ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER EN BIG DATA: TECNOLOGÍA Y ANALÍTICA AVANZADA

DESPLIEGUE Y AUTOMATIZACIÓN DE SOLUCIONES BIG DATA EN CLOUD Y ON PREMISE

Autor: Andrés Jiménez Arocha

Director: Carlos Cuezva Tordable

Alberto Antón Pérez

Madrid

Julio 2021

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

DESPLIEGUE Y AUTOMATIZACIÓN DE SOLUCIONES BIG DATA EN CLOUD Y ON PREMISE

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2020/21 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

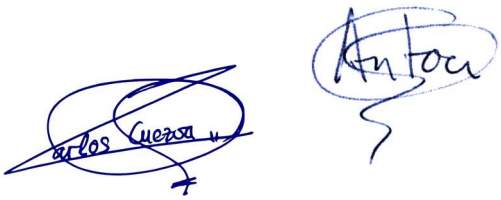
El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Firmado por JIMENEZ AROCHA ANDRES - 43382096W el día 24/06/2021 con un certificado emitido por AC FNMT Usuarios

Fdo.: Andrés Jiménez Arocha Fecha: 22/06 /21

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Carlos Cuezva Tordable

Alberto Antón Pérez

Fecha: 22/06/21

Vº Bº del Coordinador de Proyectos

Fdo.: Carlos Morrás Ruiz-Falcó Fecha: 22/06/21

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Andrés Jiménez Arocha

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: **DESPLIEGUE Y AUTOMATIZACIÓN DE SOLUCIONES BIG DATA EN CLOUD Y ON PREMISE,**

que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas

en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

- El autor se compromete a:
  a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
  b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
  c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
  d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 22 de junio de 2021

ACEPTA

Fdo…… Andres Jimenez

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

MÁSTER EN BIG DATA: TECNOLOGÍA Y ANALÍTICA AVANZADA

DESPLIEGUE Y AUTOMATIZACIÓN DE SOLUCIONES BIG DATA EN CLOUD Y ON PREMISE

Autor: Andrés Jiménez Arocha

Director: Carlos Cuezva Tordable

Alberto Antón Pérez

Madrid

Julio 2021

“Doutez toujours de vous-mêmes, jusqu'à ce que les données ne laissent aucun doute”

“Duda siempre de ti mismo, hasta que los datos no dejen lugar a dudas”

Louis Pasteur (1822-1895)

Agradecimientos

A mi familia, acompañándome y apoyándome en todas mis decisiones, buenas y no tan buenas.
A mi otra familia, la que afortunadamente he formado estos meses en el máster, una Catástrofe. Ellos me entenderán.
Y como no, a mis compañeros de Telefónica, Carlos, Alberto, Jose, Guille e Ilde, que me ha acogido y ayudado desinteresadamente durante la realización de este proyecto.
A todos, gracias.

DESPLIEGUE Y AUTOMATIZACIÓN DE SOLUCIONES BIG DATA EN CLOUD Y ON PREMISE

Autor: Jiménez Arocha, Andrés.

Director: Cuezva Tordable, Carlos.

Antón Pérez, Alberto

Entidad Colaboradora: Telefónica S.A.

RESUMEN DEL PROYECTO

El proyecto consistirá en el despliegue y validación del correcto funcionamiento de aplicaciones big data, tanto en arquitecturas cloud como on premise. El producto en cuestión a desplegar, ayudará a conseguir el objetivo principal de la 4 Plataforma (cloud interno), es decir, la homogenización de los datos utilizados por todas las aplicaciones internas para conseguir la unificación y estandarización de los procesos que engloban a todas las unidades de negocio, así como de la automatización de las validaciones de los datos y entidades usadas en estos productos internos, consiguiendo así también la estandarización de los datos a nivel global. Este producto encargado de la validación de los datos será desplegado mediante tecnologías Ansible y ejecutado sobre un clúster big data kerberizado y creado sobre el proveedor Google Cloud mediante tecnologías Terraform.

Palabras clave: Cloud, Clúster Spark, Estandarización, Datos, Kerberos, Terraform, Ansible

DESPLIEGUE Y AUTOMATIZACIÓN DE SOLUCIONES BIG DATA EN CLOUD Y ON PREMISE

Author: Jiménez Arocha, Andrés.
Supervisor: Cuezva Tordable, Carlos.
Antón Pérez, Alberto.
Collaborating Entity: Telefónica S.A.

ABSTRACT

The project will consist of the deployment and validation of the correct operation of big data applications, both in cloud and on premise architectures. The product in question to be deployed will help to achieve the main objective of the 4 Platform (internal cloud), that is, the homogenization of the data used by all internal applications to achieve the unification and standardization of the processes that encompass all business units, as well as the automation of the validations of the data and entities used in these internal products, thus also achieving the standardization of the data at a global level. This product in charge of data validation will be deployed using Ansible technologies and executed on a kerberized big data cluster created on the Google Cloud provider using Terraform technologies.
Keywords: Cloud, Cluster Spark, standardization, Datos, Kerberos, Terraform, Ansible

Índice de la memoria

Índice de imágenes

Índice de tablas

Capítulo 1. INTRODUCCIÓN

En pleno siglo XXI, el volumen de datos ha aumentado de manera exponencial y el costo de su almacenaje se ha reducido considerablemente. Gracias al uso de los sistemas de información y de las nuevas tecnologías los científicos e ingenieros de datos están pudiendo hacer uso de todos estos datos almacenados y pudiendo procesar los mismos y obtener información valiosa. Esta información es válida tanto para el mundo empresarial, como para el medico o para cualquier otro sector en el que haya datos e información de la que se pueda disponer.

Es, por tanto, que este trabajo busca, entre otras cosas, facilitar la infraestructura big data, el despliegue y la posterior ejecución de un software de validación de datos que facilite a la compañía propietaria del mismo una información de gran valía para le gestión de sus productos.

1.1 OBJETIVOS DEL PROYECTO

El principal objetivo del proyecto es lograr el despliegue de un clúster big data capaz de soportar el despliegue y la ejecución de un producto encargado de la validación de los datos y entidades utilizadas por las distintas OB (Operational Business) en Telefónica S.A. Este clúster debe ser totalmente compatible con las tecnologías necesarias para la ejecución de este producto big data y además garantizar la seguridad en el acceso al mismo y a sus datos. Para esto es necesario implementar distintas reglas de seguridad y tecnologías de securización de entornos Hadoop.

Todos estos despliegues se harán haciendo uso de tecnologías de despliegue de infraestructura como código (Ansible y Terraform) garantizando así la automatización de estos procesos.

El objetivo del uso de estas tecnologías es la rapidez y la facilidad con la que permiten, una vez realizados todos los script y configuraciones necesarios, desplegar, modificar y eliminar las infraestructuras y configuraciones realizadas. Esto es de gran valor para un

equipo de DevOps pues el ahorro en tiempo es muy importante y permite a los equipos de desarrollo de software poder probar productos de una manera rápida y automatizada.
De este modo, como objetivo final se busca la homogenización en la 4 Plataforma (cloud interno) de todas las aplicaciones internas para conseguir la unificación y normalización de los procesos que engloban a todas las unidades de negocio, así como de la automatización de las validaciones de los datos y entidades usadas en estos productos internos, consiguiendo así también la normalización de los datos a nivel global.

1.2 MOTIVACIÓN DEL PROYECTO

La motivación de este proyecto, como ya se ha hecho saber en la introducción de este, es el despliegue, tanto en cloud como on premise, de aplicaciones big data de uso interno empresarial que permiten a los distintos departamentos realizar distintas operaciones para obtener los datos y KPI's necesarios para la realización de las tareas que les conciernen.
Estos productos big data han de ser desplegados tanto en cloud como on premise debido a los distintos requerimientos de los países, llamados Operational Business y de ahora en adelante OB, que hacen uso de estos y que por sus distintas particularidades es necesario que en muchas ocasiones tengan el producto no sólo en cloud sino también en sus propios clústeres on premise.
Es debido a estas particularidades que uno de los objetivos de este proyecto sea también el desarrollo y posterior despliegue de un producto interno cuya función es mejorar la gobernabilidad de los datos usados por todas las OB y la homogenización y normalización en un mismo lugar, cloud, de todos estos.
Por ello este proyecto se va a centrar en el despliegue de una plataforma cloud donde se despliegue y ejecute un producto big data encargado de esta validación de datos.
Este clúster big data debe ser totalmente compatible con el producto de validación y gobernabilidad del dato que se ha de desplegar dentro, y también garantizar la seguridad de los datos que este clúster almacena, ya que son de vital importancia para la empresa, debido a que son estos los usados por los demás productos empresariales de los cuáles se obtiene el beneficio. Un fallo en el clúster podría obligar a que se dejarán de validar ciertas entidades de datos (URM o Unified Resource Model) estrictamente necesarias para el

correcto uso de los demás productos, como recomendadores de terminales, etc. por lo que este fallo conduciría en una pérdida de ingresos considerable.

De este modo, como objetivo final se busca la homogenización en la 4 Plataforma (cloud interno) de todas las aplicaciones internas para conseguir la unificación y normalización de los procesos que engloban a todas las unidades de negocio, así como de la automatización de las validaciones de los datos y entidades usadas en estos productos internos, consiguiendo así también la normalización de los datos a nivel global.

1.3 DESCRIPCIÓN DE LAS TECNOLOGÍAS

En cuanto a los distintos recursos empelados en el proyecto cabe destacar que hay dos grupos, aquellos donde se van a alojar tanto los productos big data como los datos, y aquellos que son usados para llegar a este fin.

En los recursos del primer grupo cabe destacar el proveedor de cloud usado en este proyecto, Google Cloud, donde se desplegarán tanto el clúster big data con aquellas tecnologías necesarias para la ejecución del producto, como el producto big data en sí que será ejecutado para la validación de los datos.

En el segundo grupo englobaríamos las herramientas como Terraform o Ansible, siendo estos últimos softwares de infraestructura como código donde se define la estructura de los clústeres tanto en cloud como on premise necesarios para alojar los productos big data así como el despliegue de estos productos en estas infraestructuras.

Por otro lado, cabe destacar ciertas tecnologías que han sido desplegadas en el clúster para el correcto funcionamiento del mismo, como son Spark, Scala, HDFS, Yarn y Kerberos.

Tanto Spark (en su versión 2.4.7), como Scala (en su versión 2.12) son estrictamente necesarios para la ejecución de este producto ya que ha sido desarrollado en Scala.

Además, necesita de la gestión de Yarn sobre los executors de spark para la correcta ejecución distribuida del producto. Dado que este producto necesita acceder a determinados datos y que se ejecuta de manera distribuida, estos datos son alojados en HDFS.

Por último, se ha securizado el clúster, tanto a nivel de firewall (protocolos, reglas y puertos) como a nivel del entorno Hadoop mediante la kerberización del clúster para la autenticación de los usuarios a nivel de servicios.

Capítulo 2. REVIEW TECNOLÓGICO

El objetivo de este apartado es revisar el abanico tecnológico que se puede encontrar actualmente a la hora del desarrollo de un proyecto de estas características y que podrían haber sido usado durante el desarrollo de este proyecto, en lugar de las tecnologías usadas.

2.1 CLOUD COMPUTING

Cloud computing o computación en la nube es una computación basada y alojada en la web que permite a las distintas instituciones, empresas y personas individuales hacer uso de recursos informáticos como pueden ser máquinas virtuales, bases de datos, tecnologías de procesamiento y análisis, almacenamiento, etc. Existen distintos tipos de suscripción, pero uno de los más interesantes es el modelo de pago por uso, conocido como pay-as-you-go. Esta cobra únicamente por lo recursos que se usan, a diferencia de los modelos tradicionales de suscripciones en el mundo informático, si no utiliza ningún recurso, no paga.

Un proveedor de servicios en la nube es una empresa que proporciona todos los servicios anteriores en sus centros de datos, donde están disponibles miles de computadoras, servidores, bases de datos y otros recursos. Un sistema operativo en la nube se diseña y gestiona estos servicios con la ayuda de otros productos.

La computación en la nube es la culminación de numerosos intentos de computación a gran escala con acceso continuo a recursos virtualmente ilimitados (Ahmed, 2021).

Algunas de las ventajas más importantes de la computación en la nube son:

1. La reducción de costos para las empresas ya que todo el hardware, servidores de bases de datos, servidores web, software, productos y servicios están en la nube, y se agregan a la cuenta de servicio a medida.
2. La computación en la nube ofrece 24 horas al día, 7 días a la semana y 365 días al año de servicio. Todos los servidores son administrados por el proveedor sin necesidad de tener a un empleado administrándolos.

3. La computación en la nube es escalable y confiable. No hay límite de recursos ni de usuarios de la nube. Los recursos son escalables dependiendo del uso que se les dé y de sus necesidades en todo momento
4. La computación en la nube proporciona actualizaciones automáticas de todos los servicios y tecnologías que estemos usando.
5. Los proveedores de servicios en la nube tienen centros distribuidos en varios lugares del mundo lo que garantiza una mayor redundancia y velocidad.

La computación en la nube no solo ofrece Infraestructura como servicio (IAAS), sino que también ofrece Plataforma como servicio (PAAS) y Software como servicio (SAAS). Computación en la nube = Software como servicio + Plataforma como servicio + Infraestructura como servicio.

2.1.1 MODELOS DE SERVICIO CLOUD

Los tres modelos de servicio cloud destacables son Software as a Service (SaaS), Platform as a Service (PaaS) y Infraestructure as a Service (IaaS). De este modo es posible alojar un producto software en el cloud, alojar una plataforma, o alojar tu infraestructura entera.

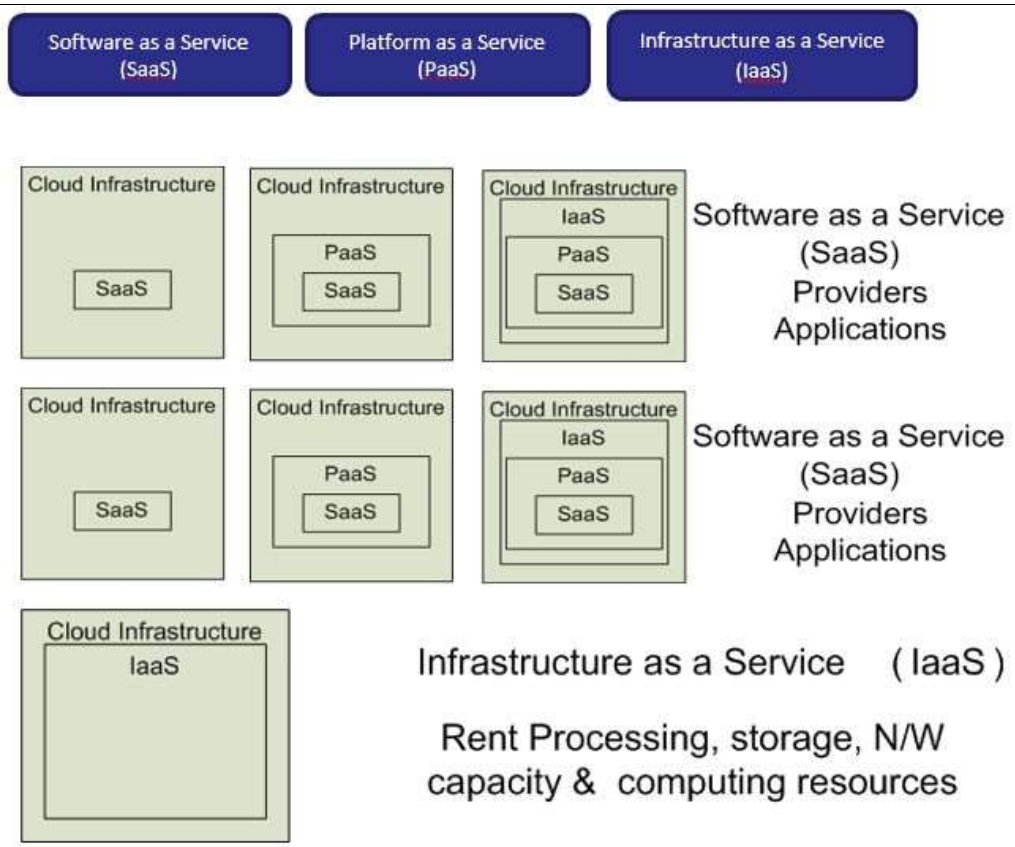


Imagen 1. Cloud Services Models. Fuente:www.c-sharpcorner.com

2.1.2 PRINCIPALES PROVEEDORES

Según el cuadrante de Gartner, en la imagen 2, AWS, Microsoft Azure y Google Cloud, lideran como principales proveedores de *cloud computing.*

Imagen 2. Cuadrante Gartner. Fuente: Gartnert Inc

Según un informe de Canalys que se muestra en el siguiente gráfico, en el cuarto trimestre de 2020, la nube de AWS creció un 28% y las nubes de Azure, Google y Alibaba crecieron un 50%, 58% y 54% respectivamente. Según este informe, AWS tiene el 31% de la cuota de mercado total de la nube, seguido de Azure, Google y Alibaba, que tienen el 20%, el 7% y el 6% respectivamente.

A modo de breve resumen, se explicarán algunas de las características principales de estos tres proveedores de cloud.

2.1.2.1 Microsoft Azure

Microsoft Azure, antes conocida como Windows Azure, es una plataforma e infraestructura de computación en la nube, creada por Microsoft, que permite la

construcción, el despliegue y la gestión de servicios y aplicaciones a través de una red global de centros de datos gestionados por Microsoft. Ofrece una gran variedad de servicios y soporta muchos lenguajes de programación, herramientas y marcos de trabajo, incluyendo tanto software y sistemas específicos de Microsoft como de terceros. Azure se lanzó el 1 de febrero de 2010 (Bigelow, 2020).

Para su uso se ofrece la API de conexión con Azure, a través de la cual se pueden realizar todas las gestiones necesarias.

Microsoft Azure es una de las nubes de más rápido crecimiento entre todas ellas. Azure se lanzó años después del lanzamiento de AWS y Google Cloud, pero sigue llamando a la puerta para convertirse en el principal proveedor de servicios en la nube. Microsoft Azure ha ganado recientemente un contrato con el gobierno estadounidense por valor de 10.000 millones de dólares (Chand, 2021).

