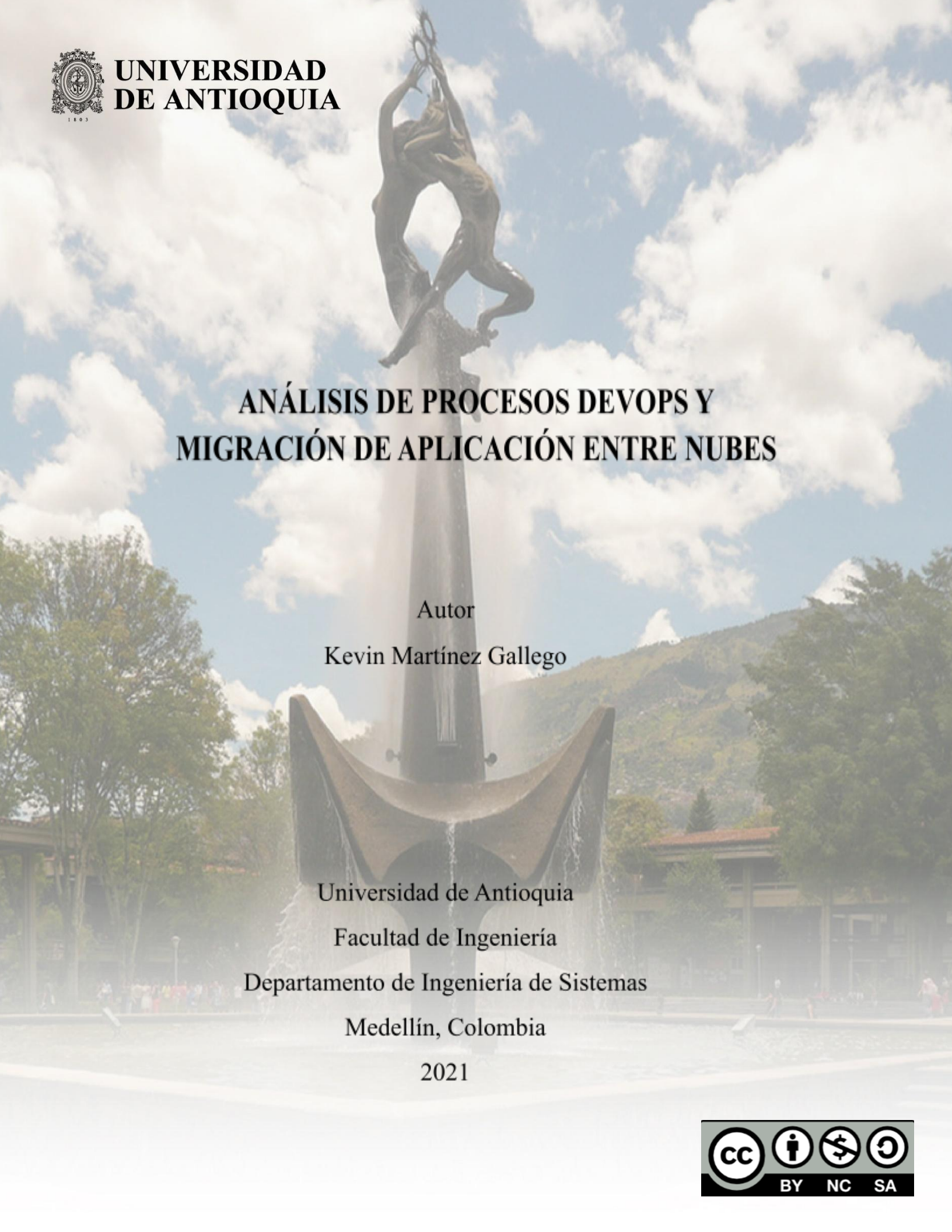


UNIVERSIDAD DE ANTIOQUIA

ANÁLISIS DE PROCESOS DEVOPS Y MIGRACIÓN DE APLICACIÓN ENTRE NUBES

Autor

Kevin Martínez Gallego

Universidad de Antioquia

Facultad de Ingeniería

Departamento de Ingeniería de Sistemas

Medellín, Colombia

2021

Análisis de Procesos DevOps y Migración de Aplicación Entre Nubes

Kevin Martínez Gallego

Informe de práctica académica presentado como requisito parcial para optar al título de:
Ingeniero de Sistemas

