UNIVERSIDAD DE ANTIOQUIA

Línea base de infraestructura en la nube

Autor(es)

John Alexander Galeano Ospina

Universidad de Antioquia

Facultad de Ingeniería

Medellín, Colombia

2020

Línea base de infraestructura en la nube

John Alexander Galeano Ospina

Tesis o trabajo de investigación presentado como requisito parcial para optar al título de:

Pregrado en Ingeniería de Sistemas

Asesores (a):

Carlos Mauricio Duque Restrepo, Ingeniero de Sistemas

Línea de Investigación:

Infraestructura del software

Universidad de Antioquia

Facultad de Ingeniería

Medellín, Colombia

2020

Tabla de Contenido

1. Tabla de figuras 4
2. Resumen 5
3. Introducción 5
4. Objetivos general y específicos 6
5. Marco teórico 6
6. Metodología 7
7. Resultados y análisis 8
8. Conclusiones 14
9. Referencias 15

1. Tabla de Figuras

Figura 1: Estructura del proyecto en Python.

Figura 2: Servicios del proyecto de ML.

Figura 3: Despliegue de la infraestructura desde Terraform.

Figura 4 - Comparativa de tiempos en los despliegues de infraestructura

Figura 5: Infraestructura de microservicios en un AKS.

Figura 6: Istio como malla de servicios.

Figura 7: WSO2 como ingress con un reverse ingress con Istio.

2. Introducción

Celerik siendo una empresa que realiza consultorías y desarrollo de aplicaciones web y móvil con la metodología SCRUM facilitando el desarrollo del producto según la necesidad del cliente; fundada el año 2008 en Medellín, e ingresando a Reino Unido en el 2018, donde se han desarrollado productos con la Space Agency, Transcends, y en Colombia productos con la alcaldía de Envigado, Ministerio de Minas y Energía de Colombia. Actualmente contamos con 2 sedes: Londres y Medellín. En Celerik, actualmente hago parte del equipo de infraestructura, se propone impulsar la implementación de nuevas herramientas en un proyecto piloto de aprendizaje de máquinas (Machine learning) desarrollado en python, esto con el fin de plantear una línea base.

Línea base de infraestructura para la implementación de proyectos de Machine Learning, compuesta de manejo de repositorios en los ambientes de la nube de Azure DevOps de Microsoft, pipelines para integración continua y despliegue/entrega continua (CI/CD), infraestructura para ambientes de ML con python y sus respectivas dependencias, estructura de código limpio y buenas prácticas de codificación (core, data access, views), manejo de microservicio por contenedores con la herramienta kubernetes (K8S), y otras instancias de las cuales Celerik no puede revelar más información debido a motivos de confidencialidad;

3. Resumen

Celerik en sus proceso de oferta de servicios en la nube, plantea una nueva línea de infraestructura en la nube para los campos de inteligencia artificial y proyectos en microservicios orquestado en contenedores (Kubernetes), para ello se propone una conjunto de tecnologías y herramientas para llevar a cabo la implementación de un proyecto de aprendizaje de máquinas (ML en sus siglas en inglés) en un ambiente de alta disponibilidad.

Utilizando el ciclo de calidad de software a fin de obtener los resultados esperados en cada etapa del desarrollo de infraestructura, se procede a realizar ciclo de integración y despliegue, se entrega al equipo de infraestructura una línea de repositorios y pipelines genéricos para adecuaciones según los requerimientos de posteriores proyectos.

4. Objetivo General

Implementar una estrategia de infraestructura limpia y escalable para los ambientes de ML de la empresa.

4.1 Objetivos específicos:

4.1.1 Crear repositorios enfocados en los ambientes de Python, en el servicio de Azure DevOps con sus respectivos pipelines.

4.1.2 Establecer un conjunto de pipelines para la infraestructura de Kubernetes.

4.1.3 Definir una estructura con buenas prácticas en el enfoque de estructura limpia para proyectos de Machine Learning en Python.

4.1.4 Definición e implementación de microservicios por contenedores orquestado por Kubernetes K8S con un API Management en WS02 Enterprise y ISTIO .