Se espera que los ingresos de Azure de Microsoft crezcan entre 33.000 y 35.000 millones de dólares. Esto convierte a Azure en uno de los servicios en la nube más rentables del mundo.

Azure ofrece cientos de servicios dentro de varias categorías, incluyendo AI + Machine Learning, Analytics, Blockchain, Compute, Containers, Databases, Developer Tools, DevOps, Identity, Integration, Internet of Things, Management, Media, Microsoft Azure Stack, Migration, Mixed Reality, Mobile, Networking, Security, Storage, Web, y Windows Virtual Desktop.

Si hablamos de almacenamiento, procesamiento y real time podríamos hablar de tres partes muy simples y a grandes rasgos herramientas de Azure:

- HDInsight: Clúster Hadoop
- Blob Storage/Datalake: Almacenamiento
- Event Hubs: Servicio de ingesta de datos en real time

Microsoft Azure, es el proveedor oficial de Telefónica para computación en la nube, por lo que fue mi primera elección de cara al despliegue del clúster, pero dado que la versión de prueba ofrecida por Azure no permite el despliegue de infraestructuras avanzadas como un HDInsight, se ha tenido que optar por otra opción, concretamente Google Cloud.

2.1.2.2 Amazon Web Services (AWS)

Amazon Web Services (AWS) es una empresa de Amazon que se lanzó en el año 2002. AWS es el proveedor de servicios en la nube más popular y adoptada del mundo, ofreciendo más de 165 servicios en todo el mundo (Chand, 2021).
Algunos de ellos son Virtual Private Cloud, EC2, AWS Data Transfer, Simple Storage Service, DynamoDB, Elastic Compute Cloud, AWS Key Management Service, AmazonCloudWatch, Simple Notification Service, Relational Database Service, Route 53, Simple Queue Service, CloudTrail y Simple Email Service.
AWS tiene la infraestructura en la nube más amplia del mundo. Ningún otro proveedor de servicios en la nube ofrece tantas regiones con varias zonas de disponibilidad conectadas por redes de baja latencia, alto rendimiento y alta redundancia. AWS cuenta con 80 zonas de disponibilidad repartidas en 25 regiones geográficas de todo el mundo y se considera el servicio más seguro pues fue creado para cumplir con los requisitos de seguridad del ejército, los bancos internacionales y otras organizaciones que deben cumplir requisitos de confidencialidad estrictos. (Services Amazon Web, n.d.)

Si hablamos de almacenamiento, procesamiento y real time podríamos hablar de tres partes muy simples y a grandes rasgos herramientas de AWS:

- EMR: Clúster Hadoop
- S3: Almacenamiento
- Kinesis: Servicio de ingesta de datos en real time

2.1.2.3 Google Cloud

Google Cloud ha sido el proveedor elegido finalmente para la creación del cluster big data, ya que en su *free trial* permite la creación y uso de cualquiera de los servicios que proporciona, ofreciendo un crédito de 300$.
La plataforma en la nube de Google es GCP. Al igual que AWS y Azure, Google Cloud también ofrece servicios muy similares como computación, almacenamiento, identidad, seguridad, base de datos, IA y aprendizaje automático, virtualización, DevOps, etc.
Los servicios en la nube de Google están disponibles en 20 regiones, 61 zonas y más de 200 países y su beneficio estimado anual son 8 billones de dólares.

	AWS	Azure	Google Cloud
Company	AWS Inc.	Microsoft	Google
Launch year	2006	2010	2008
Geographical Regions	25	54	21
Availability Zones	78	140 (countries)	61
Key offerings	Compute, storage, database, analytics, networking, machine learning, and AI, mobile, developer tools, IoT, security, enterprise applications, blockchain.	Compute, storage, mobile, data management, messaging, media services, CDN, machine learning and AI, developer tools, security, blockchain, functions, IoT.	Compute, storage, databases, networking, big data, cloud AI, management tools, Identity and security, IoT, API platform
Compliance Certificates	46	90	
Annual Revenue	$33 billion	$35 billion	$8 billion

Tabla 1. Comparativa providers. Fuente:: www.c-sharpcorner.com

2.2 HERRAMIENTAS DE DESPLIEGUE

Las herramientas de despliegue son lo que se conoce como Infrastructure as Code (IaC). La IaC se define, según Red Hat (2021), como la administración y el aprovisionamiento de infraestructura a través del código en lugar de a través de procesos manuales.De este modo se logra la creación de ficheros de configuración que contienen la información y especificaciones de la creación y configuración de infraestructuras, además de permitir la modularización de distintas componentes que permitirán la combinación en diferentes formas durante la automatización. De igual modo, el control de versiones es parte indispensable de la IaC y sus ficheros de configuración deben estar bajo un control de fuente (Red Hat, 2021).

Hay dos formas de abordar IaC: declarativa o imperativa. Cuando se habla de enfoque declarativo define el estado deseado del sistema, incluidos los recursos que necesita y las propiedades que deberían tener

Un enfoque declarativo también mantiene una lista del estado actual de los objetos de su sistema, lo que hace que el desmantelamiento de la infraestructura sea más sencillo de

administrar y que en todo momento haya una sincronización entre las infraestructuras que se tengan desplegadas, evitando así duplicidades o errores.

Por otro lado, en el enfoque imperativo se definen los comandos específicos necesarios para lograr la configuración deseada, y esos comandos deben ejecutarse en el orden correcto.

Todo esto permite con IaC a los desarrolladores no tener que aprovisionar y administrar manualmente los servidores, sistemas operativos, almacenamiento y otros componentes de infraestructura cada vez que desarrollan o implementan una aplicación.

La principal ventaja respecto de la infraestructura como código frente a métodos anteriormente utilizados radica en que está dirigida a un entorno virtualizado, pudiendo a través del código crear, modificar o eliminar recursos de una manera rápida. Por el contrario, en el entorno clásico no virtualizado, todos los recursos están siempre conectados con el hardware físico, lo que resulta en una infraestructura poco flexible y que conlleva muchas horas de trabajo manual a la hora de realizar configuraciones (IONOS, n.d.).

Con la virtualización del hardware del servidor, del almacenamiento y de las estructuras de red, esta situación se torna. Con esta nueva tecnología los proveedores pueden ofrecer servicios a medida, por lo tanto, flexibles y sin necesidad de inversión en hardware.

Algunos de los principales beneficios del uso de estas tecnologías son:

- Reducción de costes
- Aumento de la velocidad de despliegue
- Reducción de errores
- Mejora de la consistencia

En este proyecto han sido utilizadas dos herramientas de despliegue automatizado, Terraform y Ansible, pero no son las únicas existentes. Existen otras como Chef, Puppet o Saltstack de las que hablaremos a continuación.

2.2.1 TERRAFORM

Terraform es una herramienta de despliegue de infraestructura como código desarrollada por la empresa de software HasiCorp. Permite la definición de los recursos y la infraestructura que se desea desplegar en ficheros declarativos. Terraform usa una arquitectura sin agentes. Con la arquitectura basada en agentes, los nodos deben instalar localmente un proceso de comunicaciones con la máquina de control mientras que con la arquitectura sin agentes los nodos no necesitan instalar ni ejecutar en segundo plano ningún proceso que se comunique con la máquina de control. Evitando así la sobrecarga de la red y previniendo el uso de estrategias de control más agresivas por parte del servidor. Esto es llamado Push-based deployment model.
Permite la gestión y creación en múltiples plataformas, algunas de las líderes como AWS, Azure y GCP de una manera legible por humanos y rápida y la creación de módulos independientes que podrán ser lanzados conjunta o de manera individual. La definición de los recursos vendrá limitada por el proveedor que se esté usando, de este modo podremos acceder a todos los recursos de GCP si es este nuestro proveedor. Estos recursos son completados mediante un fichero de definición de variables, o a mano. Lo más recomendable por modularidad es la creación de un fichero de variables donde se definan gran parte de las variables que van a ser utilizadas en los recursos.

En la siguiente imagen vemos un ejemplo de definición de un recurso en GCP, llamado "Google_compute_network" y que concretamente se encarga de la creación de una red en el clúster. Vemos como se hace una llamada a una variable llamada "dataproc_network", la cual a sido definida en un fichero de variables y que consta de 3 atributos a los que se les está haciendo referencia: name, mtu y routing mode

```
resource "google_compute_network" "network_cluster" {
  name                    = var.dataproc_network["name"]
  mtu                     = var.dataproc_network["mtu"]
  auto_create_subnetworks = false
  routing_mode            = var.dataproc_network["routing_mode"]
}
```

Imagen 3. Google network resource

Vemos en la imagen 4 como esta variable ha sido definida en el fichero de variables donde se han establecidos unos valores para cada una de ellas.

```
variable "dataproc_network" {
  description = "Dataproc Netoworks Properties"
  type        =  object(
    {
      name                          = string
      mtu                           = number
      routing_mode                  = string
    }
  )
  default     = {
    name                          = "network-icai"
    mtu                           = 1460
    routing_mode                  = "REGIONAL"
  }
}
```

Imagen 4. Definición network

2.2.2 ANSIBLE

Ansible (Ansible, 2018),es un lenguaje de automatización de código abierto muy potente, además de permitir también el despliegue y orquestación de productos e infraestructuras, algo que lo hace muy potente. Mientras que Ansible proporciona más recursos para sustituir muchas de las capacidades básicas de otras soluciones de automatización, también busca resolver otros grandes retos de IT no resueltos.
Uno de estos retos es permitir la integración y el despliegue continuos (CI/CD) con cero tiempo de inactividad. Este objetivo ha requerido a menudo de codificaciones muy extensas y personalizadas, trabajando además con muchos y diversos paquetes de software. El objetivo final de Ansible es proporcionar todos estos recursos y capacidades en una sola herramienta capaz de orquestar todos estos escenarios.

Combina instalación multinivel permitiendo desplegar servicios por lotes, ejecuciones de tareas ad hoc y administración de configuraciones. Gestiona los nodos a través de conexión

SSH y no requiere ningún software remoto adicional (excepto Python 2.4 o posterior) para instalarlo y nativamente utiliza YAML escribir las configuraciones.
Ansible, al igual que Terraform, es una herramienta de gestión si agentes por lo que no necesita la instalación en los nodos de ningún proceso de comunicación.

Los *Playbooks,* imagen 5*,* son el fichero básico de definición de la configuración que se quiere desplegar en destino. El formato del *Playbook* es YAML y en cada uno definen un conjunto de tareas que serán asignadas a determinados hosts.
Básicamente una tarea no es más que una llamada a un módulo de Ansible.
Al componer un *Playbook* con múltiples "jugadas", es posible orquestar despliegues de múltiples máquinas, ejecutando ciertos pasos en todas las máquinas del grupo de servidores web, otros pasos en el grupo de servidores de base de datos, y luego más comandos en los servidores web, etc (Wikipedia, n.d.) .

Vemos como en esta imagen se ve como hay dos bloques definidos (Downloads Package y Deploy Package) y como nos referimos a en qué host se realizarán las tareas (localhost o dataproc), los cuales son definidos en un fichero de hosts.

En la parte de desarrollo se explicará de manera más avanzada el funcionamiento del *Playbook* y demás ficheros

```
###

- name: "Download package"
  hosts: "localhost"  #llama a host
  connection: local  #porque vamos a hacer este bloque en local
  tags: always
  gather_facts: false  #no coge variables de entorno
  tasks:  #los bloques de tareas que vamos a hacer
  - name: BLOCK - Download and prepare package
    block:
    - name: Using local package if exists
      copy:
        src: "{{ lookup('pipe', 'ls -1 dist/{{ project_name }}-{{ version }}.tar.gz') }}"
        dest: "{{ temporal_local_package_path }}"
    rescue:
    - debug: msg='Using local file has failed. Try to download package from arfifactory.'
    - name: Download {{ project_name }}-{{ version }}.tar.gz
      get_url:
        url: http://artifactory.hi.inet/artifactory/yum-analytics/{{ project_name }}/dist/x86_64/{{ p
        dest: "{{ temporal_local_package_path }}"
    tags: download_package

## Esta parte de arriba es la que ejecutamos en local y ahora pasamos ya a la maquina
#donde desplegamos

- name: "Deploy Package"
  hosts: "dataproc"
  gather_facts: true
  tasks:
  - name: BLOCK - Check delegation
    block:
    - name: BLOCK - Install Package
      tags: install_package
      block:
      - name: Create working directory - OS
        file:
          state: directory
          path: "{{ working_dir }}"

      - name: Unarchive package
        unarchive:
          src: "{{ temporal_local_package_path }}"
          dest: "{{ working_dir }}"
          extra_opts: [--strip-components=1] #en la carpeta destino no crees la carpeta comprimida
```

Imagen 5. Playbook ejemplo

2.2.3 CHEF

Chef es una herramienta de gestión de la configuración escrita en Ruby y Erlang. Utiliza un lenguaje específico de dominio (DSL) en Ruby puro para escribir "recetas" de configuración del sistema. Chef se utiliza para agilizar la tarea de configurar y mantener los servidores de una empresa, y puede integrarse con plataformas basadas en la nube como Amazon EC2, Google Cloud Platform, Oracle Cloud, OpenStack, IBM Cloud, Microsoft Azure y Rackspace para aprovisionar y configurar automáticamente nuevas máquinas.

El usuario escribe estas "recetas" que describen la configuración que se requiere instalar en el sistema. Estas recetas, las cuales al igual que en Terraform pueden modularizarse y ordenarse, describen una serie de recursos que deben llegar a un estado determinado: paquetes que deben instalarse, servicios que deben ejecutarse o archivos que deben escribirse. Se ha de asegurar el correcto orden de instalación/configuración de los servicios ya que la lectura del fichero de configuración es secuencial. Chef se asegura de que cada recurso esté correctamente configurado y corrige cualquier recurso que no esté en el estado deseado.

La ejecución puede ser en modo cliente/servidor o en independiente, llamado “chef-solo”. En el modo cliente/servidor se intercomunican el cliente con el servidor enviando este los atributos. Posteriormente el servidor indexa los atributos enviados por el cliente y proporciona una API para que los clientes puedan acceder a la información.

Es decir, en Chef el modo de despliegue es Pull-based deployment model, al contrario que veíamos en Ansible y Terraform.

2.2.4 PUPPET

Puppet (Abril Nuñez, 2016), es reconocida por muchos como la herramienta de gestión de configuraciones remotas por excelencia. Se trata de una aplicación *open source* desarrollada en Ruby, pero internamente trabaja con DSL (Domain Scripting Language), algo parecido a JSON.

Al igual que Chef, esta tecnología trabaja con el modelo cliente/servidor por lo que necesita de la instalación de procesos de comunicación en los nodos (Pull-based-deployment model).

El enfoque de trabajo está basado en modelos, siendo el diseño del código como una lista de dependencias que ayuda al mejor o peor funcionamiento del script, dependiendo de su caso de uso.

El diseño del código de Puppet funciona como una lista de dependencias que puede ayudar a hacer las cosas mucho más simples o a complicar la situación gravemente. Todo dependerá de la configuración que le apliquemos.

A continuación, se expondrá un ejemplo de trabajo de Puppet:

Puppet está basado en un modelo Pull como se explicó anteriormente, donde los nodos agentes chequean cada 1800 segundos con el nodo master para comprobar si algo necesita ser actualizado en el agente. Si hay alguna actualización pendiente, los agentes descargan el código necesario del master y realizan las acciones correspondientes.

El master debe ser una máquina Linux con Puppet instalado y será la responsable del mantenimiento de las configuraciones. El master solo puede ser Linux.

Los agentes pueden ser configurados independientemente de su OS, aceptando Linux, MacOS, Windows o Solaris. La comunicación entre agentes y master se realiza a través de certificados seguros.

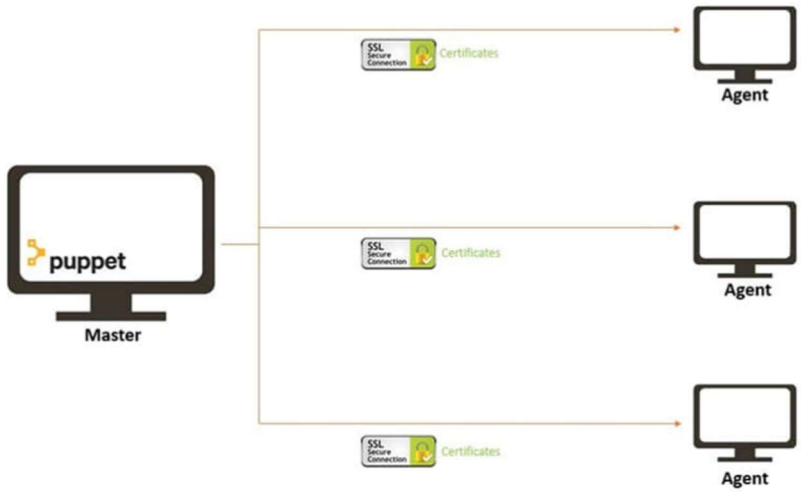


Imagen 6. Comunicacion Puppet

Una vez se ha establecido la comunicación, los agentes envían la información sobre su estado al master, y este analiza si alguno requiere alguna modificación/actualización. Esta información es llamada *Facts,* imagen 7*,* y en ella van atributos como *hostname, kernel, IP address*, etc.…

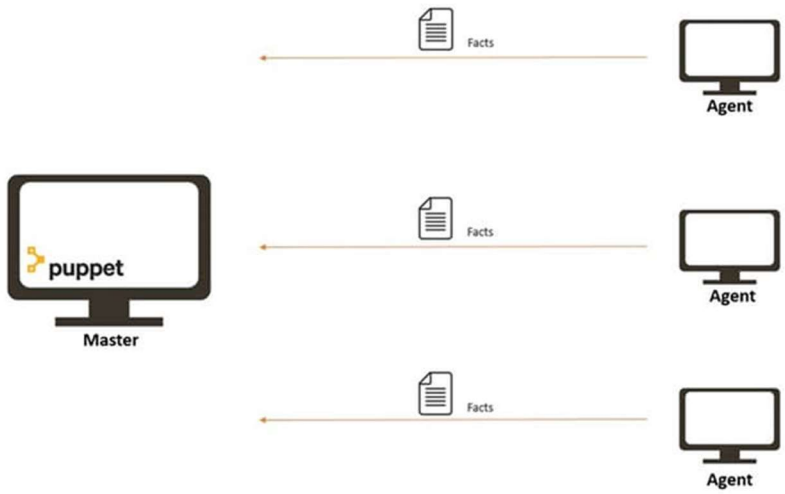


Imagen 7. Envío de Facts

Posteriormente, véase imagen 8 y 9, cuando el master ha recibido esta información, decide realiza una lista, llamada *Catalog* con los cambios a realizar en ese agente. Una vez se ha realizado la lista, esta es aplicada en los agentes.

