Universidad Europea VALENCIA

GRADO EN CIENCIA DE DATOS

Desarrollo de una aplicación para el despliegue de infraestructura cloud en entornos de producción basados en AWS y Snowflake

Presentado por:

MIGUEL SANTIAGO PÉREZ

Dirigido por:

RONAL MURESANO

CURSO ACADÉMICO 2024-2025

Indice

Índice de figuras

Abstract

En el entorno digital actual, los datos se han convertido en un recurso esencial para las empresas. A medida que las organizaciones dependen cada vez más de infraestructuras de datos complejas para tomar decisiones, innovar y operar eficientemente, también crece la necesidad de contar con una infraestructura en la nube confiable, escalable y resiliente. Sin embargo, su gestión suele implicar procesos complejos, múltiples equipos y esfuerzos manuales que pueden derivar en fallos de cumplimiento, interrupciones del sistema y altos costes operativos.

Este proyecto busca abordar estos desafíos mediante una solución software que facilite el despliegue y la gestión de infraestructura en la nube. Se basa en principios de automatización, estandarización y gobernanza para reducir tareas manuales, mejorar la eficiencia y asegurar el cumplimiento normativo. La solución propuesta actúa como un componente clave que fomenta la colaboración y el uso de estándares comunes durante todo el ciclo de vida de la infraestructura, desde el despliegue hasta su desmantelamiento.

Más allá de ser una herramienta, esta iniciativa representa un cambio cultural hacia una gobernanza proactiva, con enfoque en modularidad, transparencia y automatización. Permite a los equipos tecnológicos avanzar más rápido sin sacrificar calidad, seguridad ni cumplimiento. El proyecto sienta las bases para un modelo sostenible de gestión de infraestructura en la nube, adaptable a las necesidades cambiantes de las organizaciones y esencial para el futuro de las empresas digitales.

INTRODUCCIÓN

1 Evolución del procesamiento de datos en la nube

En este último periodo, el volumen de datos que manejan las empresas tecnológicas ha crecido de manera exponencial, de este modo ha emergido una digitalización masiva que se ha extendido como una mancha de aceite en todos los sectores del sector empresarial. Las empresas se han dado cuenta que la digitalización es un propulsor estratégico clave incluso para sectores típicamente arcaicos que se han modernizado para seguir con los nuevos estándares, sin embargo, estas empresas se han encontrado con una serie de problemas fruto de esta masificación de la información como la falta de gobernanza y escalabilidad que acarrea esta escalada en los volúmenes de datos.

En una primera instancia las organizaciones recurrían a entornos *on-premise*, que presentaba las limitaciones ya conocidas pero que hasta aquel entonces no eran un punto crítico, altos costes en infraestructuras, alto coste de mantenimiento y poca o nada flexibilidad. Mas adelante se consolidaron proveedores de infraestructura *cloud* como *Amazon Web Services*, *Microsoft Azure* y *Google Cloud Platform* que ofrecían una elasticidad casi infinita y asumían los costes operacionales del mantenimiento de la infraestructura cuando no tenía demanda.

Lo revolucionario de estos proveedores es la capacidad de descargar a sus clientes y presentando una mejora competitiva debida sobre todo a la capacidad que se le otorgaba a las empresas de menores recursos de competir, ya que se ahorra la inversión inicial de capital en infraestructura que puede ser punto de corte para empresas no consolidadas.

Fruto de esto nace una trasformación radical en la forma en la que se gestionan los datos, el enfoque cloud da la vuelta a como se enfoca el ciclo de vida del dato, la gobernanza, la agilidad, el trabajo bajo demanda, optimizaciones de rendimiento y costes operativos pasan a ser una bandera de esta nueva forma de entender el ciclo de vida del dato.

En este contexto, la infraestructura *cloud* es la evolución directa de los enfoques tradicionales de procesamiento basados en procesamientos y data *warehouse* estáticos, los enfoques más modernos se basan en servicios de data *warehouse serverless*, pipelines orquestados y plataformas integradas como *Snowflake*, *Databricks* o *BigQuery*.

Estas soluciones facilitan el acceso y el análisis de datos con escalabilidad masiva además de poder obtener versionado automático de los datos, procesamiento en tiempo real o linaje de datos, que son un must en organizaciones que tienen la gobernanza como un pilar de su ejercicio.

Uno de estes avances, en este caso funcionabilidades de *AWS*, con los servicios *AWS Glue* y *Athena* que integran de forma rápida y con la menor configuración posible datos en repositorios *cloud* y se les permite ejecutar consultas SQL directamente.

Paralelamente tenemos *Snowflake* que presenta una solución de almacenamiento revolucionaria que mejora mucho los tiempos a nivel de cómputo de *queries* mediante lógicas internas de agregaciones que son muy necesarias en entornos super masificados con *datasets* que presentan volúmenes muy grandes y casi imposible de gestionar de otra manera.

Este cambio de paradigma en cuanto a escalabilidad en el ciclo de vida del dato ha traído consigo un sinfín de desafíos relacionados con la automatización, el versionado de los *datasets*, trazabilidad, gobernanza, limpieza de los *datasets*. Y bajo este marco es esencial contar con elementos integradores que garanticen todos los estándares de gobernanza.

2 Relevancia actual de los entornos de *Big Data*

Ahora mismo los entornos de Big Data desempeñan un papel estratégico fundamental en las decisiones empresariales, así como una ventaja competitiva extraordinaria aportada por la capacidad de extraer valor de grandes volúmenes de datos en sectores claves como banca, salud, telecomunicaciones o logística.

Sin embargo, los retos que plantean en los entornos Big Data son tremendos ya que estos que empiezan siendo experimentales se consolidan necesitan cumplir unas condiciones mínimas en términos de robustez escalabilidad seguridad y gobernanza, garantizando disponibilidad continua integridad de los datos trazabilidad de los procesos y cumplimiento

normativo (como GDPR o ISO 27001), lo que eleva la exigencia tecnológica y organizativa de forma brutal.

Pero sin embargo las posibilidades son ilimitadas procesamiento en tiempo real o *batch*, *machine learning*, *dashboards* en tiempo real, pipelines automatizados linaje de todo el ciclo de vida del dato desde su origen hasta su explotación final etc. La automatización es clave no solo para la productividad en el día a día de los ingenieros de datos sino también para poder soportar una demanda elástica de procesamiento masiva de datos.

Sin embargo, el despliegue y mantenimiento de la infraestructura *cloud* no está exento de desafíos, la diversidad de herramientas, tanto existentes como emergentes y el paralelizamiento que sufren los equipos de desarrollo de datos entre el mantenimiento de los *datasets* en producción el desarrollo de nuevas funcionalidades tanto técnicas como de datos, introduciendo nuevas fuentes, nuevos modelos o lógicas de procesamiento que puedan mejorar la calidad, los controles o eficiencia de recursos implica una carga operativa que necesita ser atenuada mediante la estandarización y el diseño de herramientas que faciliten el monitoreo y la gobernanza de los recursos desplegados.

En este contexto, es muy útil dedicar tiempo a la automatización de tareas repetitivas que garanticen el cumplimiento de buenas prácticas y aceleren el *time-to-market* se vuelve esencial para asegurar la eficiencia operativa y la sostenibilidad en el tiempo de los ecosistemas Big Data.

3 Motivación, necesidad de control y estandarización en infraestructuras cloud

Con la adopción de estas herramientas en las organizaciones la gestión de la infraestructura se ha tornado hacia un enfoque más dinámico debido a los grandes beneficios en escalabilidad elasticidad y reducción de costes, sin embargo, el crecimiento de los equipos y los avances en el desarrollo dentro de la infraestructura hacen que de forma natural se desalineen los enfoques de una herramienta a otra, así como dentro de la misma en otros equipos. Y aunque desde la dirección de las empresas siempre se haga un esfuerzo continuo por mantener alineado todo el ecosistema a veces se hace imposible y hace que el coste operativo a largo plazo no sea asumible y sea planteable el desarrollo de una herramienta que unifique todo y desde donde todos los integrantes de la organización apunten a la hora de gestionar la infraestructura en sus dominios.

Uno de los principales retos asociados a entornos *cloud* es la proliferación descontrolada de recursos. La facilidad con la que pueden crearse bases de datos, *buckets*, *crawlers* o entornos de ejecución, muchas veces sin una supervisión centralizada, puede llevar a una infraestructura caótica, difícil de mantener y propensa a errores críticos o vulnerabilidades de seguridad. Ante este escenario, surge la necesidad urgente de establecer mecanismos automáticos que regulen y documenten la creación, uso y eliminación de dichos recursos.

En este contexto, herramientas que permitan integrar la estandarización, el control automatizado y la gobernanza desde un único punto centralizado representan una mejora clave en el punto de vida de las organizaciones desde el punto de vista tecnológico para abordar desafíos de manera más ágil y seguras una y sobre todo más eficiente desde el punto de vista de la gestión de la infraestructura *cloud* en entornos de producción complejos y dinámicos.

Objetivos del proyecto

1 Objetivo general

El objetivo a nivel general de este proyecto es el de diseñar y desarrollar una herramienta transversal en la organización sujeta a nuevas funcionalidades que sea el centro de gestión y gobernanza sobre la infraestructura *cloud*. Desde esta herramienta se espera optimizar y automatizar los procesos relacionados con la gestión de los recursos en la nube integrando servicios de *AWS* y *Snowflakes*. El propósito es agilizar el proceso de despliegue y *desaprovisionamiento* coordinado de infraestructura mejorando la trazabilidad y la gobernanza sobre la infraestructura asegurando un entorno de producción mas limpio seguro y controlado.

2 Objetivos específicos

2.1 Despliegue de infraestructura

El primer paso para el *desplegador* es la gestión de la herramienta de despliegue y se ha de hacer de la forma mas sencilla y simple y pensando en todas las casuísticas presentes en la organización, tiene que abarcar los entornos de *AWS y Snowflake*, este objetivo implica:

- Definir el proceso completo de despliegue, desde la recolección de la información sobre que base de datos se ha de desplegar y las lógicas internas hasta la base de datos final.
- Implementar desde boto3 (librería que permite desde Python interactuar con AWS) lógicas para la creación de las bases de datos y los permisos de forma automática.
- Garantizar que el proceso de despliegue se lleva a cabo sobre todos los entornos, producción desarrollo e innovación.
- Implementar funcionalidades que permitan crear y gestionar *crawler*, nuevos o ya existentes en *AWS Glue*, y que las lógicas cuadran para no eliminar bases de datos ya existentes y que puedan llegar a no estar contempladas

3 Impacto esperado en la organización

3.1 Mejora en eficiencia operativa

Uno de los impactos clave que se espera con el desarrollo y la implementación de la herramienta *DeployDynamics* es una significativa mejora en la eficiencia operativa en el entorno de Big Data. Este impacto se logrará a través de diversos factores, entre los cuales se destacan:

3.1.1 Automatización de procesos de despliegue y gestión

La herramienta facilitará la automatización de los procesos de despliegue de infraestructura en la nube, eliminando la necesidad de realizar tareas manuales repetitivas y propensas a errores. Esto no solo reducirá el tiempo que los equipos de TI y de desarrollo dedican a estas tareas, sino que también incrementará la precisión en la configuración de los entornos y recursos. La automatización permitirá que los equipos se enfoquen en actividades de mayor valor, como la optimización de los procesos de análisis de datos y la toma de decisiones estratégicas, lo que acelerará el ciclo de vida de los proyectos y permitirá una implementación más rápida de nuevas funcionalidades y actualizaciones.

3.1.2 Optimización del mantenimiento de infraestructura

Además, de la capacidad de control y monitoreo continuo que nos aporta la herramienta permitirá realizar un seguimiento en tiempo real del estado de los recursos desplegados. Esto facilitará una detección temprana de posibles problemas de infraestructura, permitiendo una resolución más rápida de los mismos. Las alertas y los informes generados por la herramienta facilitarán la identificación de fallos en el sistema, evitando interrupciones prolongadas que puedan afectar la disponibilidad de los datos o los servicios de la empresa. En consecuencia, se reducirá el tiempo de inactividad y mejorará la disponibilidad de los sistemas, lo que se traduce en una mayor eficiencia operativa.

3.1.3 Reducción de Costes Operativos

Al optimizar el proceso de despliegue y mantenimiento de infraestructura, también se prevé una reducción de los costes operativos asociados a la gestión de la infraestructura en la nube. Las tareas manuales y los errores operativos, que suelen generar costes adicionales en tiempo y recursos, serán minimizados gracias a la automatización. Además, el proceso de *desaprovisionado* controlado de recursos, como *datasets* no utilizados, garantizará que la infraestructura se mantenga limpia y eficiente, eliminando costes innecesarios derivados de recursos infrautilizados. La optimización de los recursos permitirá a la organización aprovechar mejor sus inversiones en la nube.

3.2 Aumento en gobernanza

Otro impacto significativo de *DeployDynamics* será en el ámbito de la gobernanza de datos. En entornos de Big Data, especialmente en aquellos que manejan información sensible o crítica, es fundamental mantener un control estricto sobre la calidad y el acceso a los datos.

La implementación de esta herramienta permitirá mejorar la gobernanza en los siguientes aspectos:

3.2.1 Cumplimiento de políticas de gobernanza

La herramienta facilitará la implementación y cumplimiento de políticas de gobernanza a través de una gestión automatizada de los recursos. La integración de mecanismos de control, como el versionado de *datasets*, la gestión de roles de acceso y la creación de alertas, garantizará que los equipos operen dentro de los marcos establecidos por la organización. Esto permitirá que las políticas de seguridad, calidad de datos y accesibilidad se apliquen consistentemente a lo largo de toda la infraestructura. La herramienta también facilitará el cumplimiento de normativas y estándares regulatorios relacionados con la privacidad y seguridad de los datos, lo que es especialmente importante en entornos regulados.

