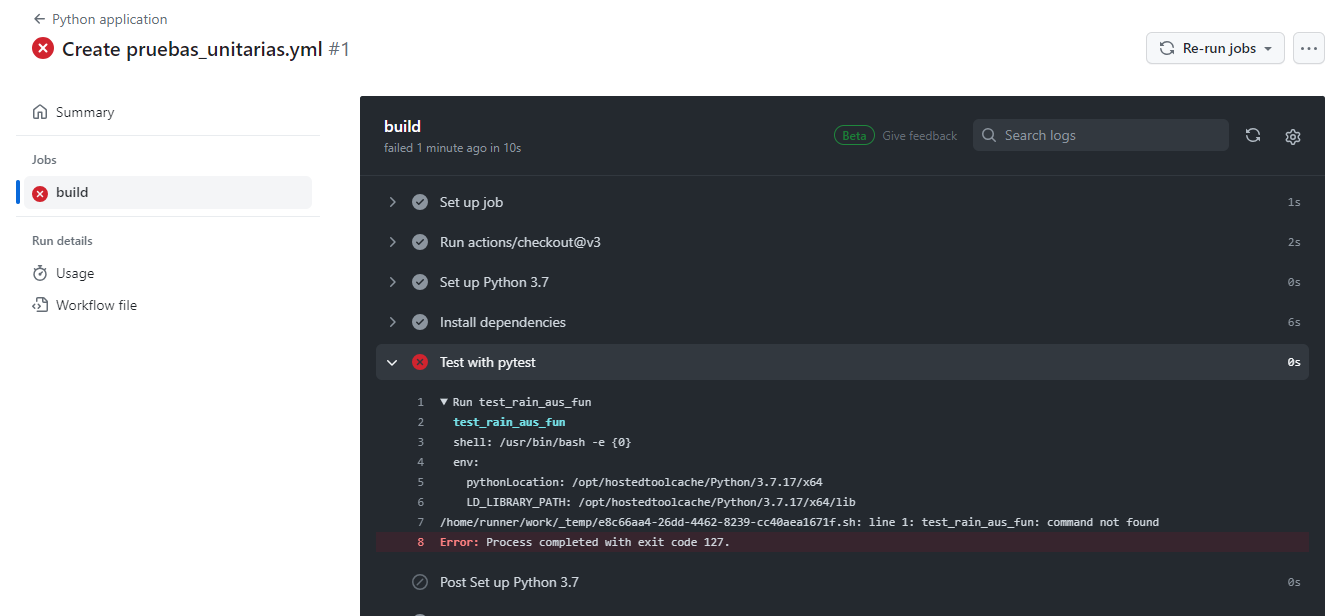
Auc model\_train: 0.7291

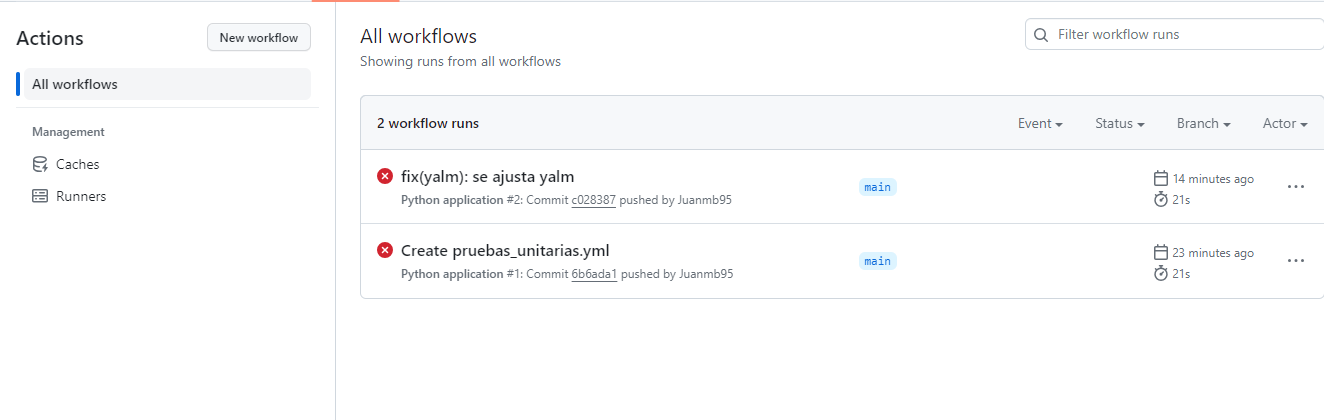
* Se genera el pkl del modelo.