Imagen 8. Realización del Catalog

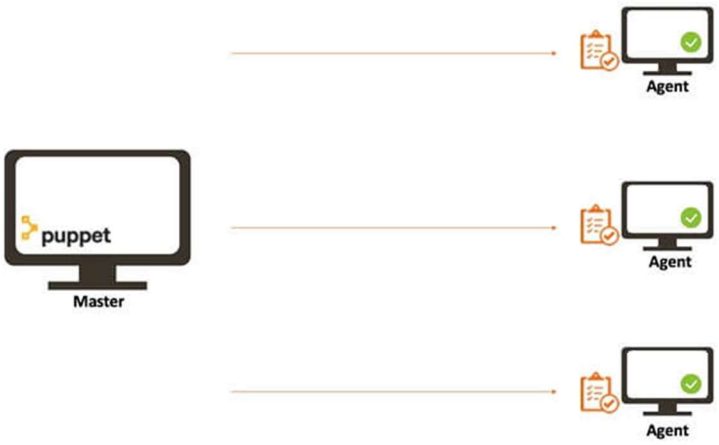


Imagen 9. Aplicación de Catalog

2.2.5 CUESTIONES A TENER EN CUENTA AL TOMAR UNA ELECCIÓN

A continuación, se pretende explicar a grosso modo las grandes diferencias en cuestiones de importancia a la hora de tomar una decisión sobre qué herramienta elegir, y cuales son sus principales similitudes y diferencias.

2.2.5.1 Administración de la configuración vs Aprovisionamiento

Chef, Puppet y Ansible son herramientas de gestión de la configuración, esto quiere decir que su principal objetivo y su función es instalar y gestionar software en servidores ya existentes. En cambio, Terraform es una herramienta de aprovisionamiento, lo que significa que está diseñada para aprovisionar los propios servidores y el resto de su infraestructura equilibradores de carga, las bases de datos, la configuración de redes, etc., dejando el trabajo de configuración de esos servidores a otras herramientas.
Estas dos categorías no son mutuamente excluyentes, ya que la mayoría de las herramientas de gestión de la configuración pueden hacer algún grado de aprovisionamiento y la mayoría de las herramientas de aprovisionamiento pueden hacer algún grado de gestión de la configuración, pero el hecho de que su diseño esté más enfocado a una u otra gestión implica que sus funcionalidades van a ser mayor en su función objetivo que en las otras(Brikman, 2016).

2.2.5.2 Infraestructuras mutables vs Infraestructuras inmutables

Las herramientas de gestión de la configuración, como Puppet, Chef o Ansible suelen adoptar un paradigma de infraestructura mutable. Esto quiere decir que cuando se produce un cambio, por ejemplo, una actualización de un software en un servidor, este cambio se produce sobre ese mismo servidor. Esto puede producir que, con el tiempo, cuántos más cambios se vayan produciendo en los servidores se llegue a un problema conocido como *configuration drift* o deriva de configuración. También quiere decir que cada servidor se vuelve ligeramente diferente a todos los demás, lo que lleva a errores de configuración sutiles que son difíciles de diagnosticar y casi imposibles de reproducir (Brikman, 2016).
En cambio, las infraestructuras inmutables, cada cambio se hace sobre una nueva estructura, que se despliega con la nueva configuración dejando de lado la anterior.

2.2.5.3 Procedural vs Declarativo

Ansible y Chef usan un lenguaje de tipo procedural, esto es, definen a que estado se quiere llegar y como lograr este objetivo, paso por paso. En cambio, Terraform y Puppet usan un lenguaje de tipo declarativo, es decir, definen a qué estado quieren llegar y es la propia herramienta quien decide cómo llegar a ese estado.

2.2.5.4 Master vs Masterless

Por defecto, Chef, Puppet y SaltStack necesitan de un servidor maestro para almacenar el estado de su infraestructura y distribuir las actualizaciones. Cada vez que desee actualizar algo en su infraestructura, es necesario utilizar un cliente, como una herramienta de línea de comandos, para emitir nuevos comandos al servidor maestro, y este es el encargado de distribuir los cambios a los otros servidores.
Esto tiene algunas ventajas, como que sea un lugar único para gestionar tu infraestructura, ya que algunas herramientas como Chef o Puppet cuentan incluso con una interfaz gráfica para poder monitorizar.
Sin embargo, también tiene desventajas, ya que tienes una infraestructura extra que mantener, y riesgo de seguridad pues necesitas mas puertos para su comunicación, nuevas reglas de firewall, etc.(Brikman, 2016) .

Por otro lado, Ansible y Terraform son todos sin maestro por defecto, o para ser más precisos, algunos de ellos pueden depender de un servidor maestro, pero este a es parte de la infraestructura que se está utilizando por lo que no requiere de infraestructura extra como si ocurría anteriormente. Terraform, por ejemplo, se comunica con los proveedores cloud utilizando las API del proveedor, por lo que en cierto modo estas API actúan como maestros, pero esto no requiere de más infraestructura. Ansible funciona conectándose directamente a cada servidor a través de SSH, por lo que no tiene que ejecutar ninguna infraestructura extra ni gestionar mecanismos de autenticación adicionales, solo necesitas de las claves de conexión SSH.

2.2.5.5 Agent vs Agentless

Como ya se ha comentado en apartados anteriores, Chef o Puppet necesitan de instalar un agente que permita el proceso de comunicación entre nodos, por lo que esto requiere de más mantenimiento o seguridad. En cambio, Ansible y Terraform no necesitan de la instalación de ningún agente para esto.

	Source	Cloud	Type	Infrastructure	Language	Agent	Master	Community	Maturity
Chef	Open	All	Config Mgmt	Mutable	Procedural	Yes	Yes	Large	High
Puppet	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	High
Ansible	Open	All	Config Mgmt	Mutable	Procedural	No	No	Huge	Medium
SaltStack	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	Medium
CloudFormation	Closed	AWS	Provisioning	Immutable	Declarative	No	No	Small	Medium
Heat	Open	All	Provisioning	Immutable	Declarative	No	No	Small	Low
Terraform	Open	All	Provisioning	Immutable	Declarative	No	No	Huge	Low

Tabla 2. Tabla comparativa herramientas. Fuente: blog.gruntwork.io

2.3 KERBERIZACIÓN

La kerberización del clúster es el proceso a través del cual se securiza el clúster haciendo uso de un protocolo denominado Kerberos.

Kerberos (MIT, n.d.)es un protocolo de autenticación desarrollado por el MIT que hace uso de criptografía de caves simétricas para autenticar usuarios con los servicios de red,

evitando de este modo el intercambio de contraseñas cifradas a través de la red e intentando frustrar así los intentos de interceptar contraseñas durante el intercambio, práctica usualmente conocida como *man in the middle.*

Los servicios de redes comunes usan un esquema de autenticación basado en contraseñas. Este implica que la autenticación de los usuarios se debe hacer mediante un usuario y una contraseña. Lamentablemente, muchos servicios tratan y transmiten la información de autenticación sin encriptarse previamente.

Una vez que la red se conecte a la Internet, ya no puede asumir que la red es segura y cualquier intruso con acceso a la red y un analizador de paquetes puede interceptar cualquier contraseña enviada sin encriptar.

El objetivo principal de Kerberos es el de eliminar la transmisión a través de la red de información de autenticación. Un uso correcto de Kerberos erradica la amenaza de analizadores de paquetes que intercepten contraseñas en su red (MIT, n.d.).

Kerberos está basado en el protocolo de clave simétrica de Needham-Schroeder, usando a un tercero de confianza, denominado centro de distribución de claves (KDC o Key Distribution Center), formado por dos partes independientes: un servidor de autenticación (AS o Authentication Server) y un servidor emisor de tickets (TGS o Ticket Granting Server). Estos tickets se utilizan para verificar la identidad de los usuarios (Wikipedia, 2021).

Además, Kerberos consta de una base de datos de claves secretas; cada entidad (*principals*) de la red posee una clave secreta, que es conocida únicamente por ella y por Kerberos. Para realizar la comunicación entre dos entidades, Kerberos genera una clave de sesión, con una duración determinada, que pueden usar para cifrar sus comunicaciones.

Como ha sido explicado anteriormente, la decisión de implementar en el proyecto este protocolo se basa tanto en garantizar las condiciones de seguridad especificadas por la compañía, como por un carácter de curiosidad práctica por parte del alumno.

Capítulo 3. DESARROLLO

En este capítulo se procederá a la explicación en profundidad del proyecto llevado a cabo, así como de todas sus partes.

3.1 DESPLIEGUE Y CONFIGURACIÓN DE UN CLÚSTER DATAPROC

Para llegar al objetivo final de este proyecto, a saber, la correcta ejecución de un software big data encargado de la validación de todas las entidades necesarias para el funcionamiento de distintos productos empresariales, es necesario un entorno big data capaz de correr este software sobre él, de manera segura y aislada.

3.1.1 DISEÑO

Este clúster ha sido implementado en Google Cloud Platform (GCP), y para su despliegue y automatización se ha hecho uso de Terraform. Esta elección ha venida impuesta por el director del departamento puesto que es la utilizada frecuentemente en los despliegues de infraestructuras, ya que es una de las herramientas más actualizada, soportada por todos los principales proveedores de cloud y que permite una mayor customización y un mejor seguimiento de los estados.

En este proyecto, dado que se ha hecho uso de una cuenta free trial para su implementación, los nodos levantados han sido bastante humildes para evitar un gran consumo de recursos en la cuenta, pero suficientes para realizar las pruebas necesarias en la ejecución del software de validaciones.

Siguiendo una guía de buena praxis, la implementación ha sido modularizada permitiendo así la creación, modificación o destrucción de las partes aisladamente. Es por ello que se va a proceder a explicar cada fichero de configuración que se ha implementado, para conectarlos luego entre sí. Dado que el proveedor será GPC es necesaria la instalación del SDK de Google, que será usado a la hora de elegir con qué cuenta queremos realizar las operaciones, en qué proyectos y nos permitirá la ejecución de varios comandos que serán necesarios.

Los ficheros a tratar son:

- Fichero de variables: Encargado de alojar todas las variables a las que se va a llamar en los demás ficheros.
- Fichero del módulo *network*: Fichero donde se ha implementado la creación de los recursos referidos a la creación de las redes internas y de los firewalls.
- Fichero del módulo clúster: Fichero donde se ha implementado la creación del clúster, con todas sus características necesarias.
- Fichero Main: fichero principal y donde se llamarán a los dos módulos a crear. Permite elegir qué módulo de este fichero ha de ser llamado

3.1.2 FICHEROS

3.1.2.1 Variables

Este fichero es el encargado de llevar las definiciones de todos aquellos atributos que serán llamadas en la creación de los recursos (*network* y clúster).

Como se puede apreciar en la imagen 11, en primer lugar, definimos una variable *project_id* donde se definirá cuál es el id del proyecto (asignado por Google al crearlo) y que por lo tanto es el proyecto donde se realizará el despliegue. Para conocer este id podemos hacerlo a través de a la API de Google o haciendo uso del SDK como se muestra a continuación.

Tras ejecutar el comando *gcloud init*, imagen 10, podemos apreciar la cuenta, el nombre y el id del proyecto asignado a esa cuenta. Además, también podemos elegir en qué localización mundial queremos alojar estos proyectos. Por costes y cercanía geográfica se ha decidido usar el oeste de Europa. En este caso podemos modificar la configuración que ya tenemos, crear una nueva o cambiar y reinicializar la existente.

```
(base) MacBook-Air-de-andres:TFM_deploy andresjimenez$ gcloud init
Welcome! This command will take you through the configuration of gcloud.

Settings from your current configuration [tmf-deployment] are:
compute:
  region: europe-west4
  zone: europe-west4-a
core:
  account: deploymentstelefonica@gmail.com
  disable_usage_reporting: 'False'
  project: artful-guru-312707

Pick configuration to use:
 [1] Re-initialize this configuration [tmf-deployment] with new settings
 [2] Create a new configuration
 [3] Switch to and re-initialize existing configuration: [default]
Please enter your numeric choice:
```

Imagen 10. Comando gcloud init

Una vez realizado esto, ya se puede saber cual es el id del proyecto, y la zona donde debe ser alojado, en este caso el oeste de Europa sería la región, y la zona sería la a del oeste de Europa. Además, se decide crear un *bucket*, véase imagen 12, a través de la interfaz de la API, en la opción de cloud *storage*, donde almacenar los *logs* de las creaciones y modificaciones que se realicen en el clúster. Este *bucket* será asignado en la variable *staging_bucket* y ha sido llamado "icai_tfm_deployments"

```
variable "project_id" {
  description = "TFM_ICAI (GCP) Project."
  type        = string
  default     = "artful-guru-312707"
}

variable "region" {
  description = "GCP region name."
  type        = string
  default     = "europe-west4"
}

variable "zone" {
  description = "GCP zone name."
  type        = string
  default     = "europe-west4-a"
}


###Almacenar los logs del cluster y como un share warehosuse para workers y master
variable "staging_bucket" {
  description = "Cluster staging bucket ."
  type        = string
  default     = "icai_tfm_deployments"
}
```

Imagen 11. Variables parte 1

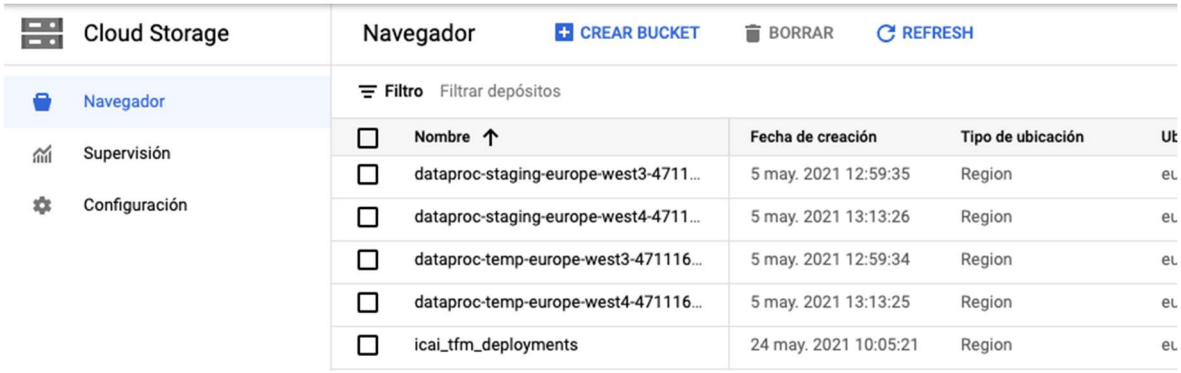


Imagen 12. Creación bucket

En lo referido a las variables del clúster y de las redes, véase imagen 13, se puede apreciar como se puede definir una lista de variables a las que ser llamadas, como el nombre, el número de instancias preemtible, o el OS que se va a usar.

En este caso, el nombre es “deployments-cluster-Terraform”, el número de instancias preemtible es 0 y el sistema operativo utilizado es Debian 1.5. Por defecto, si no se especifica sistema operativo usa Debian. Las instancias preemtible son un tipo de nodos

que Google nos ofrece a un precio inferior al normal, pero que podrán ser desasignados cuando Google necesite más recursos. En esta ocasión dado que no necesitamos más recursos y que queremos garantizar que se ejecute correctamente asegurando la presencia de nodos, no hacemos uso de esta opción. En cuanto a la versión Debian usada, ha sido elegida a conciencia debido a la versión de Spark y de Scala que trae, 2.4 y 2.12 respectivamente, y que son necesarias para la ejecución del software de validaciones.
En un primer momento fue elegida una imagen Debian con versión 2.0 pero se descubrió que el fallo en la ejecución del software se debía a que esta traía consigo una versión de Spark superior que no era válida.

```
# Cluster Variables
variable "dataproc_cluster" {
  description = "Dataproc Cluster Properties"
  type        =  object(
    {
      name                       = string
      preemptible_num_instances = number
      image_version              = string
    }
  )
  default     = {
    name                       = "deployments-cluster-terraform"
    preemptible_num_instances = 0
    image_version              = "1.5"
  }
}
```

Imagen 13. Variables parte 2

En cuanto a la red y subred creadas, véase imagen 14, se ha creado una red a la que ha sido llamado “network-icai” y donde la configuración necesaria es el tamaño máximo de los paquetes que será transmitidos a través de la red (MTU) y que ha sido asignado a 1460,valor mínimo permitido, y el “routing_mode” el cual podía ser global o regional. Si se establece en REGIONAL, los enrutadores en la nube de esta red sólo anunciarán rutas con subredes de esta red en la misma región que el enrutador. Si se establece como GLOBAL, los enrutadores en la nube de esta red anunciarán rutas con todas las subredes de esta red, en todas las regiones, y en este caso solo hará falta que enruten a nivel regional.