3.2.2 Trazabilidad y control de cambios

El registro detallado de logs y el futuro linaje de datos proporcionará una trazabilidad completa de las acciones realizadas sobre los recursos de la infraestructura, desde el despliegue hasta el *desaprovisionado*. Esto es fundamental para el seguimiento de cambios, permitiendo a la organización identificar de manera rápida quién realizó qué cambios y por qué. Esta visibilidad completa sobre las operaciones realizadas en el entorno mejora el control sobre la infraestructura y contribuye a la responsabilidad de los equipos de trabajo. Además, el control de linaje de los datos facilita el análisis de dependencias y la identificación de causas raíz en caso de errores o fallos en los datos, mejorando la capacidad de respuesta ante incidencias.

3.2.3 Estandarización y mejora en la calidad de los datos

La herramienta también contribuirá a la estandarización de la infraestructura y los datos, lo que a su vez mejorará la calidad de la información disponible para los usuarios y sistemas de la organización. El uso de un formato de configuración unificado y la implementación de políticas de versionado de *datasets* garantizará que todos los componentes de la infraestructura sigan los mismos estándares, evitando inconsistencias y problemas derivados de la falta de alineación entre equipos o entornos. Esta estandarización facilitará la integración de nuevos sistemas y procesos, mejorando la interoperabilidad y reduciendo los errores de integración.

3.2.4 Reducción de riesgos operacionales

Gracias a la gestión controlada del *desaprovisionado* de recursos y la auditoría de todas las acciones realizadas sobre los datos, el proyecto disminuirá los riesgos operacionales asociados con la manipulación y eliminación errónea de los datos. Las alertas previas a las acciones de *desaprovisionado* y la capacidad de revertir cambios antes de su ejecución definitiva proporcionarán un nivel adicional de seguridad que garantizará la integridad de los datos en todo momento.

El impacto esperado de este proyecto será significativo en términos de eficiencia operativa y gobernanza. Al optimizar los procesos de despliegue, mantenimiento y gestión de datos en la nube, se logrará una mayor productividad en las operaciones diarias y una reducción de los

costes asociados. Además, al mejorar el control sobre los recursos y las políticas de gobernanza, la organización podrá asegurar la calidad, seguridad y trazabilidad de los datos, cumpliendo con los estándares establecidos y manteniendo un entorno de trabajo más robusto y confiable. Esto contribuirá a crear un sistema que no solo sea más eficiente, sino también más seguro y alineado con las mejores prácticas del sector.

3.2.5 Reducción de errores humanos

Los entornos de producción en la nube requieren configuraciones precisas y tareas repetitivas que, cuando se realizan manualmente, son propensas a errores humanos. Estos errores pueden ir desde la configuración incorrecta de recursos, la asignación inapropiada de roles o la eliminación accidental de datos, hasta la integración incorrecta de servicios.

La automatización del proceso de despliegue y gestión de infraestructura que propone *DeployDynamics* reduce significativamente estos riesgos. Al estandarizar los procedimientos y requerir menos intervención manual, se minimiza la probabilidad de errores causados por omisiones, confusión o malentendidos. Además, la configuración unificada en y los mecanismos de validación preconfigurados, garantizarán que todos los recursos se desplieguen y gestionen bajo los mismos estándares y reglas organizativas. Esto no solo reduce los errores en la configuración inicial, sino que también asegura la coherencia a lo largo de todo el ciclo de vida del recurso.

3.3 Sostenibilidad y escalabilidad de la solución a futuro

El desarrollo de una solución para la gestión y el despliegue de infraestructuras en entornos *cloud*, como la propuesta en *DeployDynamics*, no solo debe abordar las necesidades actuales, sino también estar diseñada para mantenerse funcional y eficiente a medida que las tecnologías, los requisitos y los volúmenes de datos evolucionan con el tiempo. La sostenibilidad y la escalabilidad son dos de los aspectos clave que garantizarán que esta herramienta siga siendo útil y efectiva en el futuro.

3.3.1 Sostenibilidad de la solución

La sostenibilidad de una solución tecnológica no solo se refiere a su capacidad para operar a lo largo del tiempo, sino también a su capacidad para adaptarse a las transformaciones del entorno. En el contexto de *DeployDynamics*, la sostenibilidad se logrará mediante varias estrategias:

- Adopción de Estándares Abiertos: La solución se diseñará utilizando estándares abiertos, lo que facilita su integración con otros sistemas y servicios a medida que surjan nuevas tecnologías o requisitos. La modularidad de la arquitectura permitirá que nuevas funcionalidades se puedan incorporar sin afectar a las operaciones existentes, lo que proporciona flexibilidad para adaptarse a futuras necesidades sin necesidad de rediseñar el sistema.
- Actualizaciones y Mantenimiento Continuo: Para asegurar que la herramienta se mantenga actualizada con las últimas innovaciones en la tecnología de la nube (como

nuevas versiones de *AWS* o *Snowflake*), se implementará un plan de mantenimiento y actualizaciones periódicas. Este plan permitirá que la solución siga operando con la eficiencia esperada y que se adapten nuevas características o mejoras conforme los proveedores de la nube evolucionen.

- Optimización de Recursos: La solución está diseñada para optimizar el uso de los recursos en la nube, lo que contribuye no solo a reducir costos sino también a minimizar el impacto ambiental. Al *desaprovisionar* recursos no utilizados y evitar el uso excesivo de capacidades innecesarias, *DeployDynamics* ayudará a las organizaciones a operar de manera más eficiente y sostenible, alineándose con las tendencias actuales hacia infraestructuras más verdes.
- Control de la Gobernanza y Seguridad a Largo Plazo: A medida que las regulaciones de privacidad de datos y las políticas de gobernanza se vuelven más estrictas, la herramienta mantendrá un alto nivel de cumplimiento con las normativas vigentes o cualquier otra normativa empresarial. Esto permitirá que la organización no solo sea más eficiente, sino también que siga operando dentro de un marco de gobernanza claro y cumpla con las políticas de seguridad y privacidad a largo plazo.

- ODS 9: Industria, innovación e infraestructura

Este proyecto aporta múltiples beneficios estratégicos para las organizaciones modernas. En primer lugar, fomenta la innovación tecnológica al automatizar procesos de despliegue e integración continua mediante herramientas actuales, lo que impulsa prácticas de desarrollo más avanzadas y sostenibles. Asimismo, optimiza las infraestructuras digitales, facilitando la gestión eficiente de grandes volúmenes de datos, reduciendo errores manuales y acortando significativamente los tiempos de despliegue.

Además, impulsa la transformación digital al facilitar la adopción de arquitecturas en la nube, promoviendo entornos más resilientes, ágiles y escalables. Por último, la automatización contribuye directamente a la eficiencia operativa, minimizando la intervención humana en tareas repetitivas, lo que no solo reduce el uso ineficiente de recursos, sino que también mejora el rendimiento energético del ecosistema digital.

- ODS 8: Trabajo decente y crecimiento económico

Facilita procesos de TI más productivos y menos propensos a errores.

3.3.2 Escalabilidad de la Solución

La escalabilidad es un componente crítico para garantizar que la herramienta pueda crecer a medida que las necesidades de la organización cambian o aumentan. En el contexto de *DeployDynamics*, la escalabilidad se aborda de las siguientes maneras:

- Integración con Nuevas Tecnologías y Proveedores: A medida que surjan nuevas tecnologías en el mundo del Big Data, el sistema será capaz de integrar estas herramientas y servicios de manera fluida, ampliando sus capacidades sin necesidad de rediseñar la arquitectura existente. Este enfoque modular y abierto facilita la incorporación de nuevos proveedores y servicios a medida que la infraestructura de la nube y las herramientas de procesamiento de datos evolucionan.

- Capacidad de Manejar Aumentos de Tráfico y Datos: A medida que las empresas crecen y generan más datos, *DeployDynamics* se adaptará al crecimiento del volumen de información. La infraestructura *cloud* permite un crecimiento en paralelo, lo que significa que la herramienta podrá seguir manejando grandes cantidades de datos a medida que se expande el entorno de Big Data sin comprometer su rendimiento ni la velocidad de los procesos de despliegue y gestión.

- Soporte para Diversos Entornos: La herramienta está diseñada para poder ser lanzada desde diversidad de entornos, lo que la hace aún más escalable. Con la posibilidad de trabajar tanto con Jenkins como con otras plataformas de nube o entornos locales, la herramienta podrá adaptarse a las necesidades de las empresas que operan en entornos más complejos, permitiendo un crecimiento fluido de la infraestructura sin barreras.

3.3.3 Beneficios de la Sostenibilidad y Escalabilidad

- Adaptabilidad a Futuro: La flexibilidad de la herramienta, al basarse en estándares abiertos y ser modular, asegura que la solución pueda evolucionar conforme surjan nuevas necesidades o tecnologías en el mercado.

- Optimización de Costes y Recursos: La escalabilidad permite a las organizaciones ajustar sus recursos a medida que crecen, evitando gastos innecesarios al consumir solo lo que se necesita. Además, la optimización de la infraestructura garantiza un uso más eficiente de los recursos.

- Cumplimiento Continuo con Normativas: La sostenibilidad en términos de gobernanza y seguridad asegura que la solución pueda seguir siendo compatible con nuevas regulaciones y políticas, minimizando riesgos de incumplimiento.

- Capacidad de Crecer con la Empresa: Al permitir que la herramienta se adapte a los aumentos en los datos y las demandas operativas, las organizaciones podrán mantener una infraestructura eficiente a medida que se expanden.

3.4 Desaprovisionado de Datos

El siguiente paso es gestionar el *desaprovisionamiento*, la eliminación de bases de datos antiguas que ya han sido deprecadas y la gestión de fuentes de datos no conocidas. Parece una tarea sencilla pero no es trivial, ya que en muchos casos dentro de la infraestructura existe mucho ruido que puede ser o no relevante y como gestionarlo ha de ser revisado desde muchos puntos de vista para que sea útil de forma transversal.

- Eliminar de manera segura y controlada los elementos dentro de los *crawlers* que no se corresponde con las bases de datos que debieran estar en producción.
- Implementar un mecanismo de alerta previa que notifique a los usuarios de las acciones de *desaprovisionado*, con un periodo de retención (de una hora) para permitir revertir la eliminación si es necesario.
- Hay que asegurar que el *desaprovisionado* se realice de acuerdo con las políticas de gobernanza establecidas, evitando eliminaciones accidentales de recursos críticos.

3.5 Automatización y gestión de Logs

La herramienta se orquestará desde Jenkins, desde ahí se indicará los parámetros clave para el despliegue y se mantendrá un control del histórico de las ejecuciones, es necesario llevar a cabo un registro mediante logs que indique que como y cuando se han llevado a cabo proceso de aprovisionamiento y sobre todo de *desaprovisionamiento* que puedan llevar a la eliminación de bases de datos claves en producción, sobre todo para tener trazabilidad sobre la infraestructura:

Marco teórico, Entorno Tecnológico

El entorno tecnológico sobre el que nos vamos a mover cuenta con una serie de herramientas de diferentes proveedores que ya están siendo utilizadas en la organización, el desarrollo y la aplicación de estas tecnologías en la organización no son objeto de este trabajo, pero si el entendimiento y la integración de estas.

1 Amazon Web Services

AWS es considerado como uno de los proveedores de infraestructura *cloud* más consolidados en sector, no solo por su variedad de herramientas si no también por lo integradas que están estas entre sí, en este proyecto nos centraremos en tres servicios claves para el desarrollo del proyecto *Amazon S3, AWS Glue* y *Amazon Athena*, aunque un par más será necesario para la puesta a punto del mismo.

1.2 Amazon S3 (Simple Storage Service)

Amazon S3 es el servicio de almacenamiento que estaremos utilizando, es el sistema de almacenamiento completamente escalable y que se utiliza como data *warehouse* en toda la organización y donde están centralizados los *datasets* separados por dominios, lo útil de s3 es su fácil integración con el resto de servicios además de con *Snowflake* que mediante *triggers* es capaz de avisar a *Snowflakes* de que particiones nuevas de *datasets* ya existentes se has subido y estos se copian automáticamente a *Snowflakes*.

1.3 AWS Glue

Aws Glue es una herramienta completamente de integración entre los datos de s3 y la base de datos de *Athena*, desde aquí se permite crear *crawlers* que tienen en los metadatos la información sobre que rutas tienen que ir en según que base de datos para su posterior *explotamiento* y desde *Glue* se permitirá crear y gestionar catálogos de datos, un componente esencial para el versionado y trazabilidad de los *datasets* en entornos de Big Data.

1.4 Amazon Athena

Amazon Athena es un servicio interactivo que monta un entorno *SQL* en el *cloud* que permite hacer consultas a los datos que se encuentran en s3 mediante a *AWS Glue* como se ha comentado antes, lo mas reseñable es la velocidad con la que se pueden añadir nuevas tablas y eliminarlas de forma super simple en entornos de desarrollo y sobre todo cuando estamos hablando de *volumnenes* muy considerables de datos.

En la organización, *Athena* será considerado como fuente de verdad de todos los datos que se tienen en *s3* y sobre las cuales se aplican soluciones de BI como *DACs* o *Dashboards* en nuestro caso desde *PowerBI* que sirven desde validaciones de la calidad de los datos, paralelas siempre a los controles de calidad automáticos, como para, comparar con los datos que se les sirve a los clientes desde *Snowflake*, que estos estén alineados, así como validaciones de negocio.