Asesores:
Carlos Mauricio Duque Restrepo, Ingeniero de Sistemas
Joaquín Iván Barrera Lozada, Ingeniero Electrónico

Universidad de Antioquia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas
Medellín, Colombia
2021

ÍNDICE

RESUMEN **2**

INTRODUCCIÓN **2**

OBJETIVOS **4**

- Objetivo General 4
- Objetivos Específicos 4

MARCO TEÓRICO **5**

METODOLOGÍA **7**

RESULTADOS Y ANÁLISIS **8**

CONCLUSIONES **17**

REFERENCIAS **19**

RESUMEN

Los componentes principales del aplicativo EndavaCares se encuentran alojados en un ecosistema de la plataforma AWS; dicho proveedor de nube cuenta con amplia ocupación en la sede Medellín de la empresa Endava. Sin embargo, otra plataforma en la nube como Microsoft Azure, se presenta como una alternativa factible en la ejecución de proyectos. Así, un ejercicio de migración de los componentes del aplicativo, desde AWS hacia Azure, sirvió como insumo para realizar un análisis holístico del proyecto, incluyendo las prácticas de DevOps implementadas. En consecuencia, se mejoró el proceso de Integración Continua mediante la adopción de buenas prácticas de codificación, y la reducción de los tiempos de ejecución del pipeline de CI/CD (implementado usando Jenkins); además, se proporcionaron métricas de valor para la toma de decisiones, obtenidas de los análisis de código de SonarQube. Se logró evolucionar el proyecto desde la condición de Entrega Continua a Despliegue Continuo, mediante la incorporación (utilizando Docker) de pruebas de API e interfaz gráfica al pipeline de CI/CD. Posteriormente, se ejecutó el proceso de migración de los componentes, generando una guía paso a paso como artefacto de documentación. Finalmente, se implementó la arquitectura original en AWS siguiendo el paradigma de IaC, empleando Terraform; así se automatizó el proceso de despliegue y configuración de la infraestructura, haciéndolo más eficiente y eficaz.

INTRODUCCIÓN

En Junio de 2020, la sede Medellín de la empresa Endava puso en marcha el proyecto EndavaCares, un proyecto interno de la empresa tipo *bench* [1], que busca cumplir con los requisitos para empresas públicas y privadas en Colombia expuestos en [2], en relación con el manejo de la pandemia debida al virus COVID-19. El objetivo del proyecto EndavaCares se enmarca en generar una herramienta de valor para la empresa, mientras se exploran tecnologías y prácticas innovadoras, como es usual en los proyectos de este tipo. Los componentes principales del proyecto (Frontend, Backend y Base de Datos) están alojados en un ecosistema de la plataforma Amazon Web Services (AWS), como se muestra en la Figura 1; el componente Frontend corresponde a la interfaz gráfica del sistema con que interactúa el usuario final, mientras que el componente Backend denota la implementación de la lógica del negocio y el acceso a base de datos [3]. Aunque en la sede Medellín de Endava existen proyectos y comunidades de aprendizaje enfocadas en esta plataforma, no hay un enfoque significativo para otros proveedores de servicios en la nube como Microsoft Azure y Google Cloud Platform (GCP), que no obstante, se presentan como una alternativa relevante para proyectos futuros de la empresa. Así, este proyecto se revela como una oportunidad provechosa para estudiar y emplear los servicios de Microsoft Azure enmarcados en un proyecto de software real.