Por defecto, al crear una red, si no establecemos una creación de subredes manual, el propio proveedor crea una subred por debajo, a la cual van a pertenecer todos los nodos del clúster. En este caso se ha decidido crear manualmente, estableciendo su nombre, "subnetwork-icai" y su rango de IP, 10.20.0.0/16. Este rango con una mascara de 16 bits se ha elegido debido a que se quieren dejar libres los dos últimos octetos para ofrecer la posibilidad de crear más subredes, por ejemplo, para alojar maquinas virtuales. Por lo tanto, la máscara establecida es: 11111111.11111111.00000000.00000000 teniendo libres las dos últimas subredes y con una dirección broadcast 10.20.255.255, una dirección router 10.20.0.1.

```
variable "dataproc_network" {
  description = "Dataproc Netoworks Properties"
  type        =  object(
    {
      name                      = string
      mtu                       = number
      routing_mode              = string
    }
  )
  default     = {
    name                      = "network-icai"
    mtu                       = 1460
    routing_mode              = "REGIONAL"
  }
}

variable "dataproc_subnetwork" {
  description = "Dataproc Netoworks Properties"
  type        =  object(
    {
      name                      = string
      ip_range                  = string
    }
  )
  default     = {
    name                      = "subnetwork-icai"
    ip_range                  = "10.20.0.0/16"
  }
}
```

Imagen 14. Variables parte 3

Para continuar con del fichero de variables definidas, en la imagen 15 vemos en primer lugar como se configuran unas propiedades del entorno Hadoop, concretamente Spark y Yarn.

En este caso, se está configurando:

- yarn.resourcemanager.am.max-attempts: el máximo numero de intentos del *resource manager*, que es el número máximo de intentos que se le da al *application master* de correr la aplicación. Es un ajuste global para todos los maestros de aplicación (Hadoop, n.d.).
- spark.task.maxFailures: número de fallos de una tarea concreta antes de abandonar el trabajo. El número total de fallos repartidos entre diferentes tareas no hará que el trabajo falle; una tarea en particular tiene que fallar este número de intentos. Debe ser mayor o igual a 1. Número de reintentos permitidos = este valor – 1 (Spark, n.d.).
- spark.stage.maxConsecutiveAttempts: número de intentos de etapa consecutivos permitidos antes de abortar una etapa (Spark, n.d.).

```
variable "override_properties" {
  description = "Cluster Properties"
  type        = map
  default     = {
    "dataproc:yarn.log-aggregation.enabled" = "true",
    "yarn:yarn.resourcemanager.am.max-attempts" = 10,
    "spark:spark.task.maxFailures"=10,
    "spark:spark.stage.maxConsecutiveAttempts" = 10
  }
}
```

Imagen 15. Variables parte 4

Respecto a la configuración de los nodos, tanto máster (1 instancia) como *workers* (2 instancias*)*, se puede apreciar en la imagen 16 la configuración que les ha sido asignada. En ambos se ha configurado un tipo de máquina n1-standard-4, una máquina bastante

pequeña en cuanto a recursos con 15Gb de memoria RAM y 4 cores, suficiente para las tareas que se quieren ejecutar. Además, cada uno tiene 35 Gb de disco duro SDD (por defecto si no se indica un HDD o un SDD Premium).

```
variable "master_node" {
  description = "GCP Worker VM instance machine type."
  type        = object(
    {
      machine_type      = string
      num_instances     = number
      boot_disk_size_gb = number
    }
  )
  default     = {
    machine_type      = "n1-standard-4"
    num_instances     = 1
    boot_disk_size_gb = 35
  }
}

variable "worker_node" {
  description = "GCP Worker VM instance machine type."
  type        = object(
    {
      machine_type      = string
      num_instances     = number
      boot_disk_size_gb = number
    }
  )
  default     = {
    machine_type      = "n1-standard-4"
    num_instances     = 2
    boot_disk_size_gb = 35
  }
}
```

Imagen 16. Variables parte 5

Por último, imagen 17, podemos apreciar la etiqueta que hemos dado a nuestro clúster, algo útil de cara a tener identificadas las máquinas para la aplicación de reglas de firewall.

```
variable "tags" {
  description = "List of tags instance."
  type        = list
  default     = ["dataproc"]
}
```

Imagen 17. Variables parte 6

3.1.2.2 Módulos

Una vez se han definido todas las variables, se procede a la creación de los dos módulos necesarios, clúster y networks, donde en cada uno se llamará a las variables que se van a necesitar.

3.1.2.2.1 Clúster

En este modulo se definirá el recurso o los recursos necesarios para la creación del clúster dataproc.

En primer lugar, como se ha comentado, es necesario hacer una llamada a las variables definidas en el fichero Variables que son necesarias en este módulo, como se aprecia en siguiente imagen.

```
variable "project_id" {}
variable "region" {}
variable "zone" {}
variable "tags" {}
variable "dataproc_cluster" {}
variable "master_node" {}
variable "worker_node" {}
variable "staging_bucket" {}
variable "override_properties" {}
variable "dataproc_subnetwork" {}
```

Imagen 18. Clúster parte 1

Posteriormente, imagen 19, se crea el recurso "Google_dataproc_cluster", al que se le pasa el nombre y el id del proyecto y la región donde se quiere crear. Una vez hecho esto, se define, opcionalmente, un *staging bucket* para almacenar la información de creación, modificación etc., que en este caso es el creado anteriormente.

Una vez hecho esto, se crean los nodos master y *workers*, con las variables definidas y nombradas anteriormente, configurando el número de instancias, el tipo de máquina y el tamaño de disco.

```
# Create a google dataproc cluster
resource "google_dataproc_cluster" "deployments_cluster" {
  project = var.project_id
  name   = var.dataproc_cluster["name"]
  region = var.region

  cluster_config {
    staging_bucket = var.staging_bucket

    master_config {
      num_instances = var.master_node["num_instances"]
      machine_type  = var.master_node["machine_type"]
      disk_config {
        boot_disk_size_gb = var.master_node["boot_disk_size_gb"]
      }
    }

    worker_config {
      num_instances    = var.worker_node["num_instances"]
      machine_type     = var.worker_node["machine_type"]
      disk_config {
        boot_disk_size_gb = var.worker_node["boot_disk_size_gb"]
      }
    }
```

Imagen 19. Clúster parte 2

Una vez definido esto, se fijará el número de instancias *preemtible,* que se fijó a 0, en la parte de software, la imagen de OS que se desea, fijada una Debian 1.5 por lo ya explicado anteriormente, y las propiedades de Spark y Yarn fijadas. Por último, la etiqueta que le damos al clúster para su identificación en reglas de seguridad.

```
  preemptible_worker_config {
    num_instances = var.dataproc_cluster["preemptible_num_instances"]
  }

  software_config {
    image_version       = var.dataproc_cluster["image_version"]
    override_properties = var.override_properties
  }

  gce_cluster_config {
    tags                    = var.tags
    zone                    = var.zone
    subnetwork       = var.dataproc_subnetwork["name"]
  }
```

Imagen 20. Clúster parte 3

En último lugar, se configura la kerberización del clúster. Para ello es necesario crear un llavero de claves KMS, llamado "icai_kms", a través de la interfaz de Google Cloud, en el apartado *key managment,* o por comandos en el SDK. En este caso el llavero y la clave de encriptación de este, llamada "icai_kms_kerberos", han sido creados a través de la interfaz, usando clave simétrica, nivel de protección software (también hay HSM) y 90 días de duración. Además, se configura un realm al que se le llama "ICAI_TFM" y un tiempo de vida de los tickets de 8 horas. Es importante establecer la opción de *enable_kerberos = "true"* ya que por defecto viene a *false* y de este modo nunca se habilitará.

Una vez creado el llavero tenemos que hacer lo siguiente, imagen 22, para encriptar la contraseña principal de kerberos con la clave de encriptación que hemos creado en nuestro KMS y subirla al *bucket* (*staging bucket)* mediante el comando **gsutil cp kerberos-root-principal-password.encrypted gs://icai_tfm_deployments.**

```
security_config {
  kerberos_config {
    kms_key_uri = "projects/artful-guru-312707/locations/global/keyRings/icai_kms/cryptoKeys/icai_kms_kerberos"
    root_principal_password_uri = "gs://icai_tfm_deployments/kerberos-root-principal-password.encrypted"
    enable_kerberos = "true"
    tgt_lifetime_hours = 8
    realm = "ICAI_TFM"
  }
}
```

Imagen 21. Clúster parte 4

```
echo "clave" | \
  gcloud kms encrypt \
    --location=europe-west4 \
    --keyring=icai_kms \
    --key=icai_kms_kerberos \
    --plaintext-file=- \
    --ciphertext-file=kerberos-root-principal-password.encrypted
```

Imagen 22. Clúster parte 5

La decisión de kerberizar es clúster es tomada para garantizar la seguridad de autenticación en los servicios Hadoop como HDFS o SPARK en el momento de ejecutar el software de validaciones y cumplir con los estándares de seguridad exigidos en la compañía.

3.1.2.2.2 Network

El modulo *network*, es dónde serán definidas la red y la subred que se quieren crear, así como sus reglas de seguridad.

Para comenzar, al igual que en el módulo clúster se han de llamar a las variables que quieren utilizar, imagen 23. Vemos como en este módulo solo nos hace falta aquellas que tienen relación con las redes, y la región.

```
variable "region" {}
variable "dataproc_subnetwork" {}
variable "dataproc_network" {}
```

Imagen 23. Network parte 1

Una vez han sido referenciadas pasamos a definir los dos recursos, es decir, la red y la subred. En primer lugar, se crea la red, con el recurso llamado "Google_compute_network*"* y se le asigna un nombre, MTU, y routing mode a través de la variable "dataproc_network*"definida* en el fichero Variables. Se aprecia como además hay una variable nueva llamada "auto_create_subnetworks*"* que se ha puesto a *false* debido a que no se quiere que se creen las subredes automáticamente si no que sean creadas por el usuario.

En segundo lugar, se crea el recurso de las subredes llamado "Google_compute_subnetwork *"*y al que se le pasa el nombre, el rango, la región donde se

quiere crear la subred, la red a la que pertenece, en este caso la que acaba de ser definida anteriormente, y que para obtener su nombre se observa como se puede llamar al atributo *name* del recurso creado de la mediante *Google_compute_network.network_cluster.name* y por último se crea una red secundaria dentro de la subred, que podría ser útil en algún momento si se desea separar, por ejemplo, máquinas virtuales en otra red, aunque es algo que no va se va a ocupar este proyecto y que se ha implementado por pura curiosidad.

```
resource "google_compute_network" "network_cluster" {
  name                    = var.dataproc_network["name"]
  mtu                     = var.dataproc_network["mtu"]
  auto_create_subnetworks = false
  routing_mode            = var.dataproc_network["routing_mode"]
}


resource "google_compute_subnetwork" "subnetwork_cluster" {
  name          = var.dataproc_subnetwork["name"]
  ip_cidr_range = var.dataproc_subnetwork["ip_range"]
  region        = var.region
  network       = google_compute_network.network_cluster.name
  secondary_ip_range {
    range_name    = "tf-test-secondary-range-update1"
    ip_cidr_range = "192.168.10.0/24"
  }
}
```

Imagen 24. Network parte 2

Una vez han sido creadas la red y la subred, se necesita securizar el acceso a estas, y asegurar que las conexiones permitidas son únicamente entre elementos de la subred, en este caso los nodos, permitiendo así la configuración necesaria entre el master y los workers para el correcto despliegue de la infraestructura y las posteriores conexiones necesarias, y, por otro lado, el acceso desde el exterior a los nodos por parte del usuario. Como se puede apreciar en la imagen 25, han sido definidas dos reglas de firewall, una encargada de las conexiones externas y la otra de las internas. El primer recurso es el encargado de las externas y se le puede diferenciar en el nombre que se le ha asignado al recurso, “firewall_network_external”. Para su creación es necesario asignarle un nombre a la regla, en este caso se ha llamado *external*, el nombre del clúster al que se le ha de aplicar

que se obtiene a través del objeto "network_cluster "llamando a su atributo *name,* y la IP pública desde la que se va a conectar el usuario, en este caso la 83.57.210.XXX/32. *Tips*: usar el comando *curl ifconfig.me* para usuarios MacOS y Linux para conocer su ip pública, también se puede buscar webs como esta.

A continuación, debemos establecer qué protocolo de transferencia de datos queremos usar (TCP o UDP), en este caso se ha elegido TCP, aunque podría usarse UDP sin ningún problema. Como ventaja el protocolo TCP garantiza la recepción de las tramas correctamente al contrario que UDP, pero es más lento. Por último, se habilita el puerto 22 para permitir que esta IP pública que hemos definido se pueda conectar a través de SSH a cualquiera de los nodos, conociendo su dirección IP externa.

En cuanto a la red interna, se asigna como rango de IP el rango que ha sido definido en la variable "dataproc_subnetwork", 10.20.0.0/16 y se permiten todos los protocolos y puertos ya que van a ser necesarios una gran variedad para la comunicación entre nodos y posteriormente para la kerberización.

```
resource "google_compute_firewall" "firewall_network_external" {
  name    = "external"
  network = google_compute_network.network_cluster.name
  source_ranges = ["83.57.210.XXX/32"]

  allow {
    protocol = "tcp"
    ports = ["22"]

  }

}

resource "google_compute_firewall" "firewall_network_internal" {
  name    = "internal"
  network = google_compute_network.network_cluster.name
  source_ranges = [var.dataproc_subnetwork["ip_range"]]

  allow {
    protocol = "all"

  }

}
```

Imagen 25. Network parte 3

3.1.2.2.3 Main

En última instancia se procede a explicar la codificación del fichero Main, el cual va a ser el ejecutado por la herramienta, y en el que se ha de llamar a los dos módulos, o a cada uno independientemente.

Como se puede apreciar en la imagen 26, lo primero que se ha de hacer es definir cual va a ser el proveedor cloud a utilizar, en este caso es Google, en su versión 3.57, y donde se definen el id del proyecto, la región y la zona llamando a las variables definidas en el fichero Variables.

Desde el fichero Main se pueden llamar a las variables directamente sin tener que importarlas a este fichero como si ocurría en los módulos clúster y *network*. Esto es porque es realmente desde este fichero desde donde se definen qué variables van a poder ser importadas en los módulos. Al definir el modulo *network* se ha de especificar la ruta en la que está el fichero, en este caso dentro de una carpeta Network, y qué variables va a necesitar, que son las que posteriormente serán importadas y llamadas en este fichero.

Los mismo ocurre con el módulo clúster, donde se especifica la ruta donde se encuentra el fichero y todas las variables que se van a permitir importar y por lo tanto llamar desde ese fichero.

```
provider "google" {
  version = "3.57"
  project = var.project_id
  region  = var.region
  zone    = var.zone
}

module "network" {
  source                          = "./network"
  region                          = var.region
  dataproc_network                = var.dataproc_network
  dataproc_subnetwork             = var.dataproc_subnetwork
}

# Create Cluster module
module "cluster" {
  source                          = "./cluster"
  project_id                      = var.project_id
  region                          = var.region
  zone                            = var.zone
  tags                            = var.tags
  dataproc_cluster                = var.dataproc_cluster
  master_node                     = var.master_node
  worker_node                     = var.worker_node
  staging_bucket                  = var.staging_bucket
  override_properties             = var.override_properties
  dataproc_subnetwork             = var.dataproc_subnetwork
  depends_on = [module.network]
}
```

Imagen 26. Fichero Main

3.1.3 DESPLIEGUE Y COMANDOS

Una vez se han definido todos los ficheros necesarios se procede al despliegue de la infraestructura. Para ello es necesario abrir una terminal y situarse a la altura de donde se encuentre el fichero Variables y el fichero Main, ya que posteriormente será desde el fichero Main donde se vaya a buscar los módulos en los directorios especificados. A modo de ejemplificación véase la imagen 27 donde se aprecia como en ese directorio se encuentran (flecha roja) el fichero Variables, el fichero Main, y los dos subdirectorios clúster y *network* donde dentro se alojan los ficheros de los módulos.
Se han señalado también dos ficheros (flecha azul) los cuales se crean tras la primera ejecución de Terraform. Este fichero estado y su *backup* son una funcionalidad muy importante de Terraform que se encarga de almacenar el estado de los despliegues que se hayan hecho evitando así el choque entre despliegues iguales por distintos usuarios y

garantizando que todos los usuarios vean el mismo estado de un despliegue. Esto es lo explicado en el apartado *2.2.5.4 Master vs Masterless.*

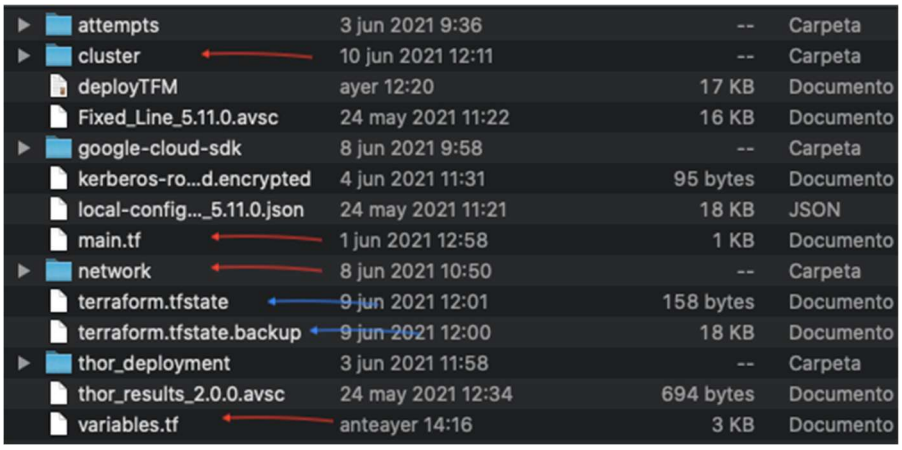


Imagen 27. Estructura directorios

Una vez explicado esto, se procede a mostrar los comandos necesarios para el despliegue de la infraestructura, una vez situado ya en el directorio referido anteriormente, en este caso, *TFM_deploy,* imagen 28.

```
(base) MacBook-Air-de-andres:TFM_deploy andresjimenez$ ls
Fixed_Line_5.11.0.avsc                          main.tf
attempts                                        network
cluster                                         terraform.tfstate
deployTFM.rtf                                   terraform.tfstate.backup
google-cloud-sdk                                thor_deployment
kerberos-root-principal-password.encrypted      thor_results_2.0.0.avsc
local-config-file-Fixed_Line_5.11.0.json        variables.tf
(base) MacBook-Air-de-andres:TFM_deploy andresjimenez$
```

Imagen 28. Directorio actual

Una vez situado en dicho directorio se ejecuta el inicializador de Terraform, con el comando Terraform *init*, a través del cual se va a llamar al fichero Main (cualquiera de los comandos que se van a presentar llaman al fichero Main) y se va a comprobar que el proveedor definido, su versión e id de proyecto son correctas y aptas para el despliegue.