1.5 IAM (Identity and Access Management)

IAM es la plataforma de seguridad de AWS, desde aquí se controla el acceso a toda la infraestructura mediante roles y permisos que no solo utilizaran los usuarios, sino que también se aplica a los *crawlers* y bases de datos maquinas etc. Así que toda la herramienta no tiene sentido si los *crawlers* que componen un pilar en gestión de la infraestructura no tienen permisos para acceder a los datos, tener un control exhaustivo de que acceso tiene exactamente que cosa no solo es crucial a nivel de gobernanza sino a nivel operativo un error en los permisos de una maquina de desarrollador un rm -rf * o una ruta mal formateado en una

sincronización puede poner en peligro la integridad de la organización, y acabar con el ejercicio al borrar datos claves que son muy costosos de generar.

1.6 EC2 (Elastic Cloud Compute)

EC2 es la solución de aprovisionamiento de máquinas para el procesamiento de datos, no será fundamental para este proyecto sin embargo el proceso de despliegue será lanzado desde una maquina de EC2 así que será necesario conocerla y tener en cuenta como hacerlo de forma orquestada y optimizada lo que se verá más adelante.

1.7 Separación *multi-entorno*

Las herramientas propias de AWS no es lo único a tener en cuenta ya que en la organización se tiene una separación en varios entornos desconectados prácticamente entre si por motivos de permisos y gobernanza, y existen ciertas políticas a tener en cuenta cuando hablamos de que puede hacerse y que no puede hacerse en función del entorno en el que estamos trabajando y sobre todo en este caso que infraestructura estamos o no estamos desplegando.

Esta diferenciación entre entornos nace de una necesidad para controlar que la limpieza de ciertos entornos limitando las pruebas a ciertos otros y manteniendo otros siempre consultables por terceros ajenos a los equipos de desarrollo, por este motivo procederé a explicar las limitaciones, características y facilidades que presentan cada entorno.

- Entorno de Producción (*prod*): Contiene los datos y servicios que están en funcionamiento y disponibles para terceros, son datos considerados como canónicos que son aquellos que el equipo dueño de estos presenta para toda la organización y que se compromete a mantener con una calidad positiva. Se considera el entorno más crítico y se aplican medidas estrictas de seguridad, trazabilidad y control de cambios. Además, es el único entorno donde se mantienen los datos, el resto de los entornos están por triplicados ya que no supone un coste extra, pero sin embargo el coste de almacenamiento no es trivial, aclaración: las bases de datos están presentes en todos los entornos, pero los ficheros solo se encuentran en producción.
- Entorno de Desarrollo (*dev*): En el entorno de desarrollo es donde se tienen todos los datos incluyendo algunos datos intermedios que se usan para validaciones de datos en producción así como desarrollo de nuevas funcionalidades o testeos, es un entorno reservado para desarrollos del propio equipo y no presenta datos a terceros, sin embargo es el que puede tender a desalinearse más con lo que debería ya que terminados algunos desarrollos todas las bases de datos temporales deberían eliminarse y desde la aplicación de despliegue se mantendrá en orden.
- Entorno de Innovación(*innovation*): dedicada a la experimentación pruebas de otros equipos y que funciona como producción en tanto en cuanto que solo pueden haber

datasets canónicos y el equipo se compromete a mantener datos con un estándar optimo.

2 Snowflake

Snowflake es una plataforma de datos como servicio (*DaaS*) que permite el almacenamiento y procesamiento de datos de manera escalable y eficiente. Es conocida por su arquitectura innovadora y su capacidad para manejar cargas de trabajo analíticas y operativas en un único sistema. *Snowflake* se destaca por su capacidad para separar el almacenamiento y el cómputo, lo que permite escalar ambos de manera independiente, optimizando los costes y el rendimiento. En este proyecto, *Snowflake* jugará un papel clave en el almacenamiento y análisis de los datos a gran escala, trabajando en estrecha colaboración con *AWS* para proporcionar una solución robusta de procesamiento de *Big Data.*

2.1 Arquitectura en Snowflake

Snowflake es utilizado por su arquitectura *multi-cluster* compartida que permite ejecutar múltiples cargas de trabajo de manera independiente sin afectar el rendimiento. Esto hace que *Snowflake* sea especialmente adecuado para las consultas de los clientes a grandes volúmenes de datos.

También se utiliza una separación por entornos y algunos *datasets* se guardan en un entorno de *stagging* que almacena particiones temporalmente antes de pasar a los *datasets* canónicos en el caso de que los clientes puedan tener acceso directo a los datos canónicos.

2.2 Integración con AWS S3

Snowflake se integra perfectamente con Amazon S3, mediante a una serie objetos de *Snowflake* que se comentaran más adelante, básicamente cuando la ETL u otro proceso escribe datos en s3 un sistema de *trigger* avisa a *Snowflake* y inserta los datos mediante un *Snowpipes* y *Stages* de forma automáticas para gestionar el flujo de datos de S3.

2.3 Escalabilidad y rendimiento

Una de las principales ventajas de *Snowflake* es la capacidad de escalado automático bajo demanda según las necesidades y la carga de trabajo lo que permite tratar con volúmenes de datos sin preocuparte de la infraestructura que hay por debajo.

Esta capacidad de escalar de forma automática asegura que el sistema se adapte tanto a las necesidades de alta demanda como a periodos de menor actividad, optimizando los costes asociados al procesamiento de datos.

2.4 Herramientas Snowflakes

Para desplegar una base de datos en *Snowflake* de manera automática deberemos contar con una serie de herramientas que ya se utilizan en la organización pero que han de ser desplegadas desde la herramienta.

Para los próximos puntos es necesario tener el esquema de los datos que se irán a introducir ya que estos se introducirán de forma automática y de lo contrario pueden ponerse en riesgo la integridad de los mismos.

2.4.1 Snowpipes

Los *Snowpipes* son una herramienta de Snowflake diseñada para facilitar la ingesta continua y automatizada de datos casi en tiempo real. Permiten cargar datos desde archivos ubicados en *stages* (áreas de almacenamiento intermedio) hacia tablas en la base de datos, sin necesidad de ejecutar manualmente comandos de carga. Para ello, se apoyan en servicios de notificación como Amazon S3 Event Notifications, que al detectar la llegada de nuevos archivos introduce los datos. Este enfoque permite que las nuevas particiones o datos disponibles en S3 sean identificados inmediatamente y procesados por el *Snowpipe*, manteniendo la base de datos actualizada de forma continua y eficiente.

2.4.2 Stages

Los *stages* en Snowflake son ubicaciones de almacenamiento intermedio que actúan como puente entre los archivos de datos externos y las tablas dentro del sistema. Estos pueden estar alojados en el propio Snowflake (*internal stages*) o en servicios externos como Amazon S3 (*external stages*). Cada *stage* se configura para apuntar a una ruta específica, y está vinculado a una o varias tablas a las que se desea cargar información. El uso de *stages* permite desacoplar el almacenamiento del procesamiento, favoreciendo una arquitectura modular y escalable.

Ilustración 1 Stage creation

```
f"""
CREATE OR REPLACE STAGE "PROD"."DATASET".S3_STAGE_PROD_DATASET_C_VERSION
storage_integration = S3_PROD_FWK
url = 'path';
"""
```

Nota. Esta es la forma de crear un Stage

2.4.3 Databases

En Snowflake, las *databases* representan contenedores lógicos donde se agrupan esquemas, tablas, vistas y otros objetos relacionados. Cada base de datos puede organizar información

bajo distintos esquemas, permitiendo así una separación clara entre diferentes dominios de datos. Son especialmente útiles para mantener distintas versiones de un mismo conjunto de datos, por ejemplo, separando entornos de desarrollo, pruebas y producción, o bien para gestionar particiones temporales e históricas de una misma fuente.

2.4.4 Allow storage paths integration

Para que *Snowflake* pueda acceder correctamente a los datos almacenados en ubicaciones externas (como Amazon S3), es necesario registrar previamente dichas rutas en el catálogo de rutas de almacenamiento permitidas. Esta acción se realiza mediante la instrucción ALTER ACCOUNT SET ALLOWED_STORAGE_LOCATIONS, lo que garantiza que solo las rutas explícitamente autorizadas puedan ser referenciadas por los *stages*. Esta medida proporciona una capa adicional de seguridad y control sobre los datos que se pueden ingerir desde fuera de *Snowflake*, asegurando que no se establezcan conexiones no deseadas o accidentales.

3 Uso de JSON como eje de la comunicación y configuración

Actualmente en la organización existe un repositorio centralizado en *Gitlab* por el cual cada *dataset* de la compañía tiene información clave que se utiliza en algunas librerías propias como la de *DQCs* y demás que guarda alguna información relevante como metadatos, así como el esquema, métricas que se le tienen que aplicar, valores validos etc. En el mismo repositorio estará un *JSON* con información para el despliegue que solo podrá ser accedida por los Data *Quality Specilist* y que dejarán información clave sobre que rutas deberán ir a según que repositorio y demás.

3.1 Configuración dinámica y flexible

El uso de la estructura JSON permite que el sistema sea altamente configurable y flexible. Mediante archivos JSON estructurados, es posible definir de manera dinámica todos los parámetros necesarios para el despliegue de los recursos en AWS y *Snowflake*. Por ejemplo, la configuración de los *datasets*, las rutas de almacenamiento, las conexiones a servicios de bases de datos (como *Athena* y *Glue*) o los *triggers* en *Snowflake* pueden ser gestionados a través de archivos JSON. Esto facilita la creación de entornos específicos para desarrollo, innovación o producción, adaptando los parámetros y recursos de acuerdo a las necesidades de cada caso.

3.2 Versionado y trazabilidad

Otro beneficio importante del uso de JSON en un entorno Git es la capacidad para llevar un control de versiones claro de la configuración. Cada vez que se realice una modificación en el JSON de infraestructura o en los parámetros de configuración, se actualizará en la infraestructura correspondiente, lo que permite hacer un seguimiento detallado de los cambios. Esto facilita la trazabilidad, la auditoría y la reversión de cambios en caso de errores,

lo cual es crucial en entornos de producción donde los datos deben estar sujetos a estrictos controles de calidad y gobernanza.

3.3 Definición del JSON

La definición de un JSON de configuración no es algo trivial ya que esta herramienta tiene la proyección de ser un elemento transversal a toda la empresa y los detalles de estos tienen que ser aprobados por un gran número de *stakeholders* para ser realmente útiles en el día a día.

Empezare a nombrar los parámetros claves del *JSON* que se ha desarrollado y que contiene los metadatos claves para el despliegue

- Nombre: Contiene el nombre del *dataset* canónico al que se referencia.
- Versión: Versión actual del *dataset*.
- Dominio: Hace referencia al equipo que está a cargo.

3.3.1 AWS

Luego tenemos los parámetros propios de AWS

- Actualizar *Crawler*: es un booleano que le indica si el *crawler* se debe actualizar, por defecto estará en True, de no estarlo se omitirá.
- *Database*: Base de datos de *Glue* donde se introducirán las tablas que salen que crea el *crawler*.
- Descripción: Una descripción breve del entorno.
- Active: Un booleano por defecto True, de lo contrario eliminara el *crawler* en caso de que el *desplegador* se lance en modo *desaprovisionamiento*.
- Lista de rutas: Es la lista de rutas que se le han de añadir al *crawler* ya que cada ruta representa una tabla y cada ruta tiene los siguientes parámetros.
    - Descripción: Una pequeña descripción sobre la tabla.
    - Prefijo: La propia ruta al directorio donde están los datos.
    - Tipo de *dataset*: Puede ser Canónico o Intermedio y guarda información sobre si debe ir a un entorno u otro
    - Información del fichero: Información sobre el tipo de fichero que se encuentra dentro etc.
    - Nombre: Nombre de la tabla
    - Entorno: Esta información de si debe de ir a uno u otro entorno, y se puede forzar meter alguna tabla a un entorno que no debería por estándar o quitar alguna es una lista con los entornos y un booleano que indica si debe de ir o no.
    - Dependencia: Son dos parámetros dependencia interna o externa que será completamente útil para tener trazabilidad de fallos y dependencias.

3.3.1 Snowflake

- Nombre: Nombre de la tabla a desplegar.
- Entorno: El entorno puede ser *Prod*, *Dev*, *Stagging* dependiendo de donde se guarde el *dataset*.
- Esquema: Esquema que utilizara para desplegar la infraestructura.
- Prefijo: Ruta de s3 de la cual va a desplegarse la base de datos.
- Descripción: una breve descripción.

4 Jenkins

Jenkins es una herramienta de integración y entrega continuas de código abierto, ampliamente utilizada para automatizar procesos relacionados con la construcción, prueba, integración y despliegue de software.

Permite definir "pipelines" o flujos de trabajo automatizados que ejecutan tareas de forma controlada, repetible y trazable. Estos flujos pueden incluir desde la compilación de un proyecto hasta su despliegue en producción.

En el contexto de esta herramienta de despliegue, Jenkins actúa como plataforma de orquestación desde la cual los usuarios finales podrán interactuar con el *desplegador* de infraestructura. Es decir, Jenkins:

- Permite lanzar el proceso de despliegue mediante un *job* (tarea automatizada).
- Lanzar una máquina de EC2
- Ejecuta los scripts Python que gestionan los servicios en *AWS* y *Snowflake*.
- Controla el entorno de ejecución (*dev*, *prod*, etc.) y los permisos asignados.
- Muestra los logs de cada ejecución para asegurar trazabilidad y depuración.
- Facilita que el proceso pueda ser reutilizado fácilmente por distintos usuarios sin necesidad de conocimientos técnicos profundos.

Jenkins utiliza un formato *groovy* para configurarse desde donde puedes programar los pasos por los cuales se va a orquestar el proceso como vemos en la Figura y desde donde podemos hacer que se lance el proceso con unos determinados parámetros u otros así como podemos hacer que se lancen maquinas se clonen repositorios se instalen dependencias etc.