En el desarrollo de software, el término DevOps se refiere a un conjunto de prácticas que agregan valor a las metodologías de desarrollo ágiles (a menudo se define el término como cultura o filosofía). Con este enfoque se pretende aumentar la calidad del software, a través de la ejecución de pruebas en diferentes niveles de las aplicaciones, así como mejorar la productividad del equipo de trabajo a través de la automatización de procesos, la colaboración activa y una retroalimentación continua. Al momento de ingresar al proyecto, se tenían implementadas algunas prácticas de la cultura DevOps, aunque no se había alcanzado un nivel de madurez suficiente para que el proyecto gozara de despliegue continuo; es decir, llegar a un estado en el que el código hecho por los desarrolladores es desplegado en el ambiente de producción (con el que interactúa el usuario final) de manera automática y confiable [4]. Esta inmadurez se debía en gran medida a la carencia de pruebas automatizadas en algunos componentes específicos de la aplicación, y a las falencias presentes en las existentes.

Así, realizar un ejercicio de migración de los componentes de la aplicación desde la plataforma AWS a la plataforma Microsoft Azure, se presenta como una actividad apropiada para identificar puntos de mejora en la arquitectura, los componentes y las prácticas DevOps implementadas. Este ejercicio es apto como escenario para analizar el estado actual del proyecto en aspectos como desarrollo, implementación de pruebas, infraestructura, despliegue y procesos de monitoreo, así como proponer acciones de mejoramiento y materializar dichas propuestas. Adicionalmente, se alimenta positivamente el enfoque de aprovisionamiento de servicios en la plataforma Microsoft Azure, lo cual representa una condición muy provechosa para la empresa. El desarrollo de las diferentes tareas se lleva a cabo bajo una metodología de trabajo ágil, en este caso SCRUM, que es el enfoque predominante de ejecución de proyectos en Endava.