En la siguiente imagen se aprecia cual ha de ser la salida. Se ve como se avisa de que la versión del proveedor que se está usando esta deprecada y en breve deberá de usarse una superior.

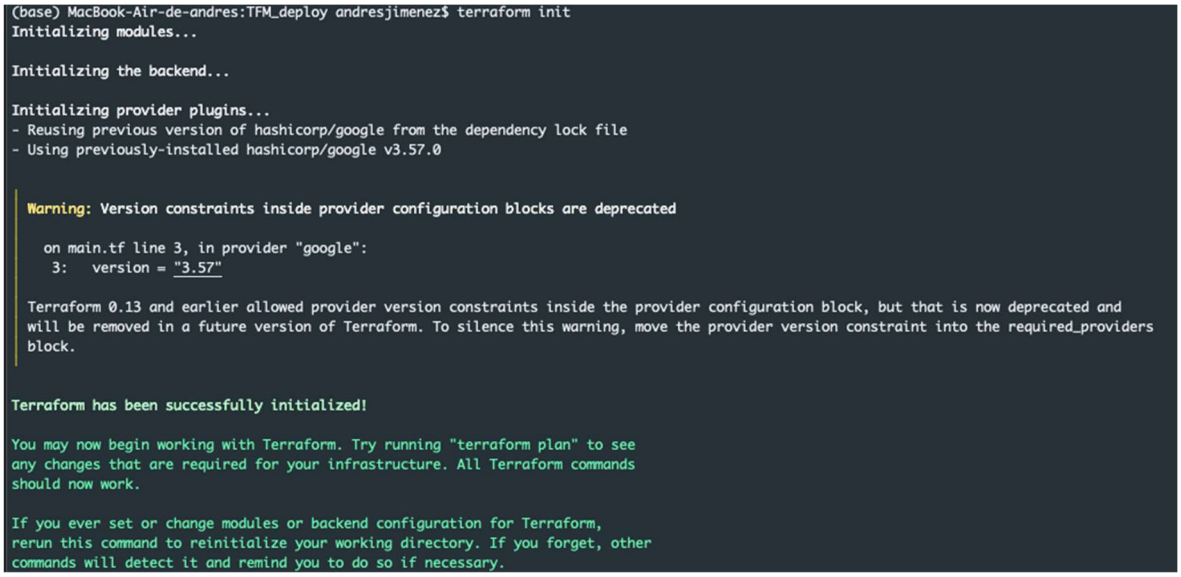


Imagen 29. Terraform *init*

Una vez inicializado, se procede a ejecutar el comando Terraform *plan*, el cual nos permite visualizar un *planning,* véase imagen de la 31 a la 33*,* de la estructura que se va a desplegar. A continuación, se muestra lo que se debería de visualizar, 5 creaciones nuevas. Se aprecia como aparece el símbolo de + a lado de cada línea ya que son todo creaciones nuevas que se van a llevar a cabo. En este paso es donde se debe comprobar que todos los atributos a los que se ha de asignar el valor definido en las variables a las que se llaman han sido asignadas correctamente por con los valores esperados.

```
(base) MacBook-Air-de-andres:TFM_deploy andresjimenez$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # module.cluster.google_dataproc_cluster.deployments_cluster will be created
  + resource "google_dataproc_cluster" "deployments_cluster" {
      + graceful_decommission_timeout = "0s"
      + id                            = (known after apply)
      + labels                        = (known after apply)
      + name                          = "deployments-cluster-terraform"
      + project                       = "artful-guru-312707"
      + region                        = "europe-west4"

      + cluster_config {
          + bucket         = (known after apply)
          + staging_bucket = "icai_tfm_deployments"
          + temp_bucket    = (known after apply)

          + gce_cluster_config {
              + internal_ip_only       = false
              + network                = (known after apply)
              + service_account_scopes = (known after apply)
              + subnetwork             = "subnetwork-icai"
              + tags                   = [
                  + "dataproc",
                ]
              + zone                   = "europe-west4-a"
            }

          + master_config {
              + image_uri        = (known after apply)
              + instance_names   = (known after apply)
              + machine_type     = "n1-standard-4"
              + min_cpu_platform = (known after apply)
              + num_instances    = 1

              + disk_config {
                  + boot_disk_size_gb = 35
                  + boot_disk_type    = "pd-standard"
                  + num_local_ssds    = (known after apply)
                }
            }
```

Imagen 30. Terraform *plan parte 1*

```
    + preemptible_worker_config {
        + instance_names = (known after apply)
        + num_instances  = 0

        + disk_config {
            + boot_disk_size_gb = (known after apply)
            + boot_disk_type    = (known after apply)
            + num_local_ssds    = (known after apply)
          }
      }

    + security_config {
        + kerberos_config {
            + enable_kerberos             = true
            + kms_key_uri                 = "projects/artful-guru-312707/locations/global/keyRings/icai_kms/cryptoKeys/icai_kms_kerberos"
            + realm                       = "ICAI_TFM"
            + root_principal_password_uri = "gs://icai_tfm_deployments/kerberos-root-principal-password.encrypted"
            + tgt_lifetime_hours          = 8
          }
      }

    + software_config {
        + image_version       = "1.5"
        + override_properties = {
            + "dataproc:yarn.log-aggregation.enabled"     = "true"
            + "spark:spark.stage.maxConsecutiveAttempts"  = "10"
            + "spark:spark.task.maxFailures"              = "10"
            + "yarn:yarn.resourcemanager.am.max-attempts" = "10"
          }
        + properties          = (known after apply)
      }

    + worker_config {
        + image_uri        = (known after apply)
        + instance_names   = (known after apply)
        + machine_type     = "n1-standard-4"
        + min_cpu_platform = (known after apply)
        + num_instances    = 2

        + disk_config {
            + boot_disk_size_gb = 35
            + boot_disk_type    = "pd-standard"
            + num_local_ssds    = (known after apply)
          }
      }
}
```

Imagen 31. Terraform *plan parte 2*

```
# module.network.google_compute_firewall.firewall_network_external will be created
+ resource "google_compute_firewall" "firewall_network_external" {
    + creation_timestamp = (known after apply)
    + destination_ranges = (known after apply)
    + direction          = (known after apply)
    + enable_logging     = (known after apply)
    + id                 = (known after apply)
    + name               = "external"
    + network            = "network-icai"
    + priority           = 1000
    + project            = (known after apply)
    + self_link          = (known after apply)
    + source_ranges      = [
        + "83.57.210.127/32",
      ]

    + allow {
        + ports    = [
            + "22",
          ]
        + protocol = "tcp"
      }
  }

# module.network.google_compute_firewall.firewall_network_internal will be created
+ resource "google_compute_firewall" "firewall_network_internal" {
    + creation_timestamp = (known after apply)
    + destination_ranges = (known after apply)
    + direction          = (known after apply)
    + enable_logging     = (known after apply)
    + id                 = (known after apply)
    + name               = "internal"
    + network            = "network-icai"
    + priority           = 1000
    + project            = (known after apply)
    + self_link          = (known after apply)
    + source_ranges      = [
        + "10.20.0.0/16",
      ]

    + allow {
        + ports    = []
        + protocol = "all"
      }
  }
```

Imagen 32. Terraform *plan parte 3*

```
  # module.network.google_compute_network.network_cluster will be created
  + resource "google_compute_network" "network_cluster" {
      + auto_create_subnetworks         = false
      + delete_default_routes_on_create = false
      + gateway_ipv4                    = (known after apply)
      + id                              = (known after apply)
      + mtu                             = 1460
      + name                            = "network-icai"
      + project                         = (known after apply)
      + routing_mode                    = "REGIONAL"
      + self_link                       = (known after apply)
    }

  # module.network.google_compute_subnetwork.subnetwork_cluster will be created
  + resource "google_compute_subnetwork" "subnetwork_cluster" {
      + creation_timestamp         = (known after apply)
      + fingerprint                = (known after apply)
      + gateway_address            = (known after apply)
      + id                         = (known after apply)
      + ip_cidr_range              = "10.20.0.0/16"
      + name                       = "subnetwork-icai"
      + network                    = "network-icai"
      + private_ipv6_google_access = (known after apply)
      + project                    = (known after apply)
      + region                     = "europe-west4"
      + secondary_ip_range         = [
          + {
              + ip_cidr_range = "192.168.10.0/24"
              + range_name    = "tf-test-secondary-range-update1"
            },
        ]
      + self_link                  = (known after apply)
    }

Plan: 5 to add, 0 to change, 0 to destroy.
```

Imagen 33. Terraform *plan parte 4*

Tras comprobar que todo ha sido modificado correctamente se procede a ejecutar el último comando, Terraform *apply,* a través del cual se aplicarán todos los cambios y de desplegará el clúster. Este comando permite, gracias a la modularización que se ha hecho, llamar únicamente a la construcción de un modulo. Esto se haría del siguiente modo: Terraform *apply --target=module.cluster* o Terraform *apply --target=module.network*.

En nuestro caso vamos a desplegar todo al mismo tiempo por lo que se ejecuta Terraform *apply.* Volverá a mostrar por pantalla el *planning* visto anteriormente con el comando *terrafrom plan* y nos preguntará si queremos aplicar los cambios a lo que respondemos *yes.* Habrá que esperar en torno a unos 15 minutos en lo que se despliegan ambos módulos. Se recomienda si da algún problema de *time out* revisar las políticas de firewall internas y volver a destruir todo lo creado mediante el comando Terraform *destroy,* que también puede ser usado con la opción --target y destruir los módulos individualmente.
Posteriormente volver a desplegarlo, y cuando todo haya funcionado correctamente debería de aparecer por pantalla el siguiente mensaje, imagen 34.

Imagen 34. Terraform *apply*

Una vez esté desplegado se recomienda comprobar en la interfaz web que se ha creado todo correctamente. Para comprobar que el nodo master y los dos workers han sido correctamente desplegados es posible acceder al apartado de *compute engine.*
Se puede observar, imagen 35, como se han creado correctamente todas las instancias. Para conocer si IP pública a la que conectarnos podemos acceder a cada una de ellas y nos dará toda la información sobre esa instancia. Otra forma, imagen 36, sería a través de los comandos de Google SDK por terminal: gcloud compute instances list

Imagen 35. Compute engine

```
(base) MacBook-Air-de-andres:TFM_deploy andresjimenez$ gcloud compute instances list
NAME                                ZONE            MACHINE_TYPE   PREEMPTIBLE  INTERNAL_IP  EXTERNAL_IP      STATUS
deployments-cluster-terraform-m     europe-west4-a  n1-standard-4               10.20.0.4    35.204.198.158   RUNNING
deployments-cluster-terraform-w-0   europe-west4-a  n1-standard-4               10.20.0.3    34.141.223.198   RUNNING
deployments-cluster-terraform-w-1   europe-west4-a  n1-standard-4               10.20.0.2    34.90.162.171    RUNNING
```

Imagen 36. Comando SDK list instances

Otra parte importante que comprobar si su creación ha sido correcta es la de las redes y las políticas de firewall ya que de estas depende que se permita la conexión desde el exterior. Para ello accedemos al apartado *Redes de VPC* donde deberíamos de encontrar una red creada por defecto para todos aquellos proyectos donde no se configure una propia, y nuestra red creada llamada *icai-network.*

En la siguiente imagen, 37, podemos apreciar que se ha creado este recurso y que contiene una subred llamada *subnetwork-icai* con dos rangos de direcciones IP, el principal (10.20.0.0/16) y el secundario (192.168.10.0/24).

Nombre	Región	Subredes	MTU	Modo	Rangos de direcciones IP	Puertas de enlace	Reglas de firewall
default		27	1460	Automática			4
network-icai		1	1460	Personalizada			2
	europe-west4	subnetwork-icai			10.20.0.0/16, 192.168.10.0/24	10.20.0.1	

Imagen 37. Redes VPC

Al pinchar en esta red, imagen 38, podemos acceder a las reglas firewall configuradas y ver que se han definido correctamente.

CCIONES IP ESTÁTICAS INTERNAS | POLÍTICAS DE FIREWALL | REGLAS DE FIREWALL | RUTAS | INTERCAMBIO DE TRÁFICO ENTRE R

AGREGAR REGLA DE FIREWALL | BORRAR

Filtro Ingresar el nombre o el valor de la propiedad

Nombre	Tipo	Destinos	Filtros	Protocolos/puertos	Acción	Prioridad	Registros
external	Entrada	Aplicar a tod	Intervalos de IP: 1	tcp:22	Permitir	1000	Desactivado
internal	Entrada	Aplicar a tod	Intervalos de IP: 1	all	Permitir	1000	Desactivado

Imagen 38. Reglas firewall

Una vez comprobado lo anterior, procedemos a conectarnos al nodo master. Para ello lo hacemos a través del puerto 22 mediante SSH y un par de claves publico privadas generadas por Google y añadidas en el apartado *SSH Keys,* añadiendo la clave pública y el usuario con el que vamos a conectarnos, imagen 39.

```
andresjimenez
```

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDBuiUzP
EZiXX+tcLrk9bLf9eTXRHnY0vHjzHJsxdo/Kk8I15QyFy
IreJJNQBYij0ZbhQ+UqJ1AVV4ZedgWxO4fL6kbMeQungJ
SqphMODhpt03zPdpxpe5Sk7xWUaZQq2jAvZPJZYy63Z+6
CepVU8hO8A/Rzx6O/2egCw0zXN53x5pZkOZYwkJd5NewI
dVkkq2hDCb6t5neZ/E187m3hgL4ObQsEIHOafHUkzlQ6l
9sh2XeFULlPNHVAkwRa9I8Il0q8vLNIh7jfr2imTq3q2j
```

Imagen 39. SSH keys

De este modo, imagen 40, es posible conectarse al nodo master del clúster, mediante el comando ssh -i *pathclaveprivada* 35.204.198.158, siendo esta última la IP publica efímera que tiene el nodo master.

```
(base) MacBook-Air-de-andres:TFM_deploy andresjimenez$ ssh -i ~/.ssh/google_compute_engine 35.204.198.158
The authenticity of host '35.204.198.158 (35.204.198.158)' can't be established.
ECDSA key fingerprint is SHA256:zAaRBo5rPIcy/5DbP7Czrv0YI2hvsV8S5cuJ5++WPaM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '35.204.198.158' (ECDSA) to the list of known hosts.
Enter passphrase for key '/Users/andresjimenez/.ssh/google_compute_engine':
Linux deployments-cluster-terraform-m 5.10.0-0.bpo.5-amd64 #1 SMP Debian 5.10.24-1~bpo10+1 (2021-03-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
andresjimenez@deployments-cluster-terraform-m:~$
```

Imagen 40. Conexión ssh

Una vez dentro es posible comprobar que hay comunicación entre nodos haciendo un ping, por ejemplo, del nodo master a uno de los workers, concretamente se realiza al woker 0 con ip privada 10.20.0.3, véase siguiente imagen.

```
andresjimenez@deployments-cluster-terraform-m:~$ ping 10.20.0.3
PING 10.20.0.3 (10.20.0.3) 56(84) bytes of data.
64 bytes from 10.20.0.3: icmp_seq=1 ttl=64 time=1.26 ms
64 bytes from 10.20.0.3: icmp_seq=2 ttl=64 time=0.306 ms
64 bytes from 10.20.0.3: icmp_seq=3 ttl=64 time=0.303 ms
64 bytes from 10.20.0.3: icmp_seq=4 ttl=64 time=0.375 ms
^C
--- 10.20.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 48ms
rtt min/avg/max/mdev = 0.303/0.560/1.258/0.404 ms
```

Imagen 41. Ping master-worker

En último lugar, es necesario comprobar si la kerberización del clúster se ha realizado correctamente y la posterior creación y configurar para el usuario administrador y encargado de la ejecución del software de validaciones una vez se desplegado.
Para ello se debe acceder al fichero de configuración de kerberos, alojado en/etc/krb5.conf.Al hacer un *cat /etc/krb5.conf*, imagen 42, se ve como se ha configurado como REALM por defecto el *ICAI_TFM*, con el *lifetime* de 8h configurado, por defecto el número de días para la renovación son 7. También se aprecia como se ha añadido el

dominio del kdc y el admin server formado por el nombre del nodo master e información extra añadida automaticamente.

```
andresjimenez@deployments-cluster-terraform-m:~$ cat /etc/krb5.conf
[libdefaults]
        default_realm = ICAI_TFM
        ticket_lifetime = 8h
        renew_lifetime = 7d
        default_ccache_name = FILE:/tmp/krb5cc_%{uid}

[realms]
       ICAI_TFM = {
               kdc = deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal
               admin_server = deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal
       }
```

Imagen 42. Krb5.conf

Para poder ver qué servicios se han añadido y agregar el usuario administrador que se va a autenticar y va a poder hacer uso de los diferentes servicios kerberizados se han de ejecutar los siguientes comandos con permisos de administración, imagen 43 y 44.

En primer lugar, en la imagen 43, se pueden observar los *principals* y *service principals* que han sido configurados por defecto al kerberizar. Todos los servicios principales de Hadoop como HDFS, Yarn, Spark, Hive o Map Reduce han sido configurados. Además, se observa también algunos usuarios como el administrador por defecto kadmin/admin@ICAI_TFM.