4.1.5 Telemetrías de los servicios desde la App Center de Azure.

5. Marco teórico

El proyecto de línea base estará enfocada en la infraestructura de software en contenedores para ello se quiere implementara la automatización por parte de un orquestador de contenedores en la tecnología K8S [2], con el principio de infraestructura como código (IaC)[3] para la autonomía de una nube descentralizada de un solo proveedor. Enfocándonos principalmente en un proyecto de Machine Learning como proyecto de código piloto, pero resultando en un flujo de trabajo genérico en contenedores para cualquier tipo de proyecto de desarrollo, cada Sprint está especificado en la herramienta Azure DevOps donde se realizar los pull request del trabajo de desarrollo hecho durante el día, las herramientas de desarrollo más importantes son Python, Kubernetes, Docker, Terraform y Bash script.

Como orquestador de microservicios se implementó Kubernetes(k8s) para el control de contenedores como herramienta de automatización de implementación y administración de contenedores. Esto facilita la gestión de código base de los servicios y poder probar las entradas y salidas específicas.

Integración con WSO2 API Manager[4], API Manager es una solución de código abierto que facilita la gestión de las API. Este software te permite:

-Diseñar y construir prototipos API.

-Publicar y administrar el uso de API.

-Control de accesos y fortalecimiento de la seguridad.

-Administrar una comunidad de desarrolladores.

-Cree un lugar para almacenar todas las API disponibles.

-Administrar el tráfico API.

-Supervisar el uso y el rendimiento de las API a través de análisis.

En general, lo que hace es proporcionar un punto de acceso principal desde el cual administrar, supervisar y tener un acceso seguro a los servicios web públicos.

Se utilizó la herramienta de Istio malla de servicio[5], permite enrutar el tráfico según los criterios que defina. Lo logra mediante el uso de proxies Envoy como sidecars dentro de cada pod y manteniendo un registro de servicio en su plano de control. En el dominio de seguridad, los servidores proxy de Envoy y el plano de control le permiten administrar el tráfico entre servicios estableciendo políticas y encriptando el tráfico dentro del clúster. Istio le ofrece capacidad de observación mediante el seguimiento de las métricas del servicio, el seguimiento del tráfico y la realización de tareas de registro de aplicaciones en todo el clúster.

6. Metodología

La metodología de desarrollo es SCRUM [1], es una metodología ágil con el propósito de realizar un producto que se construye en una serie de iteraciones llamadas sprints que descomponen pedazos del proyecto, cada sprint tiene una duración de 2 semanas, cada día se discuten los problemas en el desarrollo con el Scrum Master para ir avanzando en las funcionalidades.

Siguiendo los lineamientos del conjunto de tecnologías en Celerik, se implementó la herramienta Terraform para realizar la infraestructura como código (IaC por sus siglas en inglés) para automatizar el aprovisionamiento de recursos de infraestructura. Su uso en el proceso de la línea base focalizó para construir, administrar, actualizar y eliminar los recursos de infraestructura como son máquinas virtuales, contenedores, elementos de redes, grupos de trabajo, entre otros. Terraform es una herramienta independiente del proveedor de los recursos

(por ejemplo AWS, Azure, Google Cloud, Heroku, Oracle, etc) lo cual se alinea con la filosofía de la empresa de prestar servicios independiente de la nube.

En términos simples, si desea aprovisionar una Nube privada virtual o una instancia EC2 de AWS, puede escribir una configuración de terraformación para automatizar este proceso en lugar de hacerlo manualmente desde la consola de AWS.
Los beneficios de la infraestructura como código que se consideraron para aplicar al proyecto fueron:

- Desarrollo modular: Permitir tratar nuestra infraestructura como un software que se puede escribir una vez y usar varias veces. Esto hace la vida mucho más fácil, porque ahora podemos reutilizar el código que se escribe una vez
- Metodologías de desarrollo de software: Prácticas comprobadas como el control de versiones, el desarrollo modular, las pruebas, etc. en el mundo de la infraestructura
- Resolución de problemas: Depurar y determinar la causa raíz del problema fácilmente: como mantenemos la infraestructura como un código, podemos aprovechar los sistemas de versiones, por lo que cualquiera que intente depurar un problema o un problema con la infraestructura puede consultar el historial de cambios hecho y rastrear el problema
- Agilidad: Agilizar todo el proceso de desarrollo e implementación de aplicaciones al garantizar una menor dependencia del trabajo manual, lo que reduce los errores