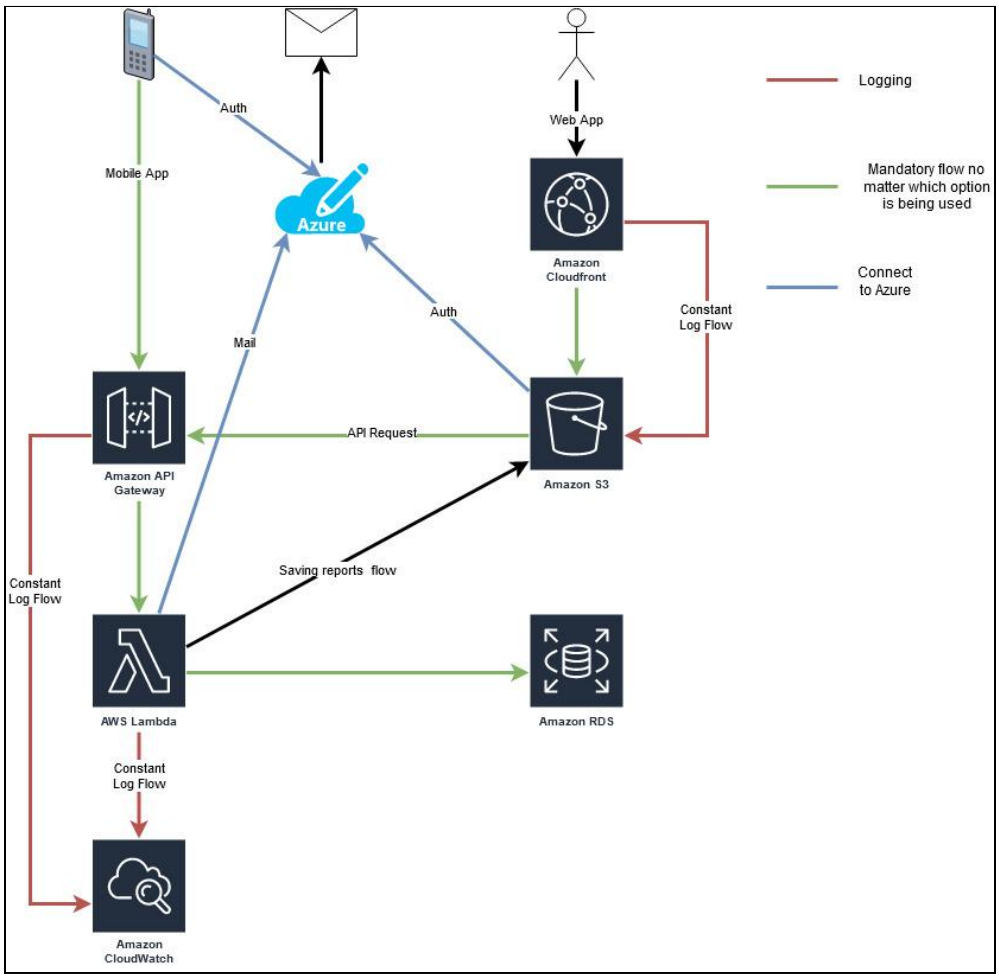


Figura 1: Componentes del proyecto EndavaCares sobre la plataforma AWS

OBJETIVOS

Objetivo General

- Realizar el análisis del aplicativo EndavaCares, identificando oportunidades de mejora en los procesos de integración continua, entrega continua y despliegue continuo, a través de la ejecución de un ejercicio de migración a la plataforma Microsoft Azure.

Objetivos Específicos

- Hacer un análisis de la arquitectura y componentes actuales del aplicativo.
- Determinar los posibles servicios a utilizar en Microsoft Azure, en comparación con la arquitectura actual.

- Efectuar un análisis de las prácticas actuales de DevOps en el proyecto: integración continua, entrega continua y despliegue continuo.
- Documentar los hallazgos encontrados en los análisis realizados.
- Proponer mejoras sobre los procesos de integración continua, entrega continua y despliegue continuo a partir de los hallazgos encontrados.
- Adquirir aptitudes en herramientas y metodologías asociadas a la cultura DevOps.
- Promover buenas prácticas en cada una de las etapas del flujo de trabajo del proyecto, siguiendo los principios de la cultura DevOps.
- Efectuar la migración de los servicios de almacenamiento.
- Ejecutar la migración de los servicios de cómputo.
- Realizar la migración de los servicios de red.
- Hacer la migración de los servicios de monitoreo.
- Desarrollar la migración de servicios de credenciales.
- Documentar los procesos llevados a cabo durante la migración de los componentes del aplicativo.

MARCO TEÓRICO

Amazon es reconocida como una empresa precursora en el paradigma de computación en la nube (Cloud Computing), que ofrece sus servicios a través de su plataforma AWS desplegada a nivel global [5]. Dicha plataforma fue seleccionada para albergar los componentes del aplicativo EndavaCares de la siguiente manera, y tal como se muestra en la Figura 1: en lo que respecta al Frontend (desarrollado con el framework React sobre un entorno de Node.js) el servicio Amazon S3 para entregar los estáticos en conjunción con el Content Delivery Network Amazon CloudFront; el Backend (implementado en el lenguaje de programación Go) corresponde a un arquitectura Serverless materializada a través de Amazon API Gateway y AWS Lambda Functions; el monitoreo se realiza con Amazon CloudWatch; y la base de datos (PostgreSQL) corresponde a una instancia de Amazon RDS. No obstante, la plataforma Microsoft Azure se ha consolidado como una opción muy relevante al momento de seleccionar un proveedor de servicios en la nube, y sus avances y mejoras en los últimos años han sido muy significativos. Su gran variedad de servicios, integración entre estos y su estructura de costos, han sido aspectos esenciales para muchas empresas a nivel global, que han adoptado esta plataforma como predilección para el aprovisionamiento de sus productos de software [6]. Los servicios equivalentes para el aplicativo EndavaCares en esta plataforma serían, respectivamente: Azure Blob Storage para la entrega de estáticos, en combinación con Azure Content Delivery Network; Azure API Management y Azure Functions para la arquitectura Serverless del Backend; Azure Monitor para las tareas de monitoreo; y Azure Database for PostgreSQL para la base de datos.