```
andresjimenez@deployments-cluster-terraform-m:~$ sudo kadmin.local
Authenticating as principal dataproc/admin@ICAI_TFM with password.
kadmin.local:  list_principals
HTTP/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
HTTP/deployments-cluster-terraform-w-0.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
HTTP/deployments-cluster-terraform-w-1.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
K/M@ICAI_TFM
dataproc/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
dataproc/deployments-cluster-terraform-w-0.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
dataproc/deployments-cluster-terraform-w-1.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
hdfs/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
hdfs/deployments-cluster-terraform-w-0.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
hdfs/deployments-cluster-terraform-w-1.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
hive/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
hive/deployments-cluster-terraform-w-0.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
hive/deployments-cluster-terraform-w-1.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
kadmin/admin@ICAI_TFM
kadmin/changepw@ICAI_TFM
kadmin/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
kiprop/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
krbtgt/ICAI_TFM@ICAI_TFM
mapred/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
mapred/deployments-cluster-terraform-w-0.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
mapred/deployments-cluster-terraform-w-1.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
root@ICAI_TFM
spark/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
yarn/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
yarn/deployments-cluster-terraform-w-0.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
yarn/deployments-cluster-terraform-w-1.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
```

Imagen 43. Lista principals

Este usuario es el único ahora mismo con el rol de administrador. Para crear otro usuario con este rol hemos de añadirlo.

Se aprecia, imagen 44, como se ha creado el usuario y ya aparece en la lista de *principals*. Para ello se usa el comnado addprinc junto con el nombre del usuario que se quiere crear y el REALM, en este caso además se ha añadido */admin* para crearlo como usuario administrador. Si se quisiera crear como usuario al uso, debería de sustituirse */admin* por */deployments-cluster*-Terraform-*m.europe-west4-a.c.artful-guru-312707.internal*

```
kadmin.local:  addprinc andresjimenez/admin
WARNING: no policy specified for andresjimenez/admin@ICAI_TFM; defaulting to no policy
Enter password for principal "andresjimenez/admin@ICAI_TFM":
Re-enter password for principal "andresjimenez/admin@ICAI_TFM":
Principal "andresjimenez/admin@ICAI_TFM" created.
kadmin.local:  list_principals
HTTP/deployments-cluster-terraform-m.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
HTTP/deployments-cluster-terraform-w-0.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
HTTP/deployments-cluster-terraform-w-1.europe-west4-a.c.artful-guru-312707.internal@ICAI_TFM
K/M@ICAI_TFM
andresjimenez/admin@ICAI_TFM
```

Imagen 44. Add principal

Una vez hecho esto se solicita un ticket y se comprueba que permite acceso a los servicios, por ejemplo, a HDFS, imagen 45.

Se puede observar como se hace el *kinit* del usuario andresjimenez/admin y se solicita la contraseña que se introdujo en el momento de la creación del mismo. Posteriormente con el comando *klist* se puede ver información sobre el ticket generado, como su fecha de creación, su fecha de expiración (8 horas) o hasta cuando se puede renovar (7 días).
Una vez hecho esto, se comprueba que al listar el directorio padre en HDFS no se reporta ningún error, pero no aparece nada pues está vacío

```
andresjimenez@deployments-cluster-terraform-m:~$ kinit andresjimenez/admin
Password for andresjimenez/admin@ICAI_TFM:
andresjimenez@deployments-cluster-terraform-m:~$ klist
Ticket cache: FILE:/tmp/krb5cc_1002
Default principal: andresjimenez/admin@ICAI_TFM

Valid starting     Expires            Service principal
06/17/21 10:39:09  06/17/21 18:39:05  krbtgt/ICAI_TFM@ICAI_TFM
        renew until 06/24/21 10:39:05
andresjimenez@deployments-cluster-terraform-m:~$ hdfs dfs -ls
andresjimenez@deployments-cluster-terraform-m:~$
```

Imagen 45. Kinit

Para asegurarnos creamos un fichero vacío y lo subimos a HDFS, comprobando que nos deja verlo, siguiente imagen.

```
andresjimenez@deployments-cluster-terraform-m:~$ touch ejemplo.txt
andresjimenez@deployments-cluster-terraform-m:~$ hdfs dfs -put ejemplo.txt .
andresjimenez@deployments-cluster-terraform-m:~$ hdfs dfs -ls
Found 1 items
-rw-r--r--   2 andresjimenez hadoop          0 2021-06-17 10:44 ejemplo.txt
andresjimenez@deployments-cluster-terraform-m:~$
```

Imagen 46. Acceso HFDS

Una vez comprobado todo esto, se puede asegurar que el despliegue de la infraestructura con la securización necesaria ha sido correcto, y por lo tanto se puede proceder al despliegue y configuración del software de validaciones.

3.2 DESPLIEGUE Y CONFIGURACIÓN DEL SOFTWARE MEDIANTE ANSIBLE

Tras haber desplegado el clúster y configurado cumpliendo todas las especificaciones requeridas y necesarias para su correcto funcionamiento, se procede a la codificación de las plantillas de Ansible requeridas para desplegar el producto de validaciones en el clúster. Ansible se basa en un funcionamiento estructurado en directorios en el que determinadas carpetas han de tener un nombre concreto para su correcta ejecución. Son varios los ficheros que intervienen en un proyecto con Ansible, y que serán explicados a continuación.

La estructura de directorio y ficheros es la siguiente:

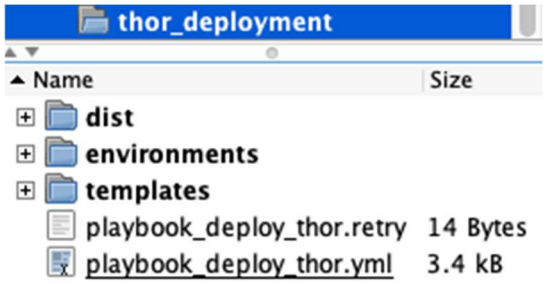


Imagen 47. Estructura drectorios Ansible

En primer lugar, la carpeta principal con el nombre que se desee, en nuestro caso *thor_deployments*. En esta carpeta se han de encontrar un varias subcarpetas necesarias:

- Dist: carpeta donde se encuentra al tar.gz de este producto que se va a depslegar.
- Environments: donde se encuentran las distintas configuraciones dependiendo del entorno donde se va a desplegar, en este caso solo GCP.
- Templates: donde se encuentran los ficheros que serán configurados on the fly mediante las variables del fichero de variables y variables de entorno y que son necesarios para la ejecución del producto
- Playbook: es el fichero principal donde se codifica la configuración y las tareas que se han de realizar.

Dentro de la carpeta *environments* es necesario explicar qué ficheros y qué estructura requiere, imagen 48 y 49 y 50.

Como se aprecia, dentro de la carpeta environments se encuentra una subcarpeta que hará referencia al entorno donde se va a desplegar, en nuestro caso, *deploy_gcloud*. Dentro de esta se encuentra una estructura obligatoria, encontrando el fichero host (imagen 49) y la subcarpeta group_vars, donde dentro encontramos otra carpeta llamada all y dentro de esta el fichero de variables llamado *environments_vars.yml*. Esta estructura, es obligatoria para el funcionamiento de la herramienta.

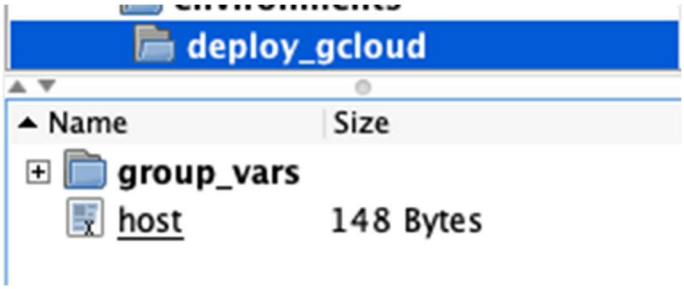


Imagen 48. Estructura environments parte 1

En este fihcero host encontramos dos etiquetas, local y dataproc, que serán llamadas desde el fichero playbook cuando se quiera hacer referencia a la máquina o servidor donde se quiere realizar las tareas de ese bloque. En este caso se configura que las tareas que se requieran en la etiqueta local se van a realizar en local, es decir en la máquina desde la que se va a lanzar Ansible, y que en la etiqueta dataproc, la máquina en la que tienen que realizarse esas tareas tiene la ip 34.90.222.23, el usuario por el que va a conectarse a través de ssh es andresjimenez y la cave privada que ha de usar se encuentra en el path ejemplificado.

```
[local]
127.0.0.1

[dataproc]
34.90.222.23 ansible_user=andresjimenez ansible_ssh_private_key_file=/Users/andresjimenez/.ssh/google_compute_engine
```

Imagen 49. Fichero host

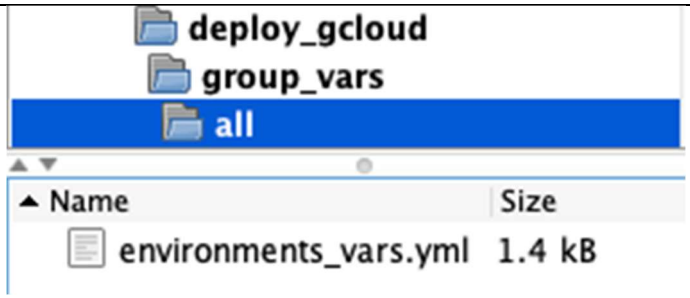


Imagen 50. Estructura environments parte 2

En cuanto a la carpeta template, dentro encontramos un fichero de configuración necesario para la ejecución del software y que será configurado on the fly.
A continuación, se procede a explicar los tres ficheros necesarios, una vez ha sido explicada toda la estructura de directorios y el host. Estos ficheros son, el playbook, el fichero de variables, y el fichero de configuración concreto de este producto

3.2.1 FICHERO DE VARIABLES

En este fichero es donde se van a definir todas las variables necesarias en los demás ficheros. Consta de 4 partes en las que se puede dividir su explicación.
La primera parte, imagen 51, es donde se definen los nombres de las carpetas, directorios, etc. que se quieren usar. Se procede a comentar aquella de relevancia.
En primer lugar, se define el nombre de la Operational Business donde va dirigido el despliegue, en este caso ninguna en concreto pues es un proyecto personal, así que se le llama tfm. Posteriormente se define el nombre del entorno, haciendo referencia a la variable ob creada, el nombre del proyecto y su versión, así como el path del entorno virtual Python que será creado por si fuera necesario su uso. Además, se definen 3 directorios más que serán usado durante la configuración. Cuando se llamar referencia a una variable de entorno como puede ser HOME, se hace mediante el comando Ansible_*env.*NombreVariable, teniendo configurada a *true* la variable *gather_facts* en el bloque del playbook donde se llame a la variable de entorno.

```
# OB Environment Name
ob: tfm

# Workspace path
workspace: workspace-test

# Working directories
environment_name: "{{ ob }}"
project_name: thor_deployment
version: latest
virtualenv_ansible: "{{ ansible_env.HOME }}/.virtualenv/{{ project_name }}-ansible"
working_dir: "{{ ansible_env.HOME }}/{{ project_name }}-{{ version }}"
workspace_setup: "{{ working_dir }}/{{ workspace }}/setup"
temporal_local_package_path: "/tmp/{{ project_name }}-{{ version }}.tar.gz"
```

Imagen 51. Fichero variables parte 1

A continuación, imagen 52, se procede con el bloque referido al entorno virtual Python, que se creará por curiosidad académica ya que, en este proyecto concreto, no es necesario Python pues el producto corre en spark scala, pero son muchos los productos que necesitan de Python para su ejecución por lo que se considera interesante su creación ya que no interfiere en el correcto funcionamiento del mismo.

En primer lugar se define cual se quiere que sea el path donde se aloje el binario del entorno, en segundo lugar qué Python se quiere usar en los nodos workers, en este caso el Python que esté en local en esos workers, en tercer lugar qué Python se requiere en el nodo driver que en este caso es el del entorno virtual que creamos, aunque esto no es obligatorio pues si no se configura cogerá el Python por defecto (en esta ocasión se hace así) y por último se configura el path de determinadas librerías internas del sistema de Hadoop y de Python.

```
# Python environment
path: "/opt/conda/miniconda3/bin/"
pyspark_python: "/opt/conda/miniconda3/bin/python3"
pyspark_driver_python: "{{ virtualenv_ansible }}/bin/python"
ld_library_path: "/usr/lib64/:/usr/lib/hadoop/lib/native/"  #/usr/lib/hadoop/lib/native/
#son librerias internas del sistema que utiliza python y de hadoop que tambien python usa
```

Imagen 52. Fichero variables parte 2

Posteriormente se definen distintas rutas necesarias para el entorno Hadoop, Yarn, y Spark.

Estas rutas son donde se van a encontrar los ficheros de configuración necesarios para estas herramientas. Las dos últimas, son dependencias necesarias en este producto, que han de ser descargadas. La primera para leer los ficheros avro y la segunda se trata de una dependencia interna de telefónica para leer datos de la 4º plataforma.

```
#Spark

HADOOP_CONF_DIR: "/etc/hadoop/conf"
YARN_CONF_DIR: "/etc/hadoop/conf"
SPARK_HOME: "/usr/lib/spark"
SPARK_AVRO_VERSION: "org.apache.spark:spark-avro_2.12:2.4.7" #dependencias para leer avros
SPARK_SDK_TELEFONICA: "com.telefonica.baikal:spark-sdk_2.12:0.5.0-BETA7" #dependencia interna de telefonica para leer datos de la 4plat
```

Imagen 53. Fichero variables parte 3

En último lugar, se definen configuraciones necesarias para el fichero .sh que es el que se va a lanzar y el encargado de ejecutar el spark-submit con la configuración requerida.
Se configura cual va a ser el modo de ejecución, en este caso modo *client* por lo que se va a elegir nodo master aquel desde el que se lance la ejecución.
Por último, se configuran las RAM y los executors, esta configuración viene predefinida por los desarrolladores del producto como configuración mínima.

```
# Thor run script configuration
DEPLOY_MODE: "client"  #orquestados yarn
DRIVER_MEMORY: "2G"
DRIVER_CORES: 3
NUM_EXECUTORS: 5
EXECUTOR_MEMORY: "10G"
EXECUTOR_CORES: 3
QUEUE: "default"
```

Imagen 54. Fichero variables parte 4

3.2.2 FICHERO PLAYBOOK

El fichero playbook es el fichero principal que va a ser ejecutado por Ansible. En este se definen los bloques, y dentro de estos las tareas a realizar y las llamadas a las variables definidas en este fichero.

En este playbook encontramos dos grandes bloques, el bloque de descarga o copia del paquete, y el bloque de despliegue. Se comienza con el bloque de descarga o copia.

En la siguiente imagen, 55, se aprecia la primera parte de este fichero, un bloque llamado *download package*. En este bloque se establece que ha de ser realizado en local, en la ip definida en el fichero host, y que no se van a coger variables de entorno, por lo que *gather_facts* está a false. En este sub-bloque se requiere un manejador de excepciones ya que, si falla la búsqueda del paquete en local, va a buscarlo al repositorio de Telefónica.

En la instrucción *copy*, se hace una búsqueda de un fichero cuyo nombre ha de ser el definido en la variable project_name-version.tar.gz, si lo encuentra lo copia a un directorio temporal definido en el fichero variables. Si esto falla se ejecuta el siguiente sub-bloque donde se va a la url definida a descargarlo.

```
- name: "Download package"
  hosts: "localhost"  #llama a host
  connection: local  #porque vamos a hacer este bloque en local
  tags: always
  gather_facts: false  #no coge variables de entorno
  tasks:  #los bloques de tareas que vamos a hacer
  - name: BLOCK - Download and prepare package
    block:
    - name: Using local package if exists
      copy:
        src: "{{ lookup('pipe', 'ls -1 dist/{{ project_name }}-{{ version }}.tar.gz') }}"
        dest: "{{ temporal_local_package_path }}"
    rescue:
    - debug: msg='Using local file has failed. Try to download package from artifactory.'
    - name: Download {{ project_name }}-{{ version }}.tar.gz
      get_url:
        url: http://artifactory.hi.inet/artifactory/yum-analytics/{{ project_name }}/dist/x86_64/{{ project_name }}-{{ version }}.tar.gz
        dest: "{{ temporal_local_package_path }}"
    tags: download_package
```

Imagen 55. Playbook parte 1

Cuando se ha realizado este bloque, se pasa al siguiente, donde se procede a desplegar en la máquina destino el paquete alojado en el repositorio temporal.

Este bloque que continua está dividido en tres sub-bloques: los dos primeros dedicados a la creación de directorios y configuración del paquete y el último a instalar las dependencias necesarias.

En primer lugar, imagen 56, se define cual va a ser la máquina sobre la que se van a realizar estas tareas, en este caso la definida bajo la etiqueta "dataproc" en el fichero host. Posteriormente se crea el directorio donde se va a descomprimir el paquete y le indicamos que no cree la carpeta comprimida ya que lo vamos a descomprimir en la ya creada.

```yaml
- name: "Deploy Package"
  hosts: "dataproc"
  gather_facts: true
  tasks:
  - name: BLOCK - Check delegation
    block:
    - name: BLOCK - Install Package
      tags: install_package
      block:
      - name: Create working directory - OS
        file:
          state: directory
          path: "{{ working_dir }}"

      - name: Unarchive package
        unarchive:
          src: "{{ temporal_local_package_path }}"
          dest: "{{ working_dir }}"
          extra_opts: [--strip-components=1] #en la carpeta destino no crees la carpeta comprimida
```

Imagen 56. Playbook parte 2.1

Una vez descomprimido el paquete en la carpeta creada en *working_dir,* variable definida en el fichero de variables como "{{ Ansible_env.HOME }}/{{ project_name }}-{{ version }}", se pasa al siguiente sub bloque.

Una vez hecho esto, imagen 57, se crea un directorio con el nombre definido en la variable *workspace_setup* formado del siguiente modo: "{{ working_dir }}/{{ workspace }}/setup"

Por último, copiamos el fichero run_thor.sh (explicado en el siguiente punto) configurado *on the fly* en la carpeta template en la ruta definida como destino y con permiso de escritura únicamente para el usuario propietario, y de lectura y ejecución para todos.

```
- name: BLOCK - Configure Package
  tags: configure_package
  block:
  - name: Create configuration directory {{ workspace_setup }}
    file:
      path: "{{ workspace_setup }}"
      state: directory

  - name: Configure project run script
    template:
      src: templates/run_thor.sh.template
      dest: "{{ working_dir }}/run_thor.sh"
      mode: 0755
```

Imagen 57. Playbook parte 2.2

En último lugar, imagen 58, se procede a la configuración del entorno virtual.
En este bloque se instalan las dependencias necesarias y al mismo tiempo que se crea el entorno virtual donde van a ser instaladas, definido en la variable *virtualenv*_Ansible. Se define también cual va a ser el Python que se quiere en este entorno y se añade el path destino el path del entorno creado. Por último, se definen las librerías del sistema necesarias.