7. Resultados y análisis

7.1. Se almacena el código del proyecto en Python en un repositorio de Azure DevOps, con el enfoque de carpetas o módulos por función, esto siguiendo las recomendaciones para un código limpio en Python[6] la cual se puede visualizar en la Figura 1. Se crea el archivo docker y se registra en un espacio de un Docker

Registry privado en Azure.

```
├── blueprint  # Our source code - name of the application/module
│   ├── app.py
│   ├── __init__.py
│   ├── __main__.py
│   └── resources
├── tests
│   ├── conftest.py
│   ├── context.py
│   ├── __init__.py
│   └── test_app.py
├── .github  # GitHub Actions
│   └── workflows
│       ├── build-test.yml
│       └── push.yml
├── Makefile
├── configure_project.sh
├── setup.cfg
├── pytest.ini
├── requirements.txt
├── dev.Dockerfile
└── prod.Dockerfile
```

Figura 1 - Estructura del proyecto en Python.

7.2. EL uso de Terraform en el despliegue de recursos en la nube representó una reducción de dos (2) horas operacionales en media, de 54 despliegues en entorno de desarrollo.

7.3. Los servicios del proyecto ML se presentan en la Figura 2 desplegados en un entorno de contenedores en una máquina

virtual.

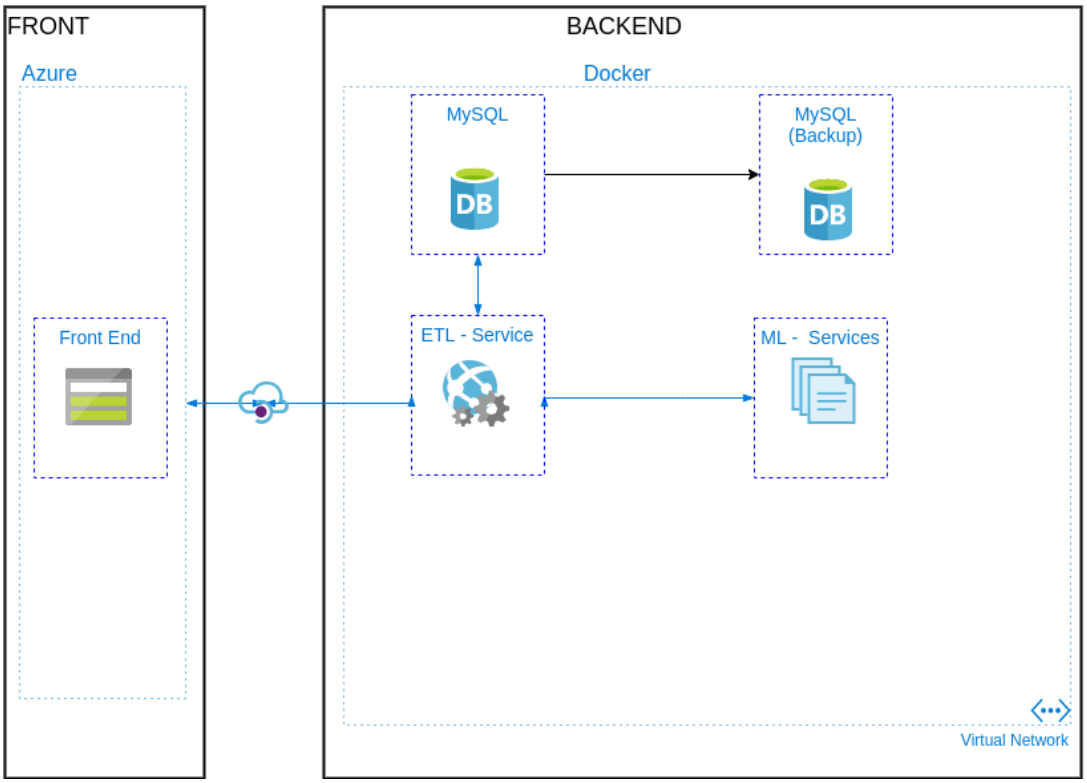

Figura 2 - Servicios del proyecto de ML.