Ilustración 2 Jenkins groovy

Script

```
pipeline {
    agent none
    options {
      timestamps ()
      ansiColor ("xterm")
    }

    environment {
        // PYTHON VIRTUAL ENVIRONMENT CONFIGURATION
        PYTHON_VIRTUAL_ENV = "./venv"
        PYTHON_VERSION = "python3.11"
        PYTHON_MAIN_FILE = "main.py"
        PYTHON_INTERPRETER = "${PYTHON_VIRTUAL_ENV}" + "/bin/" + "${PYTHON_VERSION}"

        PATH = "/var/lib/jenkins/.local/bin:/var/lib/jenkins/bin:/usr/bin:/usr/local/bin"

        DOMAIN_MANAGEMENT = 'PALAWAN'

        AWS_REGION = 'eu-west-1'
        TAG_KEY = 'Domain'
        TAG_VALUE = 'PalawanDomain'

        //GIT CONFIGURATION
        GIT_REPOSITORY =                                    dp_tools/managementtechnicalgovernance.git"
        GIT_CREDENTIALS_ID =


    }
    stages{
        stage('GET EC2 INSTANCE ID FOR ETL') {
            agent {
                label "${params.EC2_LABEL}"
            }
            steps {
                sh label: 'Print node name', script: 'echo ${NODE_NAME}'
                script {
                    def instanceId = sh(
                        script: 'cat /sys/devices/virtual/dmi/id/board_asset_tag',
                        returnStdout: true
                    ).trim()
```

Nota. Ejemplo de como se crea un pipeline de jenkins

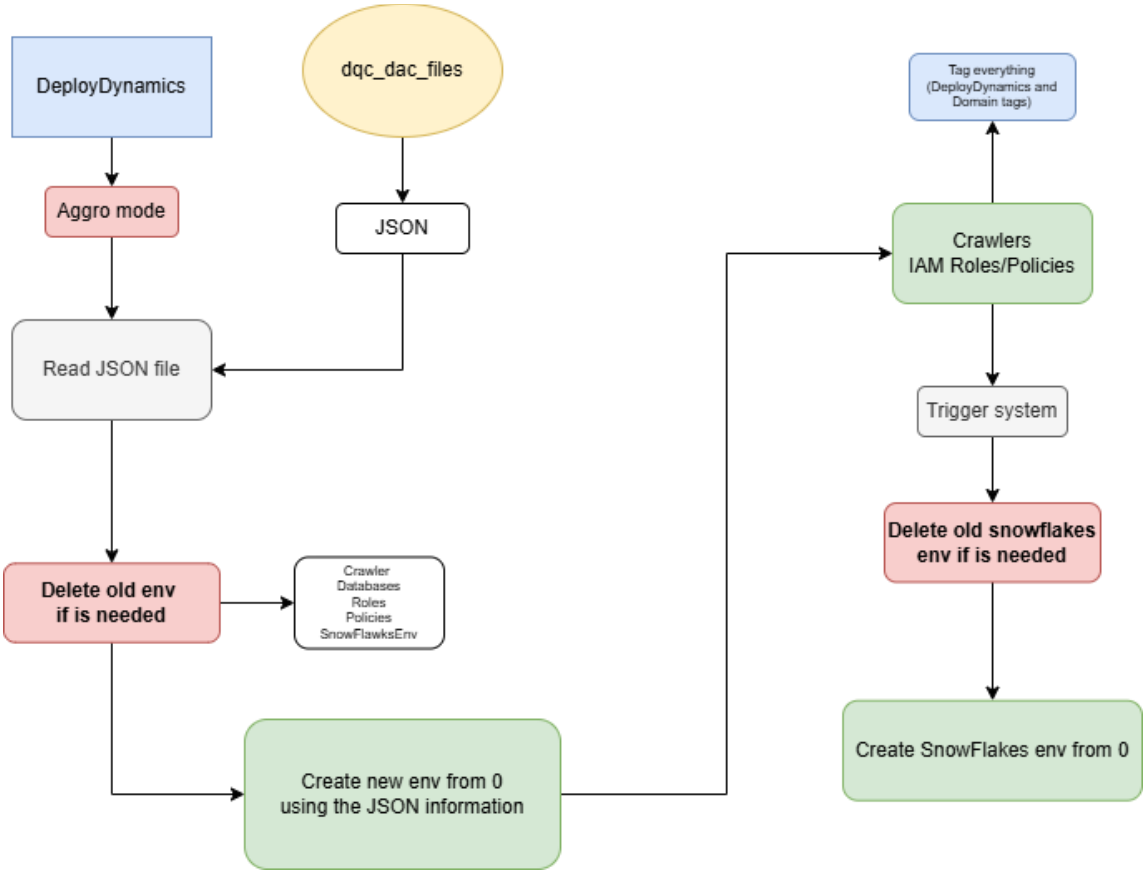
Metodología Desarrollo de DeployDynamics

1 Definición de funcionalidad general primera iteración AggroMode

El primer paso es definir una hoja de ruta lo más acotada posible que tiene que tener en primera instancia el proyecto para probar que todo lo desarrollado vaya funcionando y más adelante continuar iterando para llegar a un producto funcional.

La primera iteración se definió como Modo Agresivo, que evitaría lógicas complejas y casuísticas específicas y se centrará en tener una iteración funcional que sea capaz de desplegar infraestructura, y cuyo esquema queda reflejado en la Figura 6

Ilustración 3 First version schema



Nota. Primer esquema del proyecto

1.1 Definir y leer el JSON

Como se mencionó anteriormente la definición del JSON no es trivial porque es la única forma, aparte de parámetros de ejecución que se le pasen al *desplegador* y estos deben mantenerse simple y todo debería estar recogido en la configuración del JSON, de introducirle información a nuestro *desplegador*, y posibles lógicas internas que puedan necesitar algunos metadatos tienen que estar reflejados en el mismo y este no puede ponerse a cambiar de un momento a otro ya que un cambio en el JSON implica notificar a los interesaros y no es conveniente hacerlo de forma recurrente. Por ese motivo es necesario pararse a analizar de forma detallada todas las posibles casuísticas que fueran a ser necesarias en el futuro.

No me voy a parar a comentar todos los parámetros del JSON porque estos ya han sido explicados con anterioridad, por otro lado lo que necesitamos del *desplegador* para este paso es que se clone el repositorio de *Gitlab* donde esta centralizado el *JSON*, lo lea y parametrice toda la información recogida en él.

1.2 Crear y eliminar entorno de AWS

En este punto asumimos que cada *crawler*, base de datos, *rol*, *policie* tiene reservado su nombre es decir si yo cuento con un *dataset* llamado por ejemplo Capacidad, el nombre del *cawler* asociado, será capacidad_env, y este *env* hará referencia al entorno del que se leen los datos, para el caso del *crawler* siempre será *prod* pero para el resto de los casos será por ejemplo *RolCapacidad* o algo similar que ya estará definido previamente.

Partiendo de esto y teniendo en cuenta que el coste de eliminar y crear cosas en AWS es despreciable ya que no se eliminan datos como tal y solo consume el tiempo del nuevo crawler para recopilar todos los datos nuevos, el proceso a seguir será, eliminar todo lo que exista con los nombres reservados en todos los entornos, y crearlo de 0.

Este proceso ha de hacerse por triplicado para los tres entornos de este modo se ha de hacer una lógica de logging por entorno y replicar las acciones de uno en los tres generando para cada caso la lista con las rutas que se quieren desplegar.

1.2.1 Proceso de eliminación y creación de IAM

Aunque parezca sencillo las eliminaciones tienen que hacerse de forma controlada y para no eliminar nada que no se dese, los pasos son los siguientes:

- Identificar si existe el rol de IAM con la palabra reservada.
- Acceder a las *policies* anexadas al Rol.
- Desanexar la *policy* especifica y eliminarla: cada Rol tiene *policies* que se usan en común por todos los roles, son *policies* de acceso general y si la eliminamos la estaríamos eliminando de todos sitios.
- Eliminar el rol reservado.
- Crear la *policy* y formatearla con los *paths* necesarios.
- Crear el Rol y anexarle las *policies* necesarias.

1.2.2 Proceso de eliminación y creación de Glue

Una vez contamos con el Rol que ha de tener el *crawler* ya podemos proceder.

- Eliminar la base de datos vieja: Si bien el *crawler* viejo esta apuntando a esta base de datos, no existe ningún conflicto en eliminarlo.
- Crear la base de datos nueva.
- Eliminar la el *crawler* viejo.
- Crear el *crawler* nuevo.
    - Se ha de crear con las *sources* de los *paths* que correspondan.
    - Se tendrá que hacer que omita el patrón **.crc para que ignore los crc que son ficheros que se generan al generar *parquets* y *Athena* no puede gestionarlos.
    - Anexarles el rol que hemos creado con anterioridad.

1.3 Crear y eliminar entorno de Snowflake

Para trabajar con el entorno de *Snowflake* lo haremos mediante el conector de *Snowflake* y las acciones se imputarán como *queries* formateadas sin embargo hay algún detalle que es crítico.

Para el desarrollo de esta herramienta vamos a crear el propio entorno de rutas permitidas, ya que las rutas permitidas de producción las usa todos los *datasets* de producción y una ruta mal formateada puede dejar a toda la empresa sin conexión a *Snowflake*.

Los pasos a la seguir son los siguientes:

- Recuperar las rutas permitidas actuales
- Añadirle las rutas necesarias y subirlas
- Crear el *Stage*
- Crear la tabla de *Snowflake*
- Crear el *Snowpipe* apuntando del *stage* a la tabla

2 Desarrollo de Código

El proceso de desarrollo que queremos seguir para la elaboración de un código de calidad y con una gobernanza optima, va a ser mediante clases y una estructura ordenada para que resulte mas sencillo hacer nuevas iteraciones en el código a futuro.

Por tanto, crearemos una estructura de carpetas organizada donde tenemos la estructura estándar donde se separa el código por:

- Código
- Logs
- Configuración
- Documentación

Y puesto que el código ha de tener organizado funciones de cada tipo vamos a separar el código y las funciones por clases en función de para que sirvan.

2.1 AWS

En AWS no tenemos una clase como tal si no que contamos con una clase padre que es la encargada de inicializar la conexión y luego tenemos clases hijas del resto de herramientas que funcionan heredando las funciones de la clase padre y añadiendo solo las funciones específicas.

Hay que recalcar que se gestionan la conexión a los 3 entornos producción, desarrollo y Innovación que técnicamente son cuentas diferenciadas dentro del mismo usuario y esto se gestiona mediante un cliente *STS* y por el rol de acceso ARN en el cual no voy a profundizar en exceso

Las funciones iniciales tienen que permitirnos la gestión de la infraestructura generada anteriormente y quizás otras mejoras futuras así que las funciones generadas valdrán para todos los casos

2.1.1 IAM

Las funciones de que tenemos en IAM son las siguientes.

- Coger y actualizar la *Policy*: Recibe un rol y una lista de rutas y las añade como rutas nuevas.
- Eliminar rol y *policy*: usa una lógica compleja para eliminar el rol y las *policies* anexadas que son especifica, llama a otras funciones de desanexar y de eliminar *policy*
- Recoger ARN de la *policie*: Comprueba si existe una *policie* por el ARN de la misma que es un identificador único que tienen las *policies*
- Crear Rol IAM con *policie*:
    - Llama a la función de eliminar la *policie*
    - Una vez tiene el campo libre la crea el Rol de 0
    - Añade las etiquetas
        - *Dataset*
        - Dominio
        - Etiqueta del *desplegador* para indicar que este rol se ha desplegado con el desplegador
    - Crea la *policy* de confianza y anexa la *policy* de confianza
    - Crea la *policy* especifica y anexa la *policy* especifica
- Formateo de *policy*: las *policies* tienen que estar correctamente formateadas para funcionar y tienen estar escritas de una forma específica para que se puedan detectar los rutas con un formato tal que “arn:aws:s3:::{ruta}”
- Start: inicializa y funciona como un orquestador interno

Nota, todas las *poliecies* tienen el formato de un JSON por lo que deben estar correctamente formateadas para que cumplan con su cometido, la Figura 7 es un ejemplo de la policy de confianza que necesita el crawler y que es común para todos.

Ilustración 4 Trust policy

```
trust_policy = {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Service": "glue.amazonaws.com"},
            "Action": "sts:AssumeRole"
        }
    ]
}
```

Nota. Ejemplo de policy que se atachea a los roles

2.1.2 Glue

En Glue es donde más problemas puede haber a nivel de lógica, pero no tanto a nivel complejidad, pero esta complejidad vendrá en la siguiente versión y no tanto aquí en el modo agresivo aunque se verán mas tarde cuando se desarrolle el modo *desaprovisionamiento* de datos.

De este modo las funciones que tenemos son:

- Crear base de datos
- Eliminar base de datos
- Recoger o crear *crawler*: Mira si existe el *crawler* que estamos por crear y lo actualiza con las rutas necesarias en el caso de que ya exista, esta es una función que se utilizara luego en el modo *desaprovisionamiento* y que ya tiene una lógica interna para filtrar lo necesario pero que se explicara luego más en profundidad.
- Listar *crawlers* existentes
- Recuperar *crawler*: simplemente se carga el *crawler* en una variable, lo utilizan otras funciones

2.1.3 S3

Para finar con AWS tenemos s3 donde su funcionabilidad principal es la de instalar la función de notificaciones automáticas para avisar a *Snowflake*, es una única función llamada crear evento de notificación, se le pasa una ruta o una lista de rutas y crea un evento de notificación para cuando se inserten *parquets* en las susodichas rutas.

2.2 Gitlab

Este módulo de tiene que establecer la conexión a *GIT*, clonarse el repo y extraer la información del *JSON*, sin embargo, las maquinas sobre las que trabajamos ya tienen por defecto los permisos para *GIT* así que no será necesario.