```
- name: BLOCK - Install requirements
  tags: install_requirements
  block:
  - name: Update pip
    pip:
      name:
      - pip==20.2.4
      state: latest
      virtualenv: "{{ virtualenv_ansible }}"
      virtualenv_python: "{{ pyspark_python }}"
    environment:
      PATH: "{{ ansible_env.PATH }}:{{ path }}"
      LD_LIBRARY_PATH: "{{ ld_library_path }}"
```

Imagen 58. Playbook parte 2.3

3.2.3 FICHERO RUN_THOR.SH

Este fichero, es el que se va a lanzar cuando se quiera ejecutar el software, por lo que ha de ser configurado con las variables necesarias para ello. Estas variables han sido ya explicadas y definidas en el fichero variables.

De este modo, cuando se quiera realizar alguna modificación en un despliegue del producto en otro entorno, habrá que modificar el fichero variables, quedando este siempre intacto.

En primer lugar, se almacenan variables que se pasan como parámetro en la llamada al fichero, y que serán usadas posteriormente en la configuración del spark-submit.

A continuación, se configuran las variables de memoria, cores y executors definidas en el fichero de variables, y por último se exportan como variables de entorno las configuraciones necesarias de spark, yarn y hadoop para que puedan ser encontradas al lanzarse el producto.

```
SPARK_MASTER=$1
DATASET_ID=$2
DATASET_VERSION=$3
DATASET_MAJOR_VERSION=$(echo ${DATASET_VERSION} | cut -d'.' -f 1)
FILTER_DATE=$4

DEPLOY_MODE={{ DEPLOY_MODE }}
DRIVER_MEMORY={{ DRIVER_MEMORY }}
DRIVER_CORES={{ DRIVER_CORES }}
NUM_EXECUTORS={{ NUM_EXECUTORS }}
EXECUTOR_MEMORY={{ EXECUTOR_MEMORY }}
EXECUTOR_CORES={{ EXECUTOR_CORES }}
QUEUE={{ QUEUE }}


export HADOOP_CONF_DIR={{ HADOOP_CONF_DIR }}
export YARN_CONF_DIR={{ YARN_CONF_DIR }}
export SPARK_HOME={{ SPARK_HOME }}
export SPARK_AVRO_VERSION={{ SPARK_AVRO_VERSION }}
export SPARK_SDK_TELEFONICA={{ SPARK_SDK_TELEFONICA }}
```

Imagen 59. Thor.sh parte 1

Una vez configurado esto se procede a explicar el spark-submit a través del cual se ejecutará el software, siguiente imagen.

En esta configuración se llama a las variables arriba definidas de cara a la configuración de memoria, cores etc. y se configura el fichero JSON utilizado como input donde va a encontrarse la información de la data a validar, así como los esquemas de los ficheros avro, y el fichero de salida donde se va a volcar el resultado. También se configura el modo de ejecución, client, y el jar necesario para la lectura de ficheros avro.

```
SW_HOME=/opt/thor4p-external

$SPARK_HOME/bin/spark-submit \
  --deploy-mode ${DEPLOY_MODE} \
  --master ${SPARK_MASTER} \
  --driver-memory ${DRIVER_MEMORY} \
  --driver-cores ${DRIVER_CORES} \
  --num-executors ${NUM_EXECUTORS} \
  --executor-memory ${EXECUTOR_MEMORY} \
  --executor-cores ${EXECUTOR_CORES} \
  --queue ${QUEUE} \
  --packages ${SPARK_AVRO_VERSION} \
  --class com.telefonica.baikal.thor.Main ${SW_HOME}/thor1.1.jar \
  --dataset-id=${DATASET_ID} \
  --dataset-version=${DATASET_MAJOR_VERSION} \
  --filter-date=${FILTER_DATE} \
  --local-mode \
  --config-file=${SW_HOME}/utils/${DATASET_ID}/v${DATASET_VERSION}/json/local-config-file-${DATASET_ID}_${DATASET_VERSION}.json \
  --output-result-path=/user/data_avro/thor_results/v2.0.0/data/${DATASET_ID}/${DATASET_MAJOR_VERSION}/${FILTER_DATE}/
```

Imagen 60. Thor.sh parte 2

3.2.4 DESPLIEGUE

Una vez configurados todos estos ficheros, solo queda ejecutar la orden necesaria para este realizar este despliegue.

Para ello se ejecuta el siguiente comando, desde dentro de la carpeta principal *thor_deployments* y desde donde se tiene acceso al fichero *playbook*.

Se puede ver, imagen 61 y 62 como se van ejecutando de manera secuencial los bloques y dentro de estos los sub-bloques definidos. En el momento en el que se realiza la conexión ssh a la máquina destino, se pedirá la contraseña de la clave privada que se usa en la conexión.

Imagen 61. Ejecución parte 1

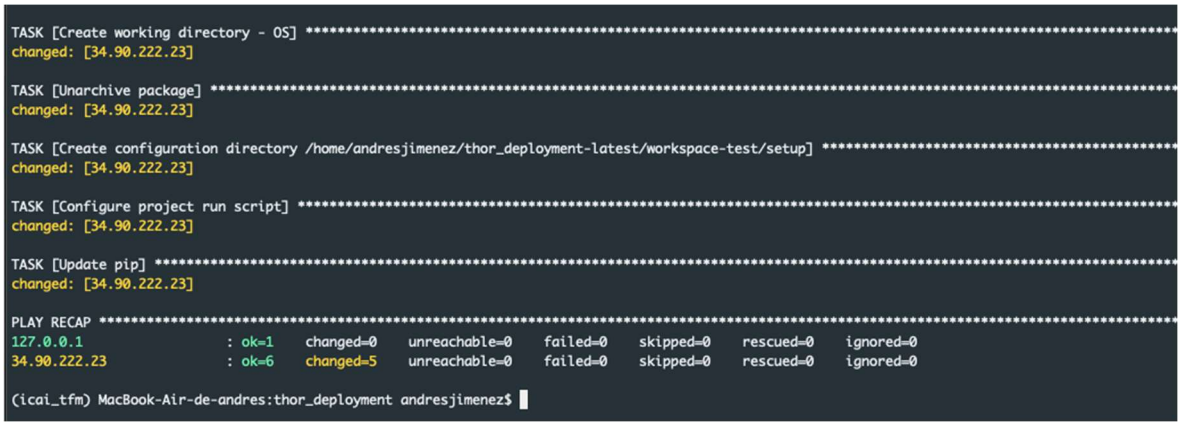


Imagen 62. Ejecución parte 2

Para comprobar que el despliegue se ha realizado correctamente, al realizar una conexión al nodo master del clúster, debería de haberse desplegado toda esta estructura, siguiente imagen.

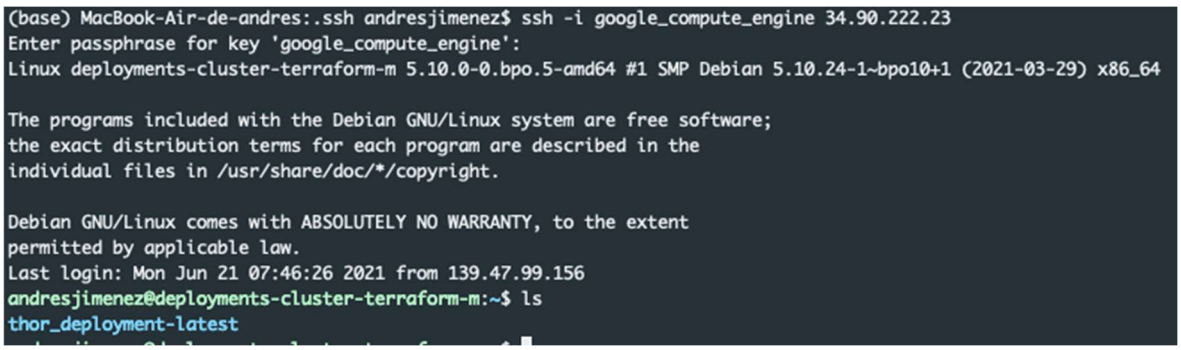


Imagen 63. Comprobación thor en destino

Capítulo 4. EJECUCIÓN Y RESULTADOS

Una vez el producto ha sido desplegado en destino, el último paso es ejecutar el software sobre el conjunto de datos que se quiere validar. En este caso, y a modo de prueba, se va a comprobar un conjunto parcial de datos de una entidad determinada.

Para esto, se ha configurado previamente los ficheros del producto que apuntan a la data y a los esquemas alojados en el *bucket icai_tfm _deployments* del que ya se ha hablado durante el despliegue de la infraestructura.

En primer lugar, para la ejecución es necesario solicitar un TGT a kerberos, ya que es necesario autenticarse contra los servicios de spark, yarn y hdfs que van a ser usados por el software. Para ellos ejecutamos la orden *kinit* sobre el usuario andresjimenez/admin y una vez autenticado procedemos a ejecutar el script run_thor.sh encargado de lanzar el spark-submit con la configuración correspondiente, siguiente imagen.

Para su ejecución se han de pasar como parámetros la entidad el orquestador que se quiere utilizar, en este caso yarn, la entidad a validar, llamada Fixed_Line, junto a su versión y la fecha concreta que se quiere validar.

Durante su ejecución, imagen 65, vamos obteniendo distintas salidas, según las querys que están definidas para cada validación.

```
andresjimenez@deployments-cluster-terraform-m:~/thor_deployment-latest$ kinit andresjimenez/admin
Password for andresjimenez/admin@ICAI_TFM:
andresjimenez@deployments-cluster-terraform-m:~/thor_deployment-latest$ sh run_thor.sh yarn Fixed_Line 5.11.0 2020-10-01
```

Imagen 64. Ejecución thor

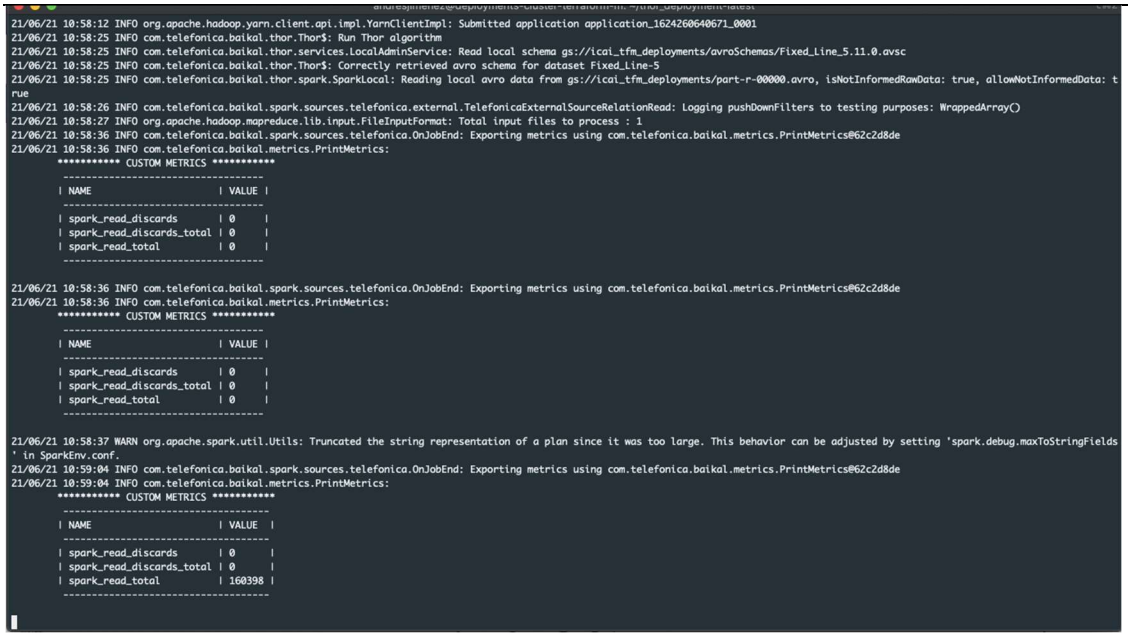


Imagen 65. Salida thor

Una vez finalizada la ejecución, se han almacenado en hdfs todos los resultados de esta, en ficheros avro, véase siguiente imagen.

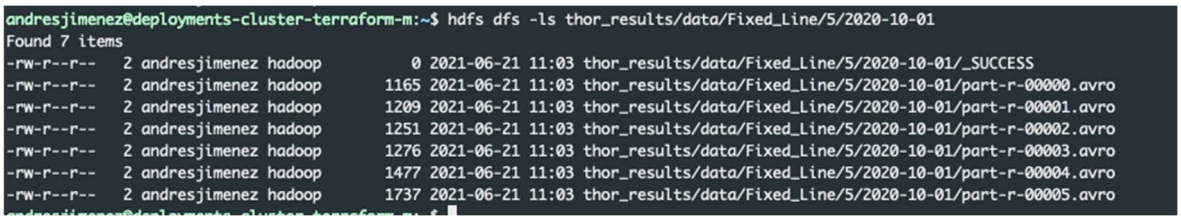


Imagen 66. Resultados thor hdfs

Estos resultados son una pieza clave para el equipo de URM encargado de analizarlos con el objetivo de comprobar y garantizar que los datos utilizados por los productos de la compañía son correctos.

Capítulo 5. POSIBLES TRABAJOS FUTUROS

Una vez concluido este proyecto, y tras estudiar sus resultados y proponer algunas posibles soluciones que suelen ser las tomadas en este tipo de problemáticas, se quiere explicar de manera breve qué otros pasos podrían llevarse a cabo a partir de los datos y resultados obtenidos en este TFM.

Se cree que sería de gran interés realizar este mismo despliegue haciendo uso de otros proveedores, como Azure o AWS y de otras herramientas de despliegue de las citadas y explicadas en el Capítulo 2. De esta manera podría realizarse una comparativa que permitiese el estudio de rendimientos, recursos y rentabilidades de cada una de ellas y permitiese así llegar a una decisión que optimizase el rendimiento y el gasto empresarial en los distintos proyectos.

Capítulo 6. LIMITACIONES Y COSTES

En cuanto a las limitaciones con las que se ha encontrado este proyecto cabe citar la dificultad de encontrar información sobre estas tecnologías, debido a que no son muchas las compañías y los proyectos donde estas se usan. Cada día más y más compañías buscan implementarlas facilitando y agilizando así sus despliegues, pero aún escasea la información en comparación con otros lenguajes o tecnologías.
Por otro lado, dado que la cuenta del proveedor cloud utilizada era personal y no empresarial, se han sufrido algunas limitaciones en proveedores como Azure, que no han permitido realizar el despliegue en este proveedor como se quería.

Por otro lado, algo a destacar en este tipo de proyectos son los costes. En este caso se ha usado una cuenta *free trial* con 300$. Los costes incurridos durante todo el proyecto, aproximadamente dos meses, donde se levantaba y destruía la infraestructura en distintas pruebas, y donde se desplegaba y ejecutaba el producto, han sido de 152,77$. En mi opinión no han sido unos costes excesivamente altos, esto es debido a que las ejecuciones realizadas han sido que una data reducida pudiendo así hacer uso de instancias humildes en cores y RAM, por lo tanto, más baratas. En la siguiente imagen se aprecia el precio por hora y cores y por hora y RAM de la máquina N1 utilizada y ubicada en el oeste de Europa.

Tipos de máquinas N1

Países Bajos (europe-west4) | Por mes | Por hora

Elemento	Precio según demanda (USD)	Precio de recurso interrumpible (USD)	Precio con compromiso de 1 año (USD)	Precio con compromiso de 3 años (USD)
CPU virtuales predefinidas	$0.0347721 / vCPU hour	$0.007325 / vCPU hour	$0.021925 / vCPU hour	$0.015661 / vCPU hour
Memoria predefinida	$0.0046607 / GB hour	$0.000987 / GB hour	$0.002938 / GB hour	$0.002099 / GB hour

Imagen 67. Costes instancias. Fuente:cloud.Google.com

Capítulo 7. BIBLIOGRAFÍA

Abril Nuñez, E. (2016). *Automatizar tareas con* Ansible*, Chef y Puppet*. https://openwebinars.net/blog/automatizar-tareas-con-Ansible-y-puppet/

Ahmed, H. (2021). *Introduction to Cloud Computing*. Introduction to Cloud Computing. https://www.c-sharpcorner.com/UploadFile/834980/introduction-to-cloud-computing/

Ansible. (2018). *Continuous Integration and Delivery with* Ansible. 1–6. https://www.Ansible.com/hubfs/pdfs/ContinuousDelivery-with-Ansible-WhitePaper.pdf

Bigelow, S. J. (2020). *Microsoft Azure*. https://searchcloudcomputing.techtarget.com/definition/Windows-Azure

Brikman, Y. (2016). *Why we use* Terraform *and not Chef, Puppet,* Ansible*, SaltStack, or CloudFormation*. https://blog.gruntwork.io/why-we-use-Terraform-and-not-chef-puppet-Ansible-saltstack-or-cloudformation-7989dad2865c

Chand, M. (2021). *Top 10 Cloud Service Providers In 2021*. https://www.c-sharpcorner.com/article/top-10-cloud-service-providers/

Hadoop. (n.d.). *Hadoop Guide*. Retrieved June 16, 2021, from https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-common/yarn-default.xml

IONOS. (n.d.). *Infrastructure as code (IaC): gestionar la infraestructura de TI como código*. Retrieved June 10, 2021, from https://www.ionos.es/digitalguide/servidores/know-how/infrastructure-as-code/

MIT. (n.d.). *Red Hat Enterprise Linux 4: Manual de referencia*. Retrieved June 19, 2021, from https://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-kerberos.html

Red Hat. (2021). *What is Infrastructure as Code (IaC)?* https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac

Services Amazon Web. (n.d.). *What is AWS*. Retrieved June 10, 2021, from https://aws.amazon.com/es/what-is-aws/

Spark. (n.d.). *Spark Configuration*. Retrieved June 16, 2021, from https://spark.apache.org/docs/latest/configuration.html

Wikipedia. (n.d.). Ansible. Retrieved June 14, 2021, from https://es.wikipedia.org/wiki/Ansible_(software)

Wikipedia. (2021). *Kerberos*. https://es.wikipedia.org/wiki/Kerberos

ANEXO A

Terraform

variables.tf