Adicionalmente, el proyecto tiene un componente de DevOps involucrado. DevOps es una estrategia de desarrollo de software que integra equipos multifuncionales para agregar valor a las metodologías de desarrollo ágiles; esto se logra a través de un conjunto de principios como la comunicación y colaboración activa, la automatización de procesos, la aplicación de métricas para medir el progreso, y la retroalimentación constante. El núcleo de la cultura DevOps es recurrir a un modelo iterativo que parte de la planeación y el diseño, se nutre de la integración continua, entrega continua y el despliegue continuo (CI/CD por sus siglas en Inglés), y se mantiene mejorando activamente a través de la retroalimentación y el monitoreo persistente [4]. La integración continua se refiere a una práctica en la que los desarrolladores incorporan sus funcionalidades al código fuente, de manera recurrente, tal que automáticamente se realiza una verificación para comprobar que los nuevos cambios funcionan adecuadamente, y no afectan al código ya existente. En caso afirmativo, se despliegan automáticamente las nuevas funcionalidades en un ambiente de pruebas o de producción, lo que corresponde a la práctica de entrega continua; o despliegue continuo si el proyecto es lo suficientemente robusto como para permitir que todo el flujo de trabajo, desde el código desarrollado hasta el ambiente de producción, se ejecute sin intervención humana, donde solo un fallo en alguna de las pruebas automatizadas prevendría un despliegue al ambiente productivo [7]. Así, un objetivo fundamental de DevOps es aumentar la calidad del software, apelando a un proceso de automatización de pruebas en los diferentes niveles de la aplicación; es aceptado seguir una estrategia piramidal que consta de pruebas de unidad, pruebas de servicios y pruebas de interfaz de usuario, tal como muestran Mike Cohn y Martin Fowler en [8] y [9] respectivamente. La idea de esta cultura es cerrar la brecha entre el personal de desarrollo (quienes escriben el código fuente) y el personal de operaciones (quienes despliegan y operan el código). Surge entonces el rol de DevOps Engineer, quien es una persona con aptitudes en ambas áreas y con la actitud de promover activamente dicha cultura en su equipo de trabajo [4].

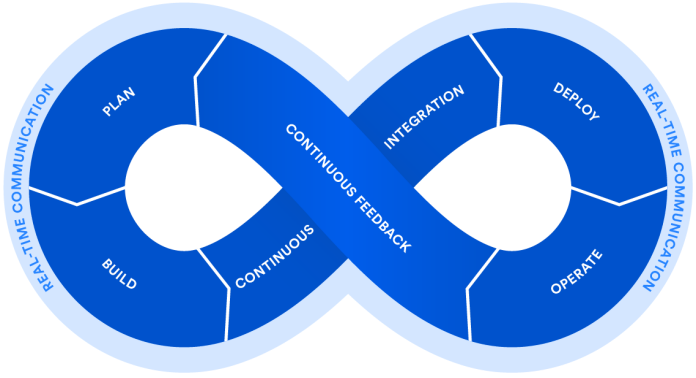


Figura 2: Ciclo de vida DevOps [4]

Es sabido que dentro del Ciclo de vida DevOps, se involucran diversos conceptos que se apoyan en una gran cantidad de herramientas, y que su elección depende en gran medida de las necesidades y constreñimientos propios de cada proyecto. Para EndavaCares, las siguientes son algunas herramientas y conceptos de relevante importancia:

- Docker: es una herramienta de virtualización a través de contenedores; un contenedor corresponde a una unidad estándar de software que permite "empaquetar" código fuente y sus dependencias, tal que este puede ser ejecutado de manera fiable y eficiente en diferentes entornos informáticos [10] [11].
- Jenkins: es un proyecto de código abierto que permite instanciar un servidor de automatización, con el objetivo de implementar prácticas de integración continua y despliegue continuo en proyectos de software [12].
- SonarQube: es una herramienta que permite realizar análisis estático de código fuente, con el objetivo de asegurar estándares apropiados de codificación para diferentes lenguajes de programación [13].
- Terraform: es una herramienta de código abierto que permite administrar la infraestructura de un proyecto de software (hardware y configuraciones de red) como código, a través de un lenguaje de programación declarativo (HCL) y sobre diferentes proveedores de servicios en la nube como AWS, Microsoft Azure, GCP, entre otros [14] [15]. El paradigma de infraestructura como código es usualmente referido como IaC por sus siglas en Inglés (Infrastructure as Code).