7.4. Línea base en terraform se implementa para recursos bases para una infraestructura en microservicios, el recurso principal es el Azure Kubernetes Service(AKS) en un módulo de Terraform, dentro de un repositorio privado en Azure DevOps. Los nodos de este cluster en AKS tiene recursos de alta demanda como son gráficas y memoria bajo demanda con capacidades de hasta 64 Gb, por los requisitos de un proyecto de aprendizaje de máquinas para el proyecto.

7.5. Terraform nos permitió un despliegue controlado de los principales recursos principales de Azure como el AKS, API Manager, virtual networks(vnet) y sus correspondientes subnets, como recursos de control de carga como balanceadores de carga, y baúles de contraseñas/claves, todo se presenta en la

Figura 3.

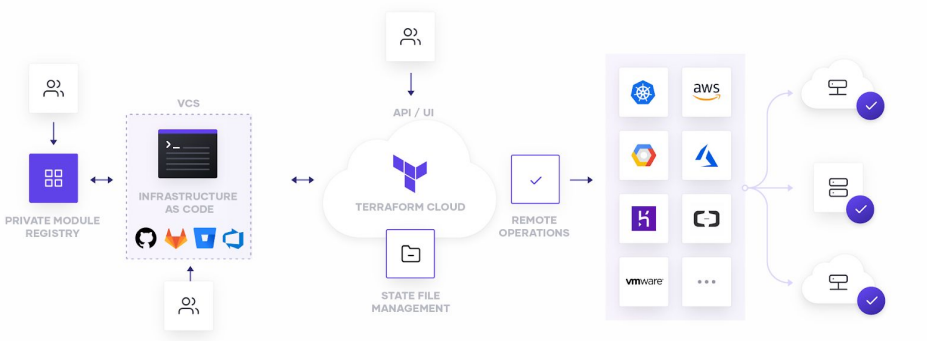

Figura 3 - Despliegue de la infraestructura desde Terraform.

7.6. La infraestructura base descrita en el numeral anterior se registra con un número de despliegues en ambiente Desarrollo y Producción de 64 y 52 respectivamente, con un control de versiones e impotencia de los recursos en la nube de casi el 96.66% de precisión respecto a la planificación de recursos en cada despliegue(figura 4).

Método					
Portal	Recurso	Tiempode implementación (Minutos)	Errores en la implemntación	Tiempo de despliegue (Minutos)	Errores en el despliegue
	AKS	50	10	40	2
	APIM	45	15	30	2
	VNET	25	2	15	1
AZ CLI	Recurso	Tiempo	Errores	Tiempo	Errores
	AKS	35	10	35	1
	APIM	25	12	27	1
	VNET	15	4	12	1
Script	Recurso	Tiempo	Errores	Tiempo	Errores
	AKS	40	15	25	2
	APIM	25	10	24	3
	VNET	15	5	10	2
Terraform	Recurso	Tiempo	Errores	Tiempo	Errores
	AKS	25	2	15	0
	APIM	15	2	10	0
	VNET	5	1	1	0

Datos promedio al efectuar 64 y 52 despliegues en ambiente Desarrollo y Producción respectivamente

Figura 4 - Comparativa de tiempos en los despliegues de infraestructura

7.7. Identificación de errores en el flujo de la implementación de la metodología Scrum en un proyecto con cultura DevOps, donde no solo se debe considerar la minimización del tiempo, sino cumplir con la calidad de las entregas por igual prioridad, de los errores identificados: falta de linters (un análisis estático del código fuente), pruebas en las plantillas de Terraform, control de flujo de despliegue de los recursos de infraestructura en producción(Bases de datos, registro de contenedores, etc).

7.8. Se implementa un pipeline para el despliegue de la infraestructura, con el propósito de reaccionar a los nuevos Pull Request y realizar las respectivas pruebas de creación y despliegue de los recursos en cada nueva versión del código,

presentado en la Figura 5.