```
[1]   variable "project_id" {
[2]     description = "TFM_ICAI (GCP) Project."
[3]     type        = string
[4]     default     = "artful-guru-312707"
[5]   }
[6]
[7]   variable "region" {
[8]     description = "GCP region name."
[9]     type        = string
[10]    default     = "europe-west4"
[11]  }
[12]
[13]  variable "zone" {
[14]    description = "GCP zone name."
[15]    type        = string
[16]    default     = "europe-west4-a"
[17]  }
[18]
[19]
[20]  ###Almacenar los logs del cluster y como un share warehosuse para workers y
      master
[21]  variable "staging_bucket" {
[22]    description = "Cluster staging bucket ."
[23]    type        = string
[24]    default     = "icai_tfm_deployments"
[25]  }
```

```
[26]
[27]
[28]    # Cluster Variables
[29]    variable "dataproc_cluster" {
[30]      description = "Dataproc Cluster Properties"
[31]      type        =  object(
[32]        {
[33]          name                    = string
[34]          preemptible_num_instances = number
[35]          image_version           = string
[36]        }
[37]      )
[38]      default     = {
[39]        name                    = "deployments-cluster-terraform"
[40]        preemptible_num_instances = 0
[41]        image_version           = "1.5"
[42]      }
[43]    }
[44]
[45]
[46]    variable "dataproc_network" {
[47]      description = "Dataproc Netoworks Properties"
[48]      type        =  object(
[49]        {
[50]          name                    = string
[51]          mtu                     = number
[52]          routing_mode            = string
[53]        }
[54]      )
[55]      default     = {
[56]        name                    = "network-icai"
```

```
[57]      mtu                   = 1460
[58]      routing_mode          = "REGIONAL"
[59]    }
[60]  }
[61]
[62]  variable "dataproc_subnetwork" {
[63]    description = "Dataproc Netoworks Properties"
[64]    type        =  object(
[65]      {
[66]        name                = string
[67]        ip_range            = string
[68]      }
[69]    )
[70]    default     = {
[71]      name                  = "subnetwork-icai"
[72]      ip_range              = "10.20.0.0/16"
[73]    }
[74]  }
[75]
[76]
[77]  variable "override_properties" {
[78]    description = "Cluster Properties"
[79]    type        = map
[80]    default     = {
[81]      "dataproc:yarn.log-aggregation.enabled" = "true",
[82]      "yarn:yarn.resourcemanager.am.max-attempts" = 10,
[83]      "spark:spark.task.maxFailures"=10,
[84]      "spark:spark.stage.maxConsecutiveAttempts" = 10
[85]    }
[86]  }
[87]
```

```
[88]  variable "master_node" {
[89]    description = "GCP Worker VM instance machine type."
[90]    type        = object(
[91]      {
[92]        machine_type      = string
[93]        num_instances     = number
[94]        boot_disk_size_gb = number
[95]      }
[96]    )
[97]    default     = {
[98]      machine_type      = "n1-standard-4"
[99]      num_instances     = 1
[100]     boot_disk_size_gb = 35
[101]   }
[102] }
[103]
[104] variable "worker_node" {
[105]   description = "GCP Worker VM instance machine type."
[106]   type        = object(
[107]     {
[108]       machine_type      = string
[109]       num_instances     = number
[110]       boot_disk_size_gb = number
[111]     }
[112]   )
[113]   default     = {
[114]     machine_type      = "n1-standard-4"
[115]     num_instances     = 2
[116]     boot_disk_size_gb = 35
[117]   }
[118] }
```

```
[119]
[120]  variable "tags" {
[121]    description = "List of tags instance."
[122]    type        = list
[123]    default     = ["dataproc"]
[124]  }
```

cluster.tf

```
[1]    variable "project_id" {}
[2]    variable "region" {}
[3]    variable "zone" {}
[4]    variable "tags" {}
[5]    variable "dataproc_cluster" {}
[6]    variable "master_node" {}
[7]    variable "worker_node" {}
[8]    variable "staging_bucket" {}
[9]    #variable "operational_user" {}
[10]   variable "override_properties" {}
[11]   variable "dataproc_subnetwork" {}
[12]
[13]
[14]   # Create a Google dataproc cluster
[15]   resource "Google_dataproc_cluster" "deployments_cluster" {
[16]     project = var.project_id
[17]     name   = var.dataproc_cluster["name"]
[18]     region = var.region
[19]
[20]     cluster_config {
[21]       staging_bucket = var.staging_bucket
[22]
[23]       master_config {
```

```
[24]        num_instances = var.master_node["num_instances"]
[25]        machine_type  = var.master_node["machine_type"]
[26]        disk_config {
[27]          boot_disk_size_gb = var.master_node["boot_disk_size_gb"]
[28]        }
[29]      }
[30]
[31]      worker_config {
[32]        num_instances    = var.worker_node["num_instances"]
[33]        machine_type     = var.worker_node["machine_type"]
[34]        disk_config {
[35]          boot_disk_size_gb = var.worker_node["boot_disk_size_gb"]
[36]        }
[37]      }
[38]
[39]      preemptible_worker_config {
[40]        num_instances = var.dataproc_cluster["preemptible_num_instances"]
[41]      }
[42]
[43]      software_config {
[44]        image_version       = var.dataproc_cluster["image_version"]
[45]        override_properties = var.override_properties
[46]      }
[47]
[48]      gce_cluster_config {
[49]        tags                = var.tags
[50]        zone                 = var.zone
[51]        subnetwork                          = var.dataproc_subnetwork["name"]
[52]      }
[53]
[54]      security_config {
```

```
[55]        kerberos_config {
[56]          kms_key_uri = "projects/artful-guru-
        312707/locations/global/keyRings/icai_kms/cryptoKeys/icai_kms_kerberos"
[57]          root_principal_password_uri = "gs://icai_tfm_deployments/kerberos-root-
        principal-password.encrypted"
[58]          enable_kerberos = "true"
[59]          tgt_lifetime_hours = 8
[60]          realm = "ICAI_TFM"
[61]        }
[62]      }
[63]
[64]    }
[65]  }
```

networks.tf

```
[1]
[2]   variable "region" {}
[3]   variable "dataproc_subnetwork" {}
[4]   variable "dataproc_network" {}
[5]
[6]   resource "Google_compute_network" "network_cluster" {
[7]     name                  = var.dataproc_network["name"]
[8]     mtu                   = var.dataproc_network["mtu"]
[9]     auto_create_subnetworks = false
[10]    routing_mode          = var.dataproc_network["routing_mode"]
[11]  }
[12]
[13]
[14]  resource "Google_compute_subnetwork" "subnetwork_cluster" {
[15]    name          = var.dataproc_subnetwork["name"]
[16]    ip_cidr_range = var.dataproc_subnetwork["ip_range"]
```

```
[17]      region       = var.region
[18]      network      = Google_compute_network.network_cluster.name
[19]      secondary_ip_range {
[20]        range_name    = "tf-test-secondary-range-update1"
[21]        ip_cidr_range = "192.168.10.0/24"
[22]      }
[23]    }
[24]
[25]    resource "Google_compute_firewall" "firewall_network_external" {
[26]      name    = "external"
[27]      network = Google_compute_network.network_cluster.name
[28]      source_ranges = ["139.47.99.XXX/32"]
[29]
[30]      allow {
[31]        protocol = "tcp"
[32]        ports = ["22"]
[33]
[34]      }
[35]
[36]    }
[37]
[38]    resource "Google_compute_firewall" "firewall_network_internal" {
[39]      name    = "internal"
[40]      network = Google_compute_network.network_cluster.name
[41]      source_ranges = [var.dataproc_subnetwork["ip_range"]]
[42]
[43]      allow {
[44]        protocol = "all"
[45]
[46]      }
[47]
```

```
}
```

main.tf

```
[1]
[2]     provider "Google" {
[3]       version = "3.57"
[4]       project = var.project_id
[5]       region  = var.region
[6]       zone    = var.zone
[7]     }
[8]
[9]     module "network" {
[10]      source                  = "./network"
[11]      region                  = var.region
[12]      dataproc_network            = var.dataproc_network
[13]      dataproc_subnetwork           = var.dataproc_subnetwork
[14]    }
[15]
[16]    # Create Cluster module
[17]    module "cluster" {
[18]      source                  = "./cluster"
[19]      project_id                = var.project_id
[20]      region                  = var.region
[21]      zone                    = var.zone
[22]      tags                    = var.tags
[23]      dataproc_cluster            = var.dataproc_cluster
[24]      master_node               = var.master_node
[25]      worker_node               = var.worker_node
[26]      staging_bucket              = var.staging_bucket
[27]      override_properties           = var.override_properties
[28]      dataproc_subnetwork           = var.dataproc_subnetwork
```

```
[29]    depends_on = [module.network]
[30]  }
[31]
```

Ansible

host

```
[1]   [local]
[2]   127.0.0.1
[3]
[4]   [dataproc]
 34.90.222.XXX ansible_user=andresjimenez
ansible_ssh_private_key_file=/Users/andresjimenez/.ssh/Google_compute_engine
```

environments_vars.yml

```
[1]   ##########################################################################
      ########
[2]   # Spain prepro variables with new URM data to replace in template files
[3]   ##########################################################################
      ########
[4]
[5]   # OB Environment Name
[6]   ob: tfm
[7]
[8]   # Workspace path
[9]   workspace: workspace-test
[10]
[11]  # Working directories
[12]  environment_name: "{{ ob }}"
[13]  project_name: thor_deployment
```

```
version: latest
virtualenv_ansible: "{{ ansible_env.HOME }}/.virtualenv/{{ project_name }}-ansible"
working_dir: "{{ ansible_env.HOME }}/{{ project_name }}-{{ version }}"
workspace_setup: "{{ working_dir }}/{{ workspace }}/setup"
temporal_local_package_path: "/tmp/{{ project_name }}-{{ version }}.tar.gz"

# Python environment
path: "/opt/conda/miniconda3/bin/"
pyspark_python: "/opt/conda/miniconda3/bin/python3"
pyspark_driver_python: "{{ virtualenv_ansible }}/bin/python"
ld_library_path: "/usr/lib64/:/usr/lib/hadoop/lib/native/" #/usr/lib/hadoop/lib/native/
#son librerias internas del sistema que utiliza python y de hadoop que tambien python usa

#Spark

HADOOP_CONF_DIR: "/etc/hadoop/conf"
YARN_CONF_DIR: "/etc/hadoop/conf"
SPARK_HOME: "/usr/lib/spark"
SPARK_AVRO_VERSION: "org.apache.spark:spark-avro_2.12:2.4.7" #dependencias para leer avros
SPARK_SDK_TELEFONICA: "com.telefonica.baikal:spark-sdk_2.12:0.5.0-BETA7" #dependencia interna de telefonica para leer datos de la 4plat


# Thor run script configuration
DEPLOY_MODE: "client"  #orquestados yarn y dentro usamos ejecucion client, modo cluster yarn coge un nodo master aleatorio y en client coge justo el nodo desde el que tu lanzas
```

```
[38]    DRIVER_MEMORY: "2G"
[39]    DRIVER_CORES: 3
[40]    NUM_EXECUTORS: 5
[41]    EXECUTOR_MEMORY: "10G"
[42]    EXECUTOR_CORES: 3
QUEUE: "default"
```

playbook_deploy_thor.yml

```
[1]     ---
[2]
[3]     ###
[4]     ## Run the playbook:
[5]     #
[6]     # $ ansible-playbook -i environments/ENVIRONMENT_NAME
        playbook_deploy_devrec.yml --extra-vars "version=DEVREC_VERSION" -u
        USER
[7]     #
[8]     # EXAMPLE (Use local package):
[9]     #  $ ansible-playbook -i environments/argentina_prepro
        playbook_deploy_devrec.yml --extra-vars "version=2.0" -u davidl
[10]    # EXAMPLE (download package from Artifactory):
[11]    #  $ ansible-playbook -i environments/argentina_prepro
        playbook_deploy_devrec.yml --extra-vars "version=2.0-2053" -u davidl
[12]    #
[13]    # -i: Relative path of environments vars
[14]    #   * ENVIRONMENT_NAME: specifies the identifier of the OB environment
[15]    # -e or --extra-vars: Extra vars:
[16]    #   * version: version to install the software.
[17]    #       Use a local file by default and if it does not exist try to download it from
        artifactory.
```

```
[18]  #       If you want to use a version of Artifactory you have to specify the version
      with the commit number. Example: 2.0-2048
[19]  #   * deploy_cron: boolean to specify whether cron is activated or not
[20]  # -u: user to connect by SSH
[21]  # --ask-pass: OPTIONAL - request password for the user to connect by ssh
[22]  # -t: OPTIONAL - list of tags separated by commas
[23]  ###
[24]
[25]  - name: "Download package"
[26]    hosts: "localhost"  #llama a host
[27]    connection: local  #porque vamos a hacer este bloque en local
[28]    tags: always
[29]    gather_facts: false  #no coge variables de entorno
[30]    tasks:  #los bloques de tareas que vamos a hacer
[31]    - name: BLOCK - Download and prepare package
[32]      block:
[33]      - name: Using local package if exists
[34]        copy:
[35]          src: "{{ lookup('pipe', 'ls -1 dist/{{ project_name }}-{{ version }}.tar.gz') }}"
[36]          dest: "{{ temporal_local_package_path }}"
[37]      rescue:
[38]      - debug: msg='Using local file has failed. Try to download package from
      arfifactory.'
[39]      - name: Download {{ project_name }}-{{ version }}.tar.gz
[40]        get_url:
[41]          url: http://artifactory.hi.inet/artifactory/yum-analytics/{{ project_name
      }}/dist/x86_64/{{ project_name }}-{{ version }}.tar.gz
[42]          dest: "{{ temporal_local_package_path }}"
[43]      tags: download_package
[44]
```

```
[45]  ## Esta parte de arriba es la que ejecutamos en local y ahora pasamos ya a la
      maquina
[46]  #donde desplegamos
[47]
[48]  - name: "Deploy Package"
[49]    hosts: "dataproc"
[50]    gather_facts: true
[51]    tasks:
[52]    - name: BLOCK - Check delegation
[53]      block:
[54]      - name: BLOCK - Install Package
[55]        tags: install_package
[56]        block:
[57]        - name: Create working directory - OS
[58]          file:
[59]            state: directory
[60]            path: "{{ working_dir }}"
[61]
[62]        - name: Unarchive package
[63]          unarchive:
[64]            src: "{{ temporal_local_package_path }}"
[65]            dest: "{{ working_dir }}"
[66]            extra_opts: [--strip-components=1] #en la carpeta destino no crees la carpeta
      comprimida
[67]
[68]      - name: BLOCK - Configure Package
[69]        tags: configure_package
[70]        block:
[71]        - name: Create configuration directory {{ workspace_setup }}
[72]          file:
[73]            path: "{{ workspace_setup }}"
```

```
[74]            state: directory
[75]
[76]          - name: Configure project run script
[77]            template:
[78]              src: templates/run_thor.sh.template
[79]              dest: "{{ working_dir }}/run_thor.sh"
[80]              mode: 0755
[81]
[82]        - name: BLOCK - Install requirements
[83]          tags: install_requirements
[84]          block:
[85]          - name: Update pip
[86]            pip:
[87]              name:
[88]              - pip==20.2.4
[89]              state: latest
[90]              virtualenv: "{{ virtualenv_ansible }}"
[91]              virtualenv_python: "{{ pyspark_python }}"
[92]            environment:
[93]              PATH: "{{ ansible_env.PATH }}:{{ path }}"
      LD_LIBRARY_PATH: "{{ ld_library_path }}"
```

run_thor.sh.template

```
[1]     #!/usr/bin/env bash
[2]
[3]     SPARK_MASTER=$1
[4]     DATASET_ID=$2
[5]     DATASET_VERSION=$3
[6]     DATASET_MAJOR_VERSION=$(echo ${DATASET_VERSION} | cut -d'.' -f 1)
[7]     FILTER_DATE=$4
[8]
```

```
[9]     DEPLOY_MODE={{ DEPLOY_MODE }}
[10]    DRIVER_MEMORY={{ DRIVER_MEMORY }}
[11]    DRIVER_CORES={{ DRIVER_CORES }}
[12]    NUM_EXECUTORS={{ NUM_EXECUTORS }}
[13]    EXECUTOR_MEMORY={{ EXECUTOR_MEMORY }}
[14]    EXECUTOR_CORES={{ EXECUTOR_CORES }}
[15]    QUEUE={{ QUEUE }}
[16]
[17]
[18]    export HADOOP_CONF_DIR={{ HADOOP_CONF_DIR }}
[19]    export YARN_CONF_DIR={{ YARN_CONF_DIR }}
[20]    export SPARK_HOME={{ SPARK_HOME }}
[21]    export SPARK_AVRO_VERSION={{ SPARK_AVRO_VERSION }}
[22]    export SPARK_SDK_TELEFONICA={{ SPARK_SDK_TELEFONICA }}
[23]
[24]    SW_HOME=/opt/thor4p-external
[25]
[26]    $SPARK_HOME/bin/spark-submit \
[27]      --deploy-mode ${DEPLOY_MODE} \
[28]      --master ${SPARK_MASTER} \
[29]      --driver-memory ${DRIVER_MEMORY} \
[30]      --driver-cores ${DRIVER_CORES} \
[31]      --num-executors ${NUM_EXECUTORS} \
[32]      --executor-memory ${EXECUTOR_MEMORY} \
[33]      --executor-cores ${EXECUTOR_CORES} \
[34]      --queue ${QUEUE} \
[35]      --packages ${SPARK_AVRO_VERSION} \
[36]      --class com.telefonica.baikal.thor.Main ${SW_HOME}/thor1.1.jar \
[37]      --dataset-id=${DATASET_ID} \
[38]      --dataset-version=${DATASET_MAJOR_VERSION} \
[39]      --filter-date=${FILTER_DATE} \
```

```
[40]      --local-mode \
[41]      --config-
          file=${SW_HOME}/utils/${DATASET_ID}/v${DATASET_VERSION}/json/loc
          al-config-file-${DATASET_ID}_${DATASET_VERSION}.json \
[42]      --output-result-
          path=/user/data_avro/thor_results/v2.0.0/data/${DATASET_ID}/${DATASET_M
          AJOR_VERSION}/${FILTER_DATE}/
[43]
[44]
```