Las funciones por tanto son:

- Clonar repositorio *GIT*: hace un *GIT* clone apuntando a la rama en la que debemos trabajar.
- Cargar el *JSON*: Lo cargo en una variable
- Extraer valores del *JSON*: Codifica en variables los valores del JSON

2.3 Snowflake

Las funciones del módulo de *snowflake* se ejecutan mediante *query* por lo tanto vamos a contar con una función que lance *queries* y el resto que formateen esas *queries*.

- Crear puntero de *snowflake*
- Ejecutar *query*
- Crear tabla
- Crear *stage*
- Crear *snowpipe*

3 Orquestación de código

Se utilizar Jenkins como modo de orquestación desde ahí se permite introducir parámetros a la ejecución desde una interfaz amigable para el usuario final, desde aquí se pueden orquestar los parámetros claves para el lanzamiento de la ejecución

Ilustración 5 Build execution

Nota. De este modo se inicializa la ejecución

En este caso los parámetros que se utilizan para construir la ejecución son los siguientes:

- Paso: El paso que se va a ejecutar, de momento solo esta activado el modo aprovisionamiento, pero esta pensado para que en un futuro se puedan introducir más.
- Tag: Es donde apunta Jenkins cuando clona el repositorio del *desplegador*.
- EC2 Label: La etiqueta de EC2 apunta a una AMI, es la maquina que se montará para que se inicie la ejecución.
- EBS Volumen Size: Tamaño del disco, se utiliza para casos donde pueda llegar a ser necesario cargarse datos y puede que en un futuro haya funcionalidades que necesiten cargarse datos, si lo dejas por defecto este paso se pasa.
- JSON_PATH: es donde se encuentra ubicado en el repositorio el JSON de configuración, en un futuro no será necesario ya que si los repositorios están donde tocan la ruta debería ser algo como configuracion/dominio/nombre/.. estandarizado para todos los *datasets*, pero de momento como primera versión y si por aluna circunstancia algún fichero no está donde toca, así nos aseguramos de que se despliega correctamente.
- DQC_DAC_REPO_TAG: Tag del repositorio donde se encuentra la información, ya que no nos interesa coger siempre el código de master ya que para testear se debe hacer en ramas separadas que están identificadas con un tag.

3.1 Pasos del job

Para ejecutarse el código correctamente necesitamos hacer que Jenkins haga los siguientes pasos:

- Cargue las variables de entorno que se pasan por parámetro
- Crear una máquina de EC2 a partir de una AMI
- Se resécala el disco en el caso de que sea necesario (No de momento)
- Se clona el repositorio
- Se instalan las dependencias alojadas en el requirements.txt
- Lanzar proceso

La Figura es un ejemplo de como se lanza el proceso desde Jenkins, los pasos de crear la máquina y escalar el disco son funciones que se usan en todos los Jobs y no son objeto de este trabajo.

Ilustración 6 Groovy step

Nota. como se lanza en la máquina una ejecución.

4 Pros y contras modelo actual

Actualmente tenemos un modelo que lo que hace en resumidas cuentas es sobrescribir lo que hay con lo que esperamos, lo cual para una primera versión es algo que esta correcto pero no es algo que no se puede considerar un producto final, ya que aunque no suponga un coste ni se vayan a perder datos como tal, puede que se eliminen algunas tablas que no se tengan en cuenta, por lo cual vamos a hacer un balance de pros y contras, una serie de alternativas a este modelo actual y como afrontamos los posibles fallos en la gestión de las diferentes casuísticas que se pueden dar.

4.1 Pros

4.1.1 Sencillez y claridad

El modelo es super sencillo y claro de entender lo que esta en el repositorio de configuración es lo que esta en el entorno *cloud*, ni mas ni menos, todas las bases de datos antiguas, de testeo o que no están contempladas desaparecen una vez se lance en proceso dando una imagen clara y concisa de lo que se tiene en cada entorno.

4.1.2 Funcionalidad útil

Si bien como único tipo de ejecución no parece lo ideal podría ser útil generar un modo que replique el modo *agresivo* para ejecuciones semanales para todos los *datasets* y limpie todo lo que durante la semana se ha ido dejando.

4.2 Contras

4.2.1 Eliminaciones indebidas

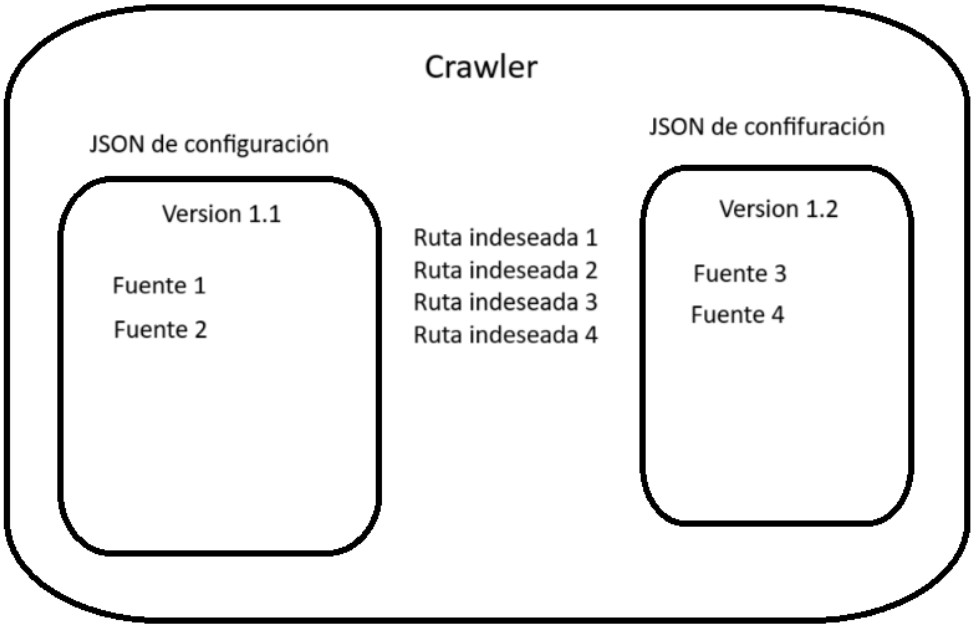
Es probable que si se trabaja con un entorno vivo y sobre todo haya ciertos casos donde se estén desarrollando cosas donde no quieras que cada viernes pierdas todo el trabajo de testeo de cada semana, o inclusive que haya varios *datasets* que no cumplan los estándares todavía y no se tengan en el registro.

4.2.2 No se alinea con los estándares completamente

El caso es que diferentes versiones de un mismo *dataset* por políticas están en el mismo *crawler*, pero en diferente JSON y las ejecuciones han de ser independientes, por lo cual un *crawler* no tiene la potestad conocer si un ruta puede estar activo por un JSON de otra versión.

Y por tanto puede hacerse imposible para el desplegador conocer si una ruta que no esta en su JSON de configuración es indeseada o pertenece a otra versión como se ve en la figura

Ilustración 7 Problem diagram



Nota. En este diagrama podemos ver el problema que puede llegar a acontecer

5 Posibles propuestas para el modo *desaprovisionamiento*

Como hemos comentado el *desaprovisionamiento* no es una tarea sencilla ni tiene una respuesta más valida que otra, al final será una funcionalidad que se adapte de una forma u otra a las políticas de la gestión de la infraestructura de la compañía.

5.1 Eliminación selectiva

Se puede añadir un parámetro que indique si un valor esta activo o inactivo en la propia configuración y que en caso de que este inactivo elimine los valores que toca y solo los que toca, de esta manera el *desplegador*, *desaprovisionara* una versión únicamente en el caso en el que se requiera.

Si bien esta es una medida buena no es suficiente, debido a que no estaría solucionando el problema de rutas extra indeseadas.

5.2 Recoger otras versiones

Esta propuesta es muy útil, pero tiene varios inconvenientes: lo ideal sería no desplegar una versión de los datos sino varias a la vez, y en el caso de que una de ellas ya exista simplemente la dejas ahí y todo el resto lo eliminas, y todo lo que esta en el crawler y en el rol de IAM es única y exclusivamente lo que hay en los ficheros de configuración de ambas versiones, sin embargo no encuentro esto demasiado útil debido a que la herramienta no está extendida y obligaría a meter todos los *datasets* a la vez, así que sería una buena implementación a futuro y no seria costosa ni por lógicas ni por complejidad.

Resultados

1 Propuesta para producto mínimo viable

Se necesita un producto que sea fácil de usar por gente que no tiene contexto y que se pueda llevar a cabo para alinear todos los *datasets* de la empresa con la herramienta y luego en siguientes operaciones actualizar las lógicas del algoritmo para hacerlas más independientes y que el ingeniero no tenga que ponerlo tan detallado, las posibles mejoras futuras se verán a continuación.

De este modo los cambios con respecto a la anterior iteración es que por defecto solo añadiremos fuentes de datos a la infraestructura, pero sin embargo contamos con un parámetro *Deprecate* que si pasa a estar a True eliminara todas las bases de datos de forma controlada

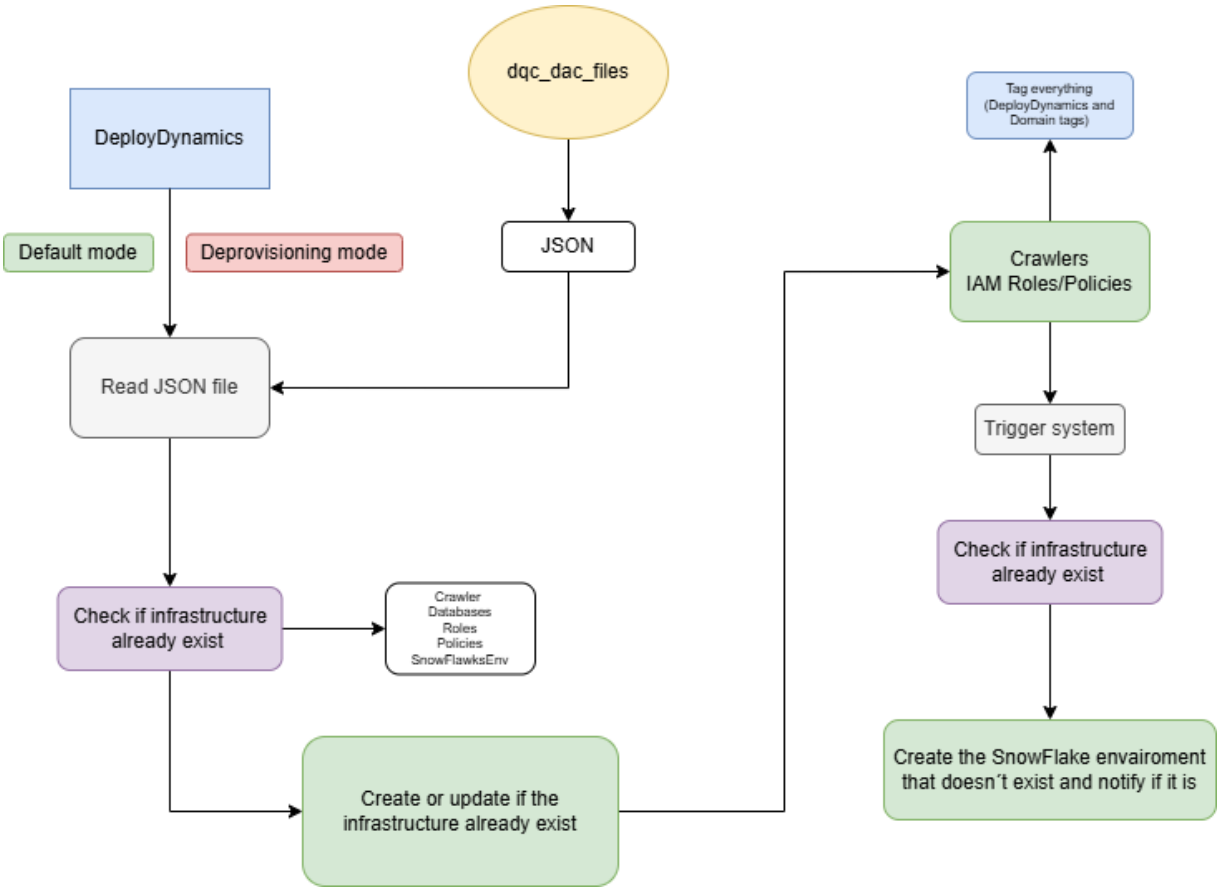
Por otro lado, tenemos la función de *deprovisioning* True que emula el modo agro y que la idea es usarlo cuando solo se tenga una versión en funcionamiento y que se utilice para

mantener la infraestructura limpia, en el caso de que se eliminen se mandara una notificación por correo y *slack* una herramienta de chat y retendrá la operación durante una 1 hora de margen para que se elimine de forma completamente segura.

De este modo el funcionamiento típico debería ser en el caso de tener más de una versión, lanzarlo de tal manera que solo deje la nueva versión y luego le añada las fuentes de datos de otras versiones, de este modo tendremos un proceder sencillo hasta que desarrollemos otras funcionalidades de las que hablaremos más adelante.

Y para añadir una nueva fuente de datos de 0 lo que había que hacer es meter primero una de las fuentes y luego otra en el caso de que para *AWS* haya mas de una.

Ilustración 8 Final diagram



Nota. Diagrama final que se lleva a cabo en la ejecución

2 Solución técnica

2.1 Orquestación

Se ha llevado a cabo un desarrollo del fichero groovy que parametriza y orquesta los posibles tipos de ejecuciones teniendo en cuenta las posibles casuísticas de uso, así como las necesidades técnicas para la ejecución del mismo.