METODOLOGÍA

El desarrollo de las actividades del proyecto se enmarcó dentro del Framework de trabajo ágil SCRUM. Este tipo de marcos de trabajo, cuyos valores y principios están consignados en el manifiesto ágil [16], surgieron de la necesidad de crear un entorno de desarrollo de software flexible, progresivo y adaptable a las necesidades de los clientes, en lugar de las metodologías tradicionales como Waterfall y RUP, que son menos adaptables al cambio. Así, SCRUM encierra los siguientes conceptos:

- Roles
    - Scrum Master: es la persona que lidera el equipo de trabajo y se encarga de garantizar la correcta implementación de la metodología.
    - Scrum Team: equipo de trabajo compuesto por desarrolladores, personal de DevOps (el rol que desempeñé) y personal de QA (Quality Assurance).
    - Stakeholders: los interesados en el proyecto, es decir, los usuarios finales.
    - Product Owner: representante de los stakeholders que interactúa con el Scrum Master y el Scrum Team.

- Artefactos
    - Product Backlog: conjunto de todas las tareas a completar por parte del Scrum Team para culminar el proyecto.
    - Sprint Backlog: subconjunto de tareas que se abordan en un Sprint específico.
    - Product Increment: incremento notable en la funcionalidad del producto.
- Actividades
    - Sprint: ejecución de tareas en iteraciones de dos semanas.
    - Sprint Planning: ceremonia en la que se planea el siguiente Sprint y se escoge el Sprint Backlog.
    - Daily Meeting: reunión diaria en la que cada miembro del Scrum Team esboza su avance con sus tareas, o reporta si tiene dificultades.
    - DEMO y Sprint Review: al finalizar un sprint se muestra a los stakeholders y al Product Owner los incrementos en el producto.
    - Sprint Retrospective: se analiza el sprint que terminó evaluando que se hizo bien, que se hizo mal, y que cosas se pueden mejorar.
    - Refinement Session: se refina el backlog de cara al próximo Sprint.

La ejecución del Framework se apoyó en la herramienta JIRA, la cual nos permite administrar el Product Backlog y Sprint Backlog, así como realizar asignación de tareas y hacer seguimiento de ellas.

RESULTADOS Y ANÁLISIS

Los resultados de este proceso de práctica académica, enmarcada en el proyecto EndavaCares, corresponden a una amalgama de mejoras sobre las prácticas de CI/CD en la implementación de la cultura DevOps del proyecto, a la par con el ejercicio de migración a la plataforma Microsoft Azure, cuyo análisis y ejecución permitió tener un mejor entendimiento del proyecto, y por lo tanto, potenciar más perspicacia al momento de identificar puntos de mejora.

Inicialmente, aunque en el Backend se estaban desarrollando pruebas unitarias, se evidenció la falta de cualquier tipo de pruebas en el componente Frontend del proyecto. Sabiendo que se empezarían a desarrollar pruebas automatizadas para este último, se decidió agregar previamente un componente de validación del código fuente en cuanto a formato y buenas prácticas se refiere, con el objetivo de mejorar el proceso de integración continua. Se realizó entonces la configuración de las herramientas Prettier (formateador de código) y ESLint (Linter) en el proyecto Frontend, en conjunción con una estrategia de pre-commit que adoptaron los desarrolladores a partir de ese momento. Las Figuras 3 y 4,

obtenidas a partir de los análisis de SonarQube, muestran un antes y un después de las métricas del proyecto con respecto al punto de mejora en cuestión: es evidente como disminuyeron los *Code Smells*, alcanzando su punto más bajo desde que estas métricas empezaron a calcularse, y reduciendo así la deuda técnica.