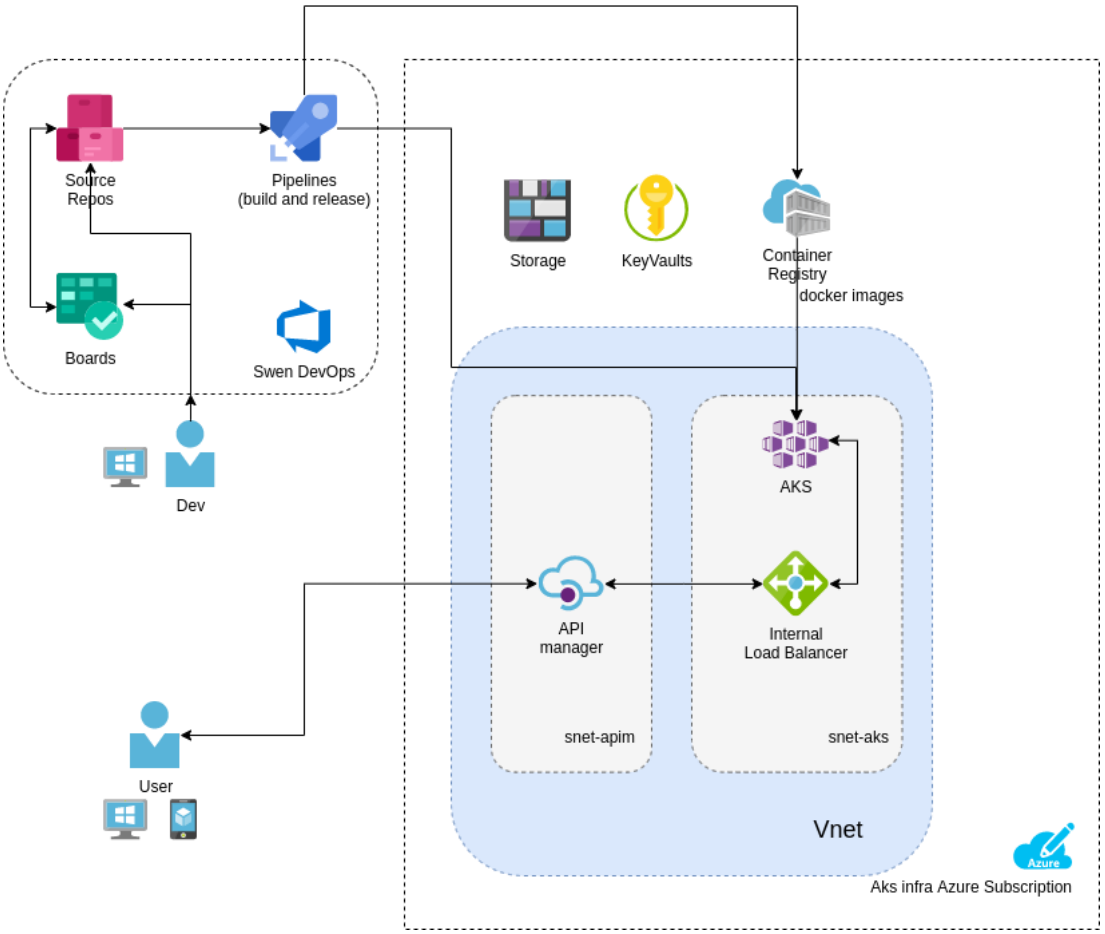

Figura 5 - Infraestructura de microservicios en un AKS.

7.9. Se implementa un pipeline y flujo de despliegue de los servicios del repositorio de Python en sus respectivos contenedores para el cluster de AKS desplegado con anterioridad.

7.10. Se realiza una investigación y prueba de concepto de WSO2 en un ambiente de kubernetes. Al analizar su carga si el recurso de WSO2 es factible en su costo y beneficio se decide implementar una versión simplificada con microgateway WSO2, esta no requiere tantos recursos de cómputo como la versión completa planteada al inicio del proyecto.

7.11. Istio como la malla de servicio para lograr un control y seguridad entre los servicios desplegados dentro del AKS (Figura 6).