Ilustración 9 Environment Jenkins

Script ?

```
pipeline {
    agent none
    options {
      timestamps ()
      ansiColor ("xterm")
    }

    environment {
        // PYTHON VIRTUAL ENVIRONMENT CONFIGURATION
        PYTHON_VIRTUAL_ENV = "./venv"
        PYTHON_VERSION = "python3.11"
        PYTHON_MAIN_FILE = "main.py"
        PYTHON_INTERPRETER = "${PYTHON_VIRTUAL_ENV}" + "/bin/" + "${PYTHON_VERSION}"

        PATH = "/var/lib/jenkins/.local/bin:/var/lib/jenkins/bin:/usr/bin:/usr/local/bin"

        AWS_REGION = 'eu-west-1'
        TAG_KEY = 'Domain'
        TAG_VALUE = 'PalawanDomain'

        //GIT CONFIGURATION
        GIT_REPOSITORY = "git@gitlab.mgt.forwardkeys.com:dp_tools/managementtechnicalgovernance.git"
        GIT_CREDENTIALS_ID = "continuous.integration"
```

Nota. Código que parametriza las variables clave de la orquestación

Ilustración 10 Starting instance to execute

```
stages{
    stage('GET EC2 INSTANCE ID FOR ETL') {
        agent {
            label "${params.EC2_LABEL}"
        }
        steps {
            sh label: 'Print node name', script: 'echo ${NODE_NAME}'
            script {
                def instanceId = sh(
                    script: 'cat /sys/devices/virtual/dmi/id/board_asset_tag',
                    returnStdout: true
                ).trim()

                def instanceType = sh(
                    script: 'cat /sys/devices/virtual/dmi/id/product_name',
                    returnStdout: true
                ).trim()

                def nodeName = sh(
                    script: 'echo $NODE_NAME',
                    returnStdout: true
                ).trim()

                env.INSTANCE_ID = instanceId
                env.NODE_NAME_ETL = nodeName

                echo "Instance ID: ${instanceId}"
                echo "Instance Type: ${instanceType}"
                echo "Node Name: ${nodeName}"

                def volumeIdOutput = sh(
                    script: "aws ec2 describe-instances --instance-ids ${env.INSTANCE_ID} --query 'Reservations[0].Instances[0].BlockDeviceMappings[
                    returnStdout: true
                ).trim()

                env.VOLUME_ID = volumeIdOutput
                echo "Volume ID: ${env.VOLUME_ID}"

                sh "aws ec2 create-tags --resources ${env.VOLUME_ID} --tags Key=${env.TAG_KEY},Value=${env.TAG_VALUE} --region ${env.AWS_REGION}"
                sh "aws ec2 create-tags --resources ${env.INSTANCE_ID} --tags Key=${env.TAG_KEY},Value=${env.TAG_VALUE} --region ${env.AWS_REGION}"

            }
```

Nota. Código que consigue una máquina de forma dinámica para la ejecución

Ilustración 11 Setup the enviroment

```
stage ('List env variables') {
    agent {
        node "${env.NODE_NAME_ETL}"
    }
    steps {
        script {
            currentBuild.displayName = "${params.TAG_TO_BUILD} - ${params.DATE_FROM}/${params.DATE_TO}"

            def fields = env.getEnvironment()
            fields.each {
                key, value -> println("${key} = ${value}");
            }
        }
    }
}
stage ('Clone repository') {
    agent {
        node "${env.NODE_NAME_ETL}"
    }
    steps {
        checkout([$class: 'GitSCM', branches: [[name: "refs/tags/${TAG_TO_BUILD}"]], doGenerateSubmoduleConfigurations: false, ext
    }
}
stage ('Install dependencies') {
    agent {
        node "${env.NODE_NAME_ETL}"
    }
    steps {
        script {
            sh "virtualenv --python=$PYTHON_VERSION venv \n" +
            "source venv/bin/activate \n" +
            "pip install boto3 \n"+
            "pip install -r requirements.txt"
        }
    }
}
```

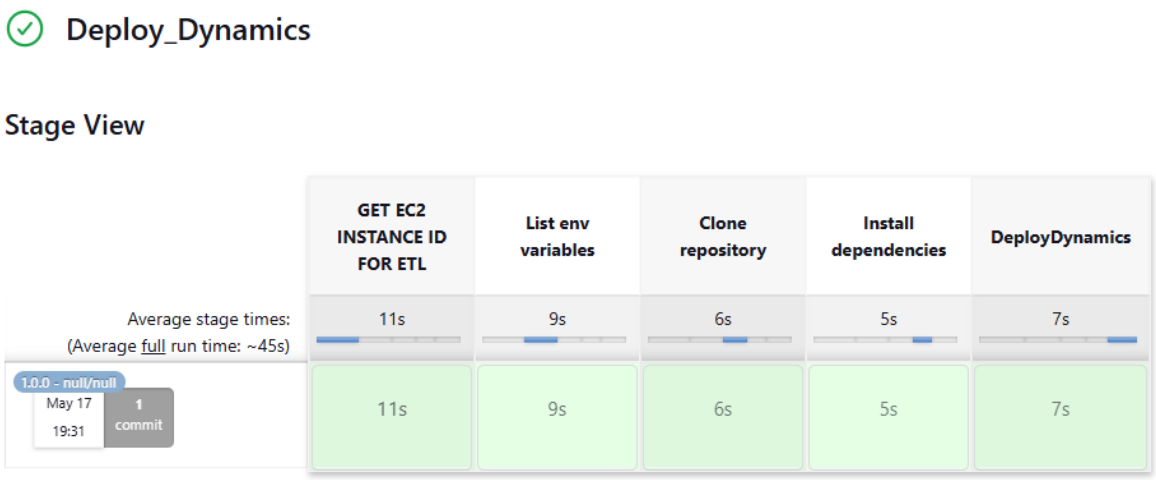
Nota. Código prepara la máquina para la ejecución.

Ilustración 12 Launch execution

```
stage ('DeployDynamics') {
    agent {
        node "${env.NODE_NAME_ETL}"
    }
    steps {
        script{
            withCredentials([
            usernamePassword( credentialsId:'cicd_aws', usernameVariable: 'AWS_KEY', passwordVariable: 'AWS_SECRET'),
            ]){
            sh """
            taskset -c 0-32 ${env.PYTHON_INTERPRETER}  ${env.PYTHON_MAIN_FILE} --json_path ${params.JSON_PATH} --git_tag ${params.DQC_DAC_REPO_TAG}
            """
            }
        }
    }
}
```

Nota. Código prepara la máquina para la ejecución.

Ilustración 13 Succesfully execution



Nota. Ejecución completa del desplegador.

Ilustración 14 Execution log get instance

```
19:31:15  Running on EC2 (FWK_AWS) - PALAWAN DOMAIN M7g.8xlarge AMI 2023 (i-0681441e67bf6f38e) in /var/lib/jenkins/workspace/Deploy_Dynamics
[Pipeline] {
[Pipeline] sh (Print node name)
19:31:16  + echo EC2 '(FWK_AWS)' - PALAWAN DOMAIN M7g.8xlarge AMI 2023 '(i-0681441e67bf6f38e)'
19:31:16  EC2 (FWK_AWS) - PALAWAN DOMAIN M7g.8xlarge AMI 2023 (i-0681441e67bf6f38e)
[Pipeline] script
[Pipeline] {
[Pipeline] sh
19:31:17  + cat /sys/devices/virtual/dmi/id/board_asset_tag
[Pipeline] sh
19:31:18  + cat /sys/devices/virtual/dmi/id/product_name
[Pipeline] sh
19:31:19  + echo EC2 '(FWK_AWS)' - PALAWAN DOMAIN M7g.8xlarge AMI 2023 '(i-0681441e67bf6f38e)'
[Pipeline] echo
19:31:20  Instance ID: i-0681441e67bf6f38e
[Pipeline] echo
19:31:20  Instance Type: m7g.8xlarge
[Pipeline] echo
19:31:20  Node Name: EC2 (FWK_AWS) - PALAWAN DOMAIN M7g.8xlarge AMI 2023 (i-0681441e67bf6f38e)
[Pipeline] sh
19:31:21  + aws ec2 describe-instances --instance-ids i-0681441e67bf6f38e --query 'Reservations[0].Instances[0].BlockDeviceMappings[?DeviceName==`/dev/sdb`].Ebs.VolumeId' --output text --region eu-
west-1
[Pipeline] echo
19:31:22  Volume ID: vol-00e148d3d68126877
[Pipeline] sh
19:31:23  + aws ec2 create-tags --resources vol-00e148d3d68126877 --tags Key=Domain,Value=PalawanDomain --region eu-west-1
[Pipeline] sh
19:31:24  + aws ec2 create-tags --resources i-0681441e67bf6f38e --tags Key=Domain,Value=PalawanDomain --region eu-west-1
```

Nota. Log get instance.

Ilustración 15 Log clone repository

```
19:31:36  Running on EC2 (FWK_AWS) - PALAWAN DOMAIN M7g.8xlarge AMI 2023 (i-0681441e67bf6f38e) in /var/lib/jenkins/workspace/Deploy_Dynamics
[Pipeline] {
[Pipeline] checkout
19:31:37  The recommended git tool is: NONE
19:31:37  using credential continuous.integration
19:31:37  Fetching changes from the remote Git repository
19:31:37   > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/Deploy_Dynamics/.git # timeout=10
19:31:37   > git config remote.origin.url git@gitlab.mgt.forwardkeys.com:dp_tools/managementtechnicalgovernance.git # timeout=10
19:31:37  Fetching upstream changes from git@gitlab.mgt.forwardkeys.com:dp_tools/managementtechnicalgovernance.git
19:31:37   > git --version # timeout=10
19:31:37   > git --version # 'git version 2.40.1'
19:31:37  using GIT_SSH to set credentials
19:31:37  Verifying host key using known hosts file
19:31:37   > git fetch --tags --force --progress -- git@gitlab.mgt.forwardkeys.com:dp_tools/managementtechnicalgovernance.git +refs/heads/*:refs/remotes/origin/* # timeout=10
19:31:38  Checking out Revision bf556f9ca3e05dc88c476bfaba8a1fd054fbc9b7 (refs/tags/1.0.0)
19:31:38   > git rev-parse refs/remotes/origin/refs/tags/1.0.0^{commit} # timeout=10
19:31:38   > git rev-parse refs/tags/1.0.0^{commit} # timeout=10
19:31:38   > git config core.sparsecheckout # timeout=10
19:31:38   > git checkout -f bf556f9ca3e05dc88c476bfaba8a1fd054fbc9b7 # timeout=10
19:31:41  Commit message: "Fist working prototype"
19:31:41   > git rev-list --no-walk c540c388dd1c32fbcf05754173f4f99a786cf20a # timeout=10
```

Nota. Log de la clonación del repositorio.

Ilustración 16 Execution log

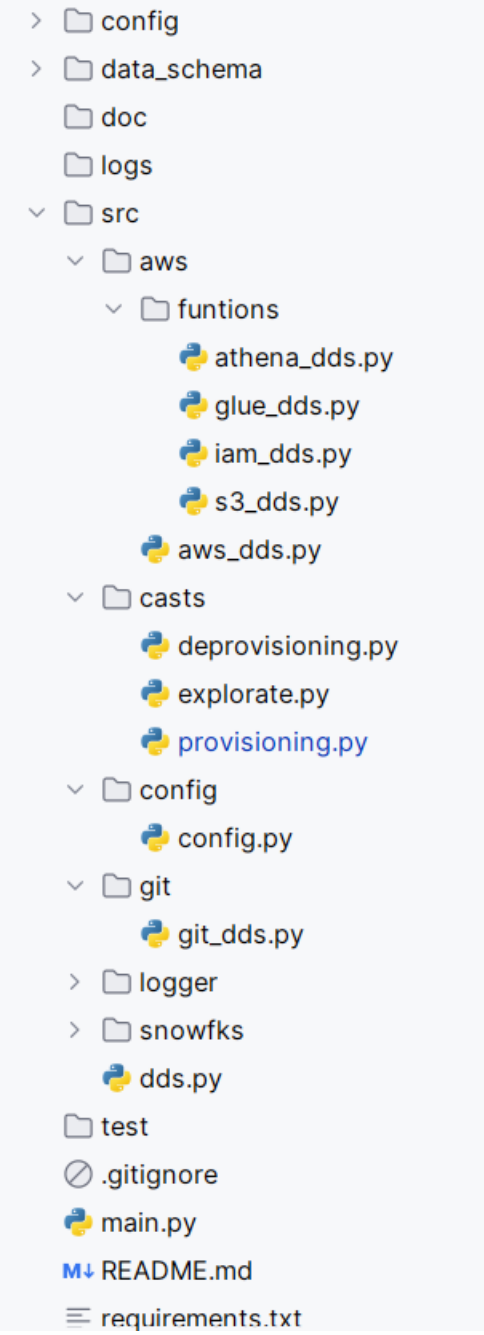
```
19:31:50  Warning: A secret was passed to "sh" using Groovy String interpolation, which is insecure.
19:31:50                 Affected argument(s) used the following variable(s): [AWS_KEY, AWS_SECRET]
19:31:50                 See https://jenkins.io/redirect/groovy-string-interpolation for details.
19:31:51  + taskset -c 0-32 ./venv/bin/python3.11 main.py --provisioning --deploy_aws --json_path test/test.json --git_tag tp_DeployDynamics --aws_key **** --aws_secret ****
19:31:51  [2025-05-17 17:31:51] || INFO || ************************************************************
19:31:51  [2025-05-17 17:31:51] || INFO ||              INITIATING DEPLOYDYNAMICS ETL
19:31:51  [2025-05-17 17:31:51] || INFO ||                    17/05/2025 17:31:51
19:31:51  [2025-05-17 17:31:51] || INFO || ************************************************************
19:31:51  [2025-05-17 17:31:51] || INFO || Starting DEPLOYDYNAMICS ETL
19:31:51  [2025-05-17 17:31:51] || INFO ||
19:31:51
19:31:51
19:31:51
19:31:51
19:31:51
19:31:51
19:31:51
19:31:51
19:31:51  [2025-05-17 17:31:51] || INFO || ETL DEPLOYDYNAMICS successfully finished
19:31:51  [2025-05-17 17:31:51] || INFO || Starting extracting json
19:31:51  [2025-05-17 17:31:51] || INFO || JSON Values extracted successfully
19:31:52  Política arn:aws:iam::399646532980:policy/AWSGlueServiceRole-dataset-dev actualizada exitosamente.
19:31:52  [2025-05-17 17:31:52] || INFO || Roles for dev ready
19:31:53  Política arn:aws:iam::529933460047:policy/AWSGlueServiceRole-dataset-prod actualizada exitosamente.
19:31:53  [2025-05-17 17:31:53] || INFO || Roles for prod ready
19:31:54  Política arn:aws:iam::537032656873:policy/AWSGlueServiceRole-dataset-inno actualizada exitosamente.
19:31:54  [2025-05-17 17:31:53] || INFO || Roles for inno ready
```

Nota. Log de la ejecución.