**TODO EL MODELADO SE ENCUENTRAN EN EL NOTEBOOK DE PIPELINE DE PROCESAMIENTO.**

**PIPELINE CI/CD/CE.**

Se crea un pipeline en github actions que ejecuta nuestras pruebas unitarias a las funciones.





Primer intento en un error en el archivo .yalm el cual se llama ‘Create pruebas\_unitarias’

Explique cómo se ejecutarı́a el pipeline de entrenamiento continuo.

Respuesta para nuestro modelo.

Según la arquitectura propuesta, el pipeline de entrenamiento continuo dependerá de la disponibilidad de la informacion para entrenamiento, se debe hacer un monitoreo continuo de la ruta donde llegan los datos de entrenamiento, en caso de que el monitoreo detecte data nueva se accionara un trigger que permitira desplegar todo el pipeline de preprocesado de datos, una vez la data haya pasado todos los test de calidad de datos este disponible para modelamiento, se implementara el mismo algoritmo de predicion con el que se construyo el modelo inicial junto con los mismo hiperparametros escogidos para desplegar un modelo nuevo que reemplazara el que se encuentra e producción.

**PROPUESTA DE MONITOREO.**

Para el monitoreo del modelo se hace uso de métricas como la AUC, precisión, recall y f1 score

AUC : Es una medida de la capacidad de discriminación de un modelo de clasificación, cuanto mayor sea el AUC, mejor será la capacidad de discriminación del modelo.

Precisión: Mide la exactitud de las predicciones positivas realizadas por el modelo.

Recall : es la proporción de verdaderos positivos (TP) sobre el total de casos positivos reales en los datos (TP + FN). Muy importante cuando es mas necesario predecir los unos que los ceros.

F1-score : El F1-score es la media armónica de precisión y recall, calculada como 2 \* (precision \* recall) / (precision + recall). Es una medida de equilibrio entre la precisión y el recall.

**Para la calidad de datos:**

Para configurar alertas para cambios y problemas en la calidad de datos en el modelo implementado.

1. Definir criterios de calidad de datos: Antes de implementar alertas, es importante definir los criterios de calidad de datos que serán monitoreados. Estos criterios pueden incluir la precisión del modelo, la estabilidad de las predicciones a lo largo del tiempo, la distribución de los datos de entrada, la calidad de los datos de entrenamiento, etc.

2. Establecer umbrales de alerta: Una vez que hayas definido los criterios de calidad de datos, establece umbrales o límites que indiquen cuándo se activará una alerta.

3. Implementar monitoreo automático: Utiliza herramientas de monitoreo automático para evaluar continuamente la calidad de los datos y el rendimiento del modelo en tiempo real.

4. Configurar alertas: Configura alertas para que se envíen cuando se detecten cambios o problemas en la calidad de datos según los criterios y umbrales establecidos.

5. Monitorización continua y ajuste.

Para alertas y acciones para una posible degradación del modelo: se debe implementar un backtesting automático cada vez que el modelo realice inferencias y definir umbrales de aceptabilidad de la predicción. En caso de que el modelo se encuentre degradado se debe realizar un reentrenamiento de este con un set de datos mas actualizados, ya que una de las posibles causas de degradación es el cambio en las distribuciones de las variables características. Para evitar una rápida degradación se debe implementar un modelo que a medida que entren datos nuevos se accione un trigger que permita el entrenamiento continuo de este, tal como se planteo en la arquitectura en nube.

**Debido a cuestiones de tiempo no se realizan el despliegue del modelo y pipeline.**