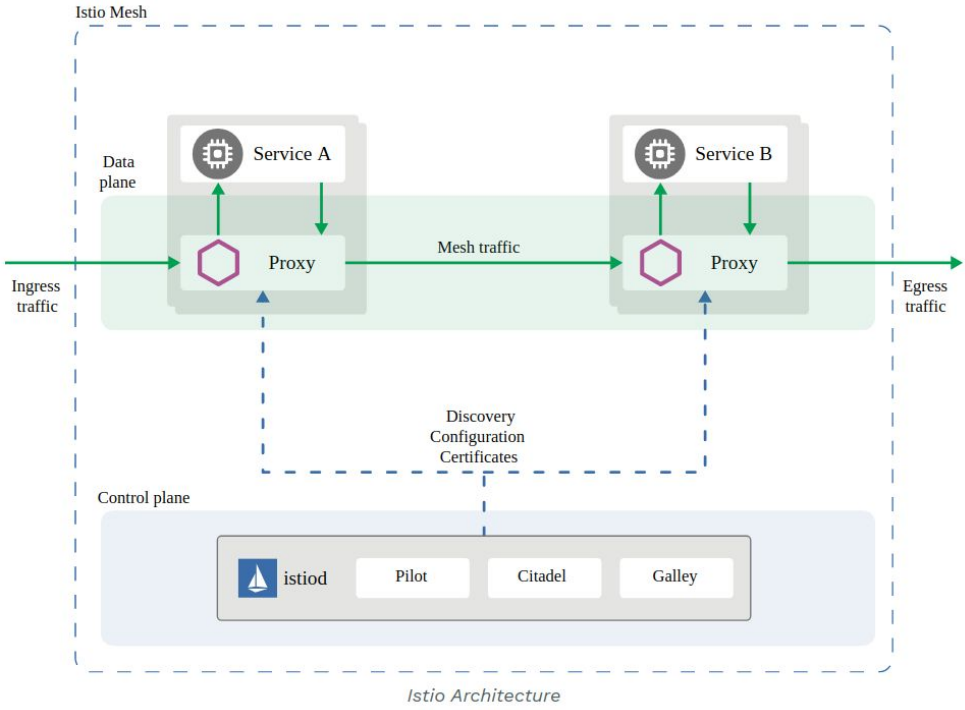

Figura 6 - Istio como malla de servicios.

7.12. Se utilizan las herramientas de microgateway WSO2 y Istio, para tener un control en la visibilidad tanto externa e interna de los servicios del proyecto de ML, una generalización de las dos herramientas se muestra en la Figura 7.

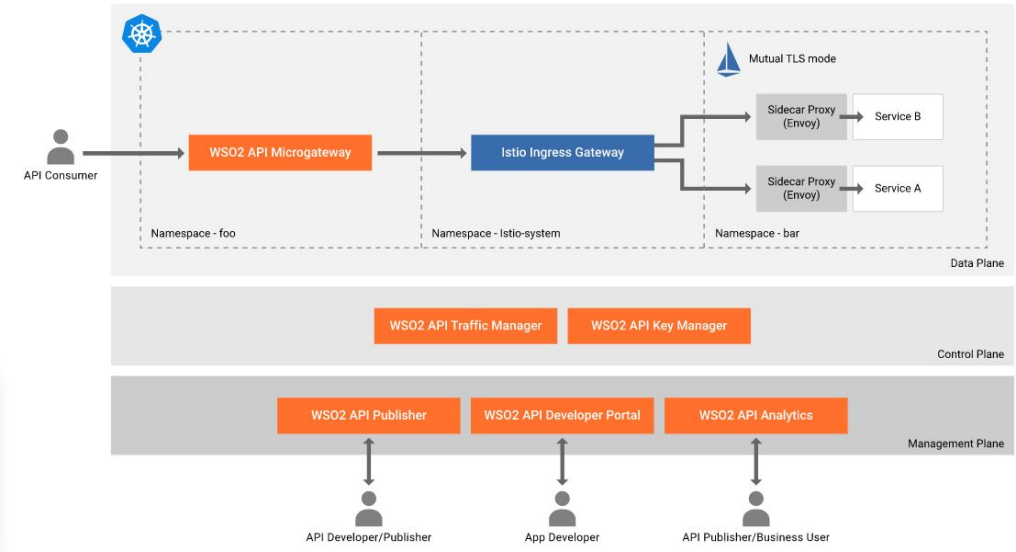

Figura 7 - WSO2 como ingress con un reverse ingress con Istio.