2.2 Despliegue de la infraestructura explicación técnica

Para este proyecto utilizaremos una estructura de código clara y organizada para maximizar la eficiencia a la hora de trabajar en el y sobre todo de introducir nuevos cambios y funcionalidades.

Ilustración 17 Code structure



Nota. Estructura de desarrollo de codigo.

2.2.1 Recuperar el JSON con los metadatos necesarios

Para este paso necesitaremos clonar el repositorio donde se encuentran los *json* cargar los datos y acceder a los valores de ese JSON, entraremos a la maquina vía SSH para asegurarnos de que se recupera de forma correcta.

Ilustración 18 Extract metadata

```
self.logger.logger.log(logging.INFO, "Starting extracting json")
git_object = GitDDs(self.args, self.config, self.logger)
json = git_object.start()

(name, version, domain_tag, crawler_update, database,
 database_description, extracted_paths) = self.extract_json_data(json)
self.logger.logger.log(logging.INFO, "JSON Values extracted successfully")

prod_prefixes, dev_prefixes, ilab_prefixes = self.filter_prefixes_by_env(extracted_paths, crawler_update)
```

Nota. Se inicializa la clase de git que recupera el JSON.

Ilustración 19 Step to clone and load the json

```
1 usage   Miguel Santiago Pérez *
def start(self):
    self.__git_clone(self.config.git_path, self.config.git_local_path, self.args.git_tag)
    self.logger.logger.log(logging.INFO, "dqc_dac repo cloned successfully")

    json = self.load_json(os.path.join(self.config.git_local_path, "dataset", self.args.json_path))
    self.logger.logger.log(logging.INFO, "JSON found successfully")

    return json
```

Nota. Estructura de la clase.

Ilustración 20 EC2 Machine

```
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sat May 17 16:48:03 2025 from 172.16.1.4
[ec2-user@ip-10-0-0-192 ~]$ cd dqc_dac_repo/
[ec2-user@ip-10-0-0-192 dqc_dac_repo]$ cd dataset/test/
[ec2-user@ip-10-0-0-192 test]$ ll
total 4
-rw-rw-r--. 1 ec2-user ec2-user 2242 May 17 16:43 test.json
[ec2-user@ip-10-0-0-192 test]$ less -1000 test.json
[ec2-user@ip-10-0-0-192 test]$
```

Nota. Comprobamos que en la propia máquina encontramos el JSON.

Ilustración 21 JSON loaded in the EC2 machine

```
{
    "dataset": {
        "name": "dataset",
        "version": "1_0_0",
        "domain_tag": "PalawanDomain",
        "glue": {
          "crawler_update": "true",
          "database": "dataset",
          "database_description": "",
          "dataset_paths": [
            {
              "descripcion": "Dataset Description",
              "prefix": "s3://data-prod/dataset_1",
              "step_type": "c",
              "file_type_information": {
                "type": "parquet",
                "delimitador": "",
                "header": "True"
              },
              "name": "dataset_1",
              "env": {
                "prod": "True",
                "dev": "True",
                "ilab": "True"
              },
              "dependency": {
                "inbound_dependency": [
                  {
                    "name": "dataset"
                  },
                  {
                    "name": "dataset"
                  }
                ],
                "outbound_dependency": [
                  {
                    "name": "dataset"
                  },
                  {
                    "name": "dataset"
                  }
                ]
              }
            },
            {
test.json
```

Nota. JSON de ejemplo cargado a la máquina de EC2.

Ilustración 22 Get paths to deploy

```
def filter_prefixes_by_env(self, extracted_paths, crawler_update):
    if self.args.force_deploy:
        prod_prefixes = [path['prefix'] for path in extracted_paths if path['env'].get('prod') == 'True']
        dev_prefixes = [path['prefix'] for path in extracted_paths if path['env'].get('dev') == 'True']
        ilab_prefixes = [path['prefix'] for path in extracted_paths if path['env'].get('ilab') == 'True']

    else:
        prod_prefixes = [path['prefix'] for path in extracted_paths if path['env'].get('prod') == 'True'
                         and path['step_type'] == "c"
                         and crawler_update == "true"]

        dev_prefixes = [path['prefix'] for path in extracted_paths if path['env'].get('dev') == 'True'
                        and crawler_update == "true"]

        ilab_prefixes = [path['prefix'] for path in extracted_paths if path['env'].get('ilab') == 'True'
                         and path['step_type'] == "c"
                         and crawler_update == "true"]

    return prod_prefixes, dev_prefixes, ilab_prefixes
```

Nota. Se selecciona las rutas a partir del json dependiendo del entorno.

2.2.2 Creación o actualización de roles.

Actualizaremos los permisos de los roles por triplicado para cada entorno.

Ilustración 23 Starting IAM object

```
for env in ["dev", "prod", "inno"]:
    path_list = (
        prod_prefixes if env == "prod" else
        dev_prefixes if env == "dev" else
        ilab_prefixes if env == "inno" else
        []
    )
    iam_object = IamDDs(self.args, self.config, self.logger, env)
    iam_object.start(name, domain_tag, path_list, env)

    self.logger.logger.log(logging.INFO, f"Roles for {env} ready")
```

Nota. Se selecciona las rutas a las que se les tiene que dar permiso en función del entorno

Ilustración 24 Update or create IAM env

```
1 usage  Miguel Santiago Pérez *
def start(self, name, domain, paths, env):
    try:
        policies = self.get_and_update_policy(self.config.role_name.format(dataset = name), paths)

    except ClientError as e:
        print(e.response['Error']['Message'])

        custom_policy = self.format_policy(paths)
        self.create_iam_role_with_policy(self.config.role_name.format(dataset = name), self.config.policie_name.format(dataset
```

Nota. Intenta actualizar los permisos y si no es capaz los crea.

Ilustración 25 Get and update policy code part

```
class IamDDs(AwsDDs):
    def get_and_update_policy(self, role_name, rutas):
        """
        :param role_name:
        :param rutas:
        :return:

        """

        response = self.iam_client.list_attached_role_policies(RoleName=role_name)
        formatted_rutas = [f"arn:aws:s3:::{ruta[5:]}*" for ruta in rutas]

        for policy in response.get('AttachedPolicies', []):
            if policy['PolicyName'] != 'AWSGlueServiceRole':
                policy_details = self.iam_client.get_policy(PolicyArn=policy['PolicyArn'])
                policy_version = self.iam_client.get_policy_version(
                    PolicyArn=policy['PolicyArn'],
                    VersionId=policy_details['Policy']['DefaultVersionId']
                )
                policy_document = policy_version['PolicyVersion']['Document']
                if 'Statement' in policy_document:
                    for statement in policy_document['Statement']:

                        if 'Resource' in statement:
                            existing_resources = statement['Resource']
                            if not isinstance(existing_resources, list):
                                existing_resources = [existing_resources]

                            if self.args.deprovisioning:
                                resources_to_remove = list(set(existing_resources) - set(formatted_rutas))
```

Ilustración 26 Get and update policy code part 2

```
                if resources_to_remove:
                    print(f" Sources to remove {policy['PolicyName']}:")
                    for res in resources_to_remove:
                        print(f" - {res}")

                statement['Resource'] = formatted_rutas
            else:
                statement['Resource'] = list(set(existing_resources + formatted_rutas))

    self.update_policy(policy['PolicyArn'], policy_document)
```

Ilustración 27 Update policy

```
1 usage   Miguel Santiago Pérez
def update_policy(self, policy_arn, policy_document):
    try:
        self.iam_client.create_policy_version(
            PolicyArn=policy_arn,
            PolicyDocument=json.dumps(policy_document),
            SetAsDefault=True
        )
        print(f"Política {policy_arn} actualizada exitosamente.")
    except Exception as e:
        print(f"Error al actualizar la política {policy_arn}: {e}")
```

Nota. Actualiza policy.

Ilustración 28 26 Create policy

```
1 usage   Miguel Santiago Pérez
@staticmethod
def format_policy(paths):
    return {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "s3:GetObject",
                    "s3:PutObject"
                ],
                "Resource": [f"arn:aws:s3:::{path.replace('s3://', '')}*" for path in paths]
            }
        ]
    }
```

Nota. Se crea la policy con el acceso a las rutas pertinentes.

Ilustración 29 Create role and atach policy part 1

```
1 usage   Miguel Santiago Pérez
def create_iam_role_with_policy(self, role_name, policy_name, policy_json, name, domain):
    self.delete_policy_rol(role_name, policy_name)
    trust_policy = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {"Service": "glue.amazonaws.com"},
                "Action": "sts:AssumeRole"
            }
        ]
    }

    try:
        # Rol
        response = self.iam_client.create_role(
            RoleName=role_name,
            AssumeRolePolicyDocument=json.dumps(trust_policy),
            Description="Rol de IAM para AWS Glue con política personalizada",
            Tags = [
                {"Key": "Domain", "Value": f"{domain}"},
                {"Key": "Dataset", "Value": f"{name}"},
                {"Key": "Origin", "Value": "DeployDynamics"}
            ]
        )
        role_arn = response['Role']['Arn']
```

Nota. Se crea un rol con los permisos creados anteriormente.

Ilustración 30 Create role and atach policy part 2

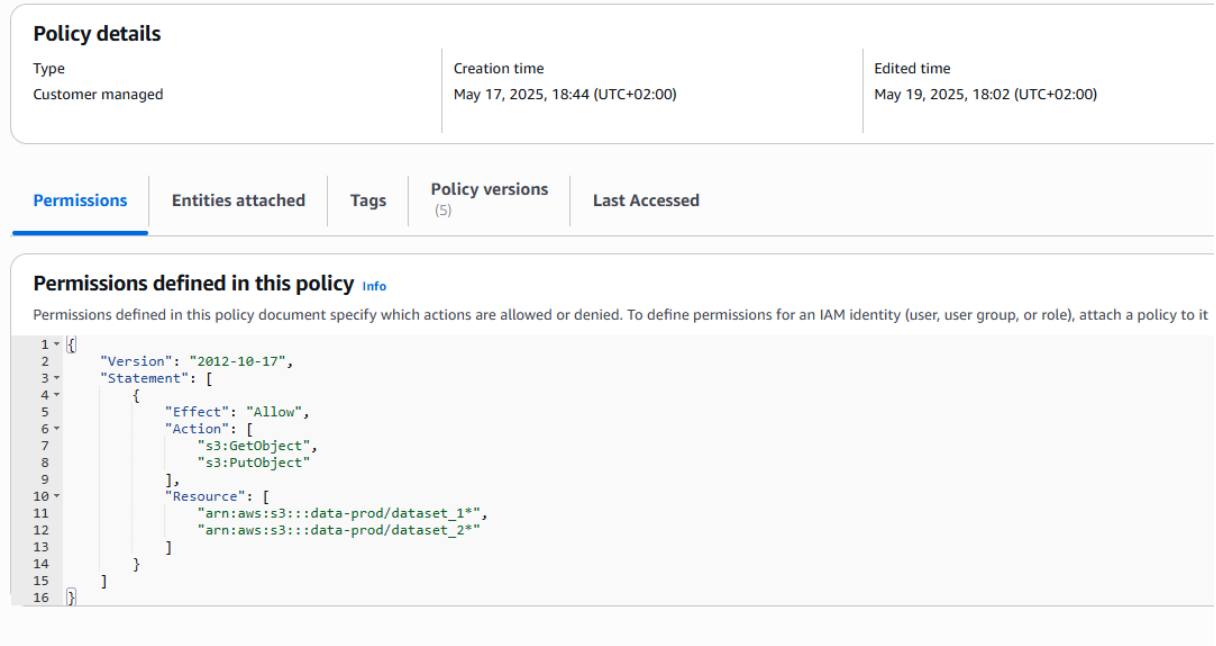
```
self.iam_client.attach_role_policy(
    RoleName=role_name,
    PolicyArn="arn:aws:iam::aws:policy/service-role/AWSGlueServiceRole"
)

# Policy
policy_response = self.iam_client.create_policy(
    PolicyName=policy_name,
    PolicyDocument=json.dumps(policy_json),
    Description="Política personalizada para AWS Glue",
    Tags=[
        {"Key": "Domain", "Value": f"{domain}"},
        {"Key": "Dataset", "Value": f"{name}"},
        {"Key": "Origin", "Value": "DeployDynamics"}
    ]
)
policy_arn = policy_response['Policy']['Arn']

# Atach Policy
self.iam_client.attach_role_policy(
    RoleName=role_name,
    PolicyArn=policy_arn
)
return role_arn
```

Nota. Se anexa los permisos pertinentes al rol.