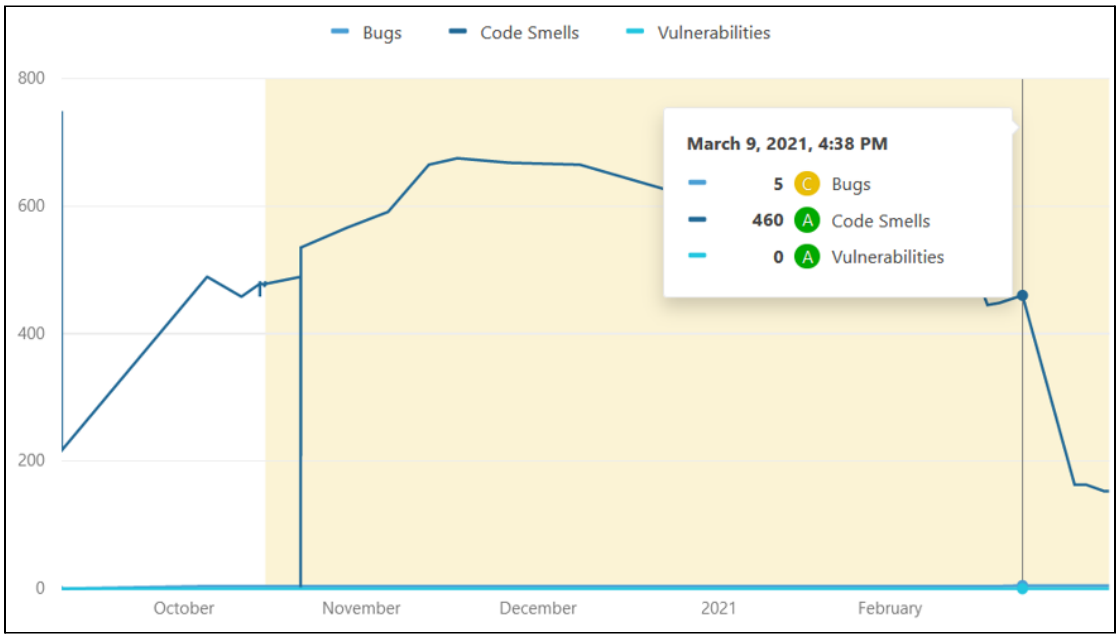


Figura 3: Métricas SonarQube proyecto Frontend antes del formateo de código fuente y adopción de pre-commit por parte de los desarrolladores

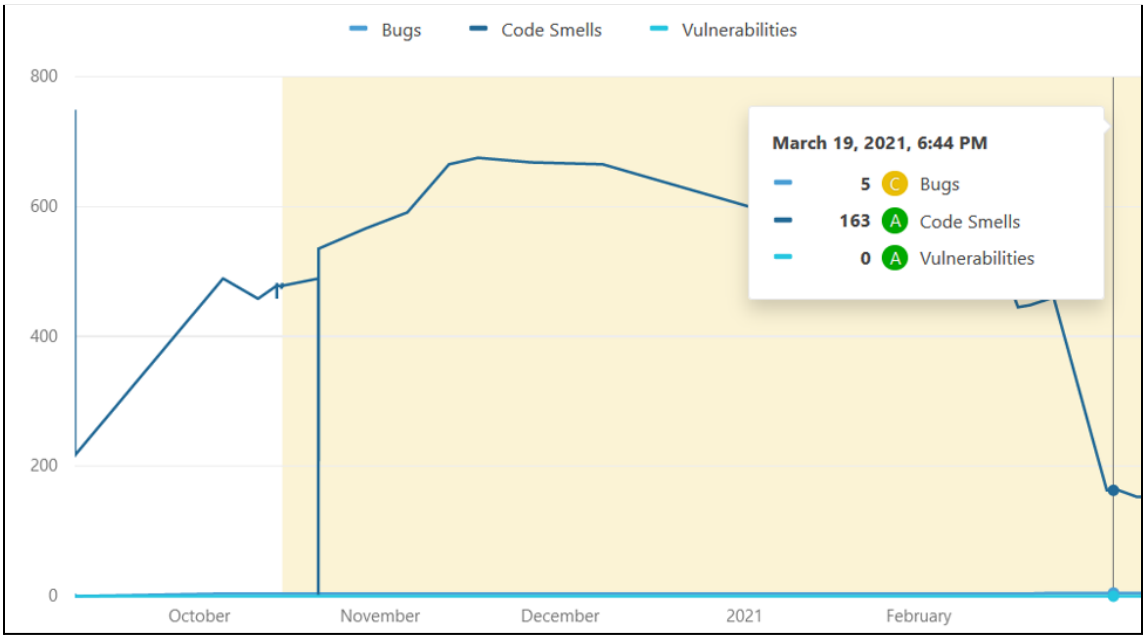