7.13. El despliegue de los microsiervos al AKS a través del pipeline se realizó con pasos de verificación de creación de los mismos y disponibilidad dentro del cluster.

7.14. Todos los procesos y herramientas se entregan con la documentación en cada repositorio, esto con el enfoque de tener versionamiento de la misma en cada iteración de los proyectos.

8. Conclusiones

8.1. Terraform nos ayudó en el control de recursos, y el versionamiento en la infraestructura, en comparación con la metodología de implementación desde el portal, la cual representaba un riesgo para el seguimiento y control de errores, esto es, si no se tenía una documentación apropiada, el conocimiento se centralizaba en solo la persona que lo realizó.

8.2. Terraform implicó una mejora en los tiempos de creación de los recursos en la nube de Azure, con una mejora en los tiempos de despliegue de 1 minuto para la orden de creación, en comparación del flujo del portal que estaba en promedio de 40 a 60 minutos para los recursos de service principal, key vault, blog storage, AKS, API manager y virtual networkings .

8.3. El flujo de una proyecto en Python con buenas prácticas representó un reto en la codificación y modularización del código, al ser un proyecto nuevo tanto en el campo de aprendizaje de máquinas (ML) y en el propio lenguaje de programación, el cual se agregara al stack de servicios en la empresa; al lograr este despliegue de servicios en la nube, se cumplio con la meta de tener un proyecto de presentación a los clientes tanto en el ámbito de proyectos Python y de ML.

8.4. Los microservicios en Python corrieron de forma satisfactoria(respuesta entre los servicios de menos de 40 ms) y su comunicación está entre las medidas establecidas en los canales (puertos seguros 443 y protocolo HTTP por su comunicación interna en el cluster).

8.5. La gestión de los servicios y sus interfaces (API) con la herramienta Micro Gateway de WSO2 facilitó el control y visualización de los mismos, al contar con una visualización interactiva y mapeada de los endpoints de los servicios, incluso permitió el análisis de cobertura y salud de los mismos con los diagnósticos que tiene la herramienta.

8.6. Istio como nuestra malla de servicios permite la estandarización de la infraestructura que ofrecen permite que la seguridad, la gestión del tráfico y los desafíos de observabilidad se eliminen de las manos de los desarrolladores y se gestionen de forma centralizada. Sin embargo, estos beneficios no son gratuitos y las

mallas de servicio como Istio son famosas por sus complejidades de administración. También son relativamente nuevos y están cambiando rápidamente. El potencial de las mallas de servicio hace que puedan persistir a medida que más y más organizaciones se trasladan a arquitecturas de microservicios.

8.7. La implementación de Istio como malla de servicio representó un costo adicional, incluso si facilitó el control interno de los diferentes recursos del AKS, se descartó para proyectos pequeños como fue el caso de este proyecto de ML, ya que el coste de tener una malla de servicios eleva los recursos de cómputo.

9. Referencias

[1] K. Schwaber and J. Sutherland, *The Definitive Guide to Scrum: The Rules of the Game*, 1st ed. Creative Commons, 2019, pp. 3-9.

[2] Burns, B. (2019). Kubernetes - Up and Running : Dive into the Future of Infrastructure. O'Reilly Media, Incorporated.

[3] Brikman, Y. (2019). Terraform: Up & Running: Writing Infrastructure as Code

[4] Overview of the WSO2 API Gateway - WSO2 API Manager Documentation 3.1.0. (2020). Retrieved 5 August 2020, from https://apim.docs.wso2.com/en/latest/learn/api-gateway/overview-of-the-api-gateway[5] What is Istio?. (2020). Retrieved 5 August 2020, from https://istio.io/latest/docs/concepts/what-is-istio/

[6] Structuring Your Project — The Hitchhiker's Guide to Python. (2020). Retrieved 5 August 2020, from https://docs.python-guide.org/writing/structure/