Ilustración 31 Policy



Nota. Policy creada de forma automática y anexada de forma automática.

2.2.3 Creación o actualización del crawler.

Creamos o actualiamos las fuentes de datos de los crawlers.

Ilustración 32Starting IAM object

```
for env in ["dev", "prod", "inno"]:
    path_list = (
        prod_prefixes if env == "prod" else
        dev_prefixes if env == "dev" else
        ilab_prefixes if env == "inno" else
        []
    )

    self.logger.logger.log(logging.INFO, f"Paths to deploy: {path_list}")

    glue = GlueDDs(self.args, self.config, self.logger, env)
    glue.start(path_list ,domain_tag, name)

    self.logger.logger.log(logging.INFO, f"{env} successfully deployed")
```

Nota. Inicializamos la clase de glue por triplicado con las rutas para desplegar por entorno.

Ilustración 33 Update or create database and crawler

```
def start(self, paths, domain, name):
    self.create_database(self.config.database_name.format(dataset = name))

    self.get_create_crawler(self.config.crawler_name.format(dataset = name),self.config.database_name.format(dataset = name)
                        ,paths, domain, name)


2 usages (1 dynamic)   Miguel Santiago Pérez
def create_database(self, database_name):
    try:
        self.glue_client.delete_database(Name=database_name)

    except:
        print(f"{database_name} Do not exist Exists.")

    self.glue_client.create_database(
        DatabaseInput={
                        "Name": database_name
                       }
    )
```

Nota. La clase de glue empieza chequeando si puede eliminar la base de datos.

Ilustración 34 Create crawler part 1

```
def get_create_crawler(self,crawler_name, database_name, paths, domain, name):
    try:
        crawler = self.extract_crawler(crawler_name)

        if crawler:
            current_targets = crawler['Targets'].get('S3Targets', [])
            current_paths = {target['Path'] for target in
                             current_targets}
            new_paths = set(paths)

            if self.args.deprovisioning:
                paths_to_remove = current_paths - new_paths

                if paths_to_remove:
                    print(f"Paths to delete {crawler_name}:")
                    for path in paths_to_remove:
                        print(f" - {path}")

                updated_targets = [{'Path': ruta, "Exclusions": ["**.crc"]} for ruta in new_paths]
            else:
                updated_targets = [{'Path': ruta, "Exclusions": ["**.crc"]} for ruta in
                                   current_paths | new_paths]

            self.glue_client.update_crawler(
                Name=crawler_name,
                Targets={'S3Targets': updated_targets}
            )
```

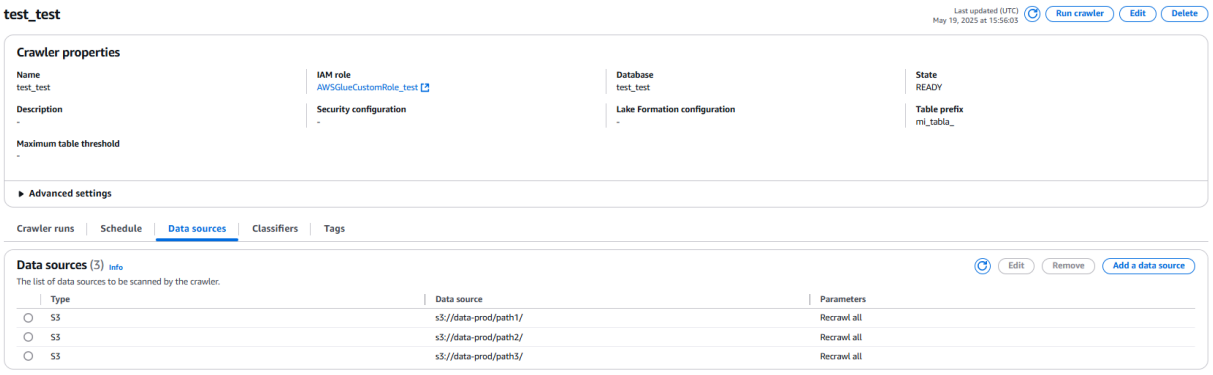
Nota. Se intenta recuperar y actualizar el crawler si existe.

Ilustración 35 Create crawler part 2

```
    except:
        response = self.glue_client.create_crawler(
            Name=crawler_name,
            Role=self.config.role_name.format(dataset = name),
            DatabaseName=database_name,
            Targets={"S3Targets": [{"Path": path, "Exclusions": ["**.crc"]} for path in paths]},
            Tags = {
            "Domain": f"{domain}",
            "Dataset": f"{name}",
            "Origin": "DeployDynamics"
            }
        )
```

Nota. En el caso de que no exista se crea.

Ilustración 36 Crawler



Nota. Aquí podemos observar el crawler creado.

2.2.4 Creación de evento de notificaciones.

Se crean los eventos de notificaciones para que se pueda avisar al snowpipe de que datos nuevos se introducen.

Ilustración 37 Create even notification

```
2 usages (1 dynamic)   Miguel Santiago Pérez *
def create_event_notification(self, bucket_name , sqs_queue_arn, event_name , prefix_filter ,suffix_filter):
    notification_configuration = {
        "QueueConfigurations": [
            {
                "Id": event_name,
                "QueueArn": sqs_queue_arn,
                "Events": ["s3:ObjectCreated:*"],
                "Filter": {
                    "Key": {
                        "FilterRules": [
                            {"Name": "prefix", "Value": prefix_filter},
                            {"Name": "suffix", "Value": suffix_filter},
                        ]
                    }
                }
            }
        ]
    }

    response = self.s3_client.put_bucket_notification_configuration(
        Bucket=bucket_name,
        NotificationConfiguration=notification_configuration
    )
```

Nota. Se crea el evento de triggers.

2.2.5 Despliegue de entorno Snowflake

El entorno de *Snowflake* consta de 2 pasos principales, en primera instancia añadir la ruta de los datos de s3 a la lista de rutas permitidas y mas adelante desplegar la infraestructura como tal que se hace mediante el conector de *Snowflake* mediante queries formateadas de la siguiente manera que se definen en un archivo de configuración y luego se formatean.

Ilustración 38 Update allow storage integration

```
snf = self.get_snowflakes_con(self.config, self.logger)
cursor = snf.conn.cursor()

cursor.execute("DESCRIBE STORAGE INTEGRATION S3_PROD_FWK")
columns = [col[0] for col in cursor.description]
results = cursor.fetchall()
df = pd.DataFrame(results, columns=columns)
df_filtered = df[df['property'] == 'STORAGE_ALLOWED_LOCATIONS']


property_value_str = df_filtered['property_value'].values[0]
paths_list = property_value_str.split(',')

paths_list = list(set(paths_list + paths_to_add))

formatted_paths = ",\n".join(f"'{path}'" for path in paths_list)

query = f"""
ALTER STORAGE INTEGRATION S3_PROD_FWK SET storage_allowed_locations = (
{formatted_paths}
);
"""

cursor.execute(query)
```

Nota. Se añaden las rutas deseadas.

Ilustración 39 Queries needed to deploy Snowflake enviroment

```
query_create_stage= f"""
                    CREATE OR REPLACE STAGE {env}.{database}.S3_STAGE_PROD_{dataset}_C_{version}
                    storage_integration = S3_PROD_FWK
                    url = {path};
                    """

query_create_table = f"""
            create or replace TABLE {env}.{dataset}.C_{version} (
            {schema}
            );
            """


pipe = f"""
        create or replace pipe {env}.{database}.S3_STAGE_PROD_{dataset}_C_{version}
        auto_ingest=true as COPY
        INTO
            dataset
        FROM (
        SELECT
               {schema}
        FROM @{env}.{database}.S3_STAGE_PROD_{dataset}_C_{version} t1) FILE_FORMAT = ( FORMAT_NAME =
        """
```

Nota. Estas queries permiten desarrollar el entorno de Snowflake en su totalidad.

Discusión

1 Limitaciones del trabajo

El proyecto presentó algunas dificultades fruto de que no se quería ceñir a un proyecto estático, sino que quería ser una plataforma integradora transversal que fuese capaz de influir en distintos campos, por ese motivo tenía que ser una solución no acotada, sino que mirara el entorno con un punto de vista muy amplio que se escapa en ocasiones a el conocimiento personal y fue preciso el contactar con diversas partes de la organización para recopilar ideas de cuál sería la proyección y construir la herramienta de tal manera que fuera a ser útil en un futuro próximo.

2 líneas futuras de desarrollo

Como ya mencioné anteriormente existen funcionalidades que se podrán desarrollar a partir de este *MVP* ya que el proyecto se ha desarrollado teniéndolo en cuenta.

2.1 Desarrollo de desplegador all

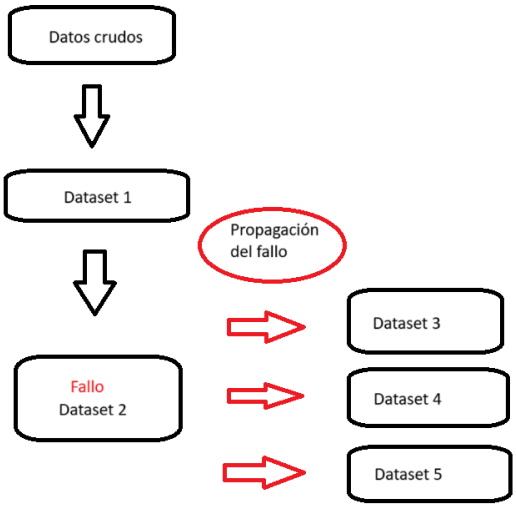
Un modo donde se comprueben a la vez todos los *datasets* de la compañía y notifique lo que esta desalineado, este tendrá una *white list* donde se podrán poner excepciones en el caso en el que se requiera y servirá para mapear todos los cambios en la infraestructura.

Sin embargo, para que eso se lleve a cabo esta herramienta ha de permear y todos los equipos han de empezar a implementarla.

2.2 Herencia de datos

Con los parámetros de los *JSON* dependencia de entrada o de salida, podremos hacer un diagrama de flujo que muestre las dependencias directas o indirectas de un *dataset,* que es tremendamente útil para detectar dependencias propagaciones de errores, así como para tener una mejor gobernanza sobre los datos que se tiene y tener un mapa global como se puede apreciar en la Figura, que sobre todo puede ayudar a generar nuevas ideas y mejor gestión sobre la infraestructura que se tiene.

Ilustración 40 Future idea



Nota. Aquí podemos ver cuál será la siguiente iteración.

Conclusión

La realización de este proyecto ha supuesto una experiencia altamente enriquecedora tanto a nivel técnico como personal. El desarrollo me ha permitido profundizar en tecnologías ampliamente utilizadas en entornos profesionales actuales.

A lo largo del proyecto, he adquirido un conocimiento sólido sobre librerías especializadas como boto3 o el conector de *Snowflake* en profundidad, así como sobre la estructura y lógica necesarias para interactuar de forma programática con servicios en la nube. Además, la integración de múltiples tecnologías y la necesidad de coordinar distintas fases del despliegue han hecho de este trabajo un ejercicio muy transversal, en el que he podido aplicar conceptos de automatización, buenas prácticas de desarrollo y gestión de configuraciones.

Uno de los principales aprendizajes ha sido la importancia de una buena organización y planificación. Diseñar un orquestador capaz de coordinar correctamente todas las funcionalidades, estructurar el código en clases modulares y definir una configuración flexible, han sido tareas que me han permitido mejorar significativamente mis habilidades.

En definitiva, este proyecto me ha permitido no solo aplicar conocimientos adquiridos durante la carrera, sino también ampliar mi perspectiva sobre el trabajo en entornos reales, donde la interoperabilidad entre servicios, la automatización y la claridad en el diseño son claves para el éxito de cualquier proyecto tecnológico.

Referencias

1 Boto3

Amazon Web Services. (s.f.). *Boto3 documentation*. Recuperado el 30 de abril de 2025, de https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

2 Snowflake

Snowflake Inc. (s.f.). *Python connector*. Recuperado el 30 de abril de 2025, de https://docs.snowflake.com/en/developer-guide/python-connector/python-connector
Snowflake Inc. (s.f.). *Python connector examples*. Recuperado el 30 de abril de 2025, de https://docs.snowflake.com/en/user-guide/python-connector-example.html
Snowflake Inc. (s.f.). *Snowpipe guide*. Recuperado el 30 de abril de 2025, de https://docs.snowflake.com/en/user-guide/data-load-snowpipe-intro
Snowflake Inc. (s.f.). *CREATE STAGE*. Recuperado el 30 de abril de 2025, de https://docs.snowflake.com/en/sql-reference/sql/create-stage
Snowflake Inc. (s.f.). *CREATE PIPE*. Recuperado el 30 de abril de 2025, de https://docs.snowflake.com/en/sql-reference/sql/create-pipe

3 AWS

Amazon Web Services. (s.f.). *Creating a service policy for AWS Glue*. Recuperado el 30 de abril de 2025, de https://docs.aws.amazon.com/glue/latest/dg/create-service-policy.html
Amazon Web Services. (s.f.). *Creating an IAM role for AWS Glue*. Recuperado el 30 de abril de 2025, de https://docs.aws.amazon.com/glue/latest/dg/create-an-iam-role.html
Amazon Web Services. (s.f.). *Crawler prerequisites*. Recuperado el 30 de abril de 2025, de https://docs.aws.amazon.com/glue/latest/dg/crawler-prereqs.html
Amazon Web Services. (s.f.). *Describing tables in AWS Glue Data Catalog*. Recuperado el 30 de abril de 2025, de https://docs.aws.amazon.com/glue/latest/dg/tables-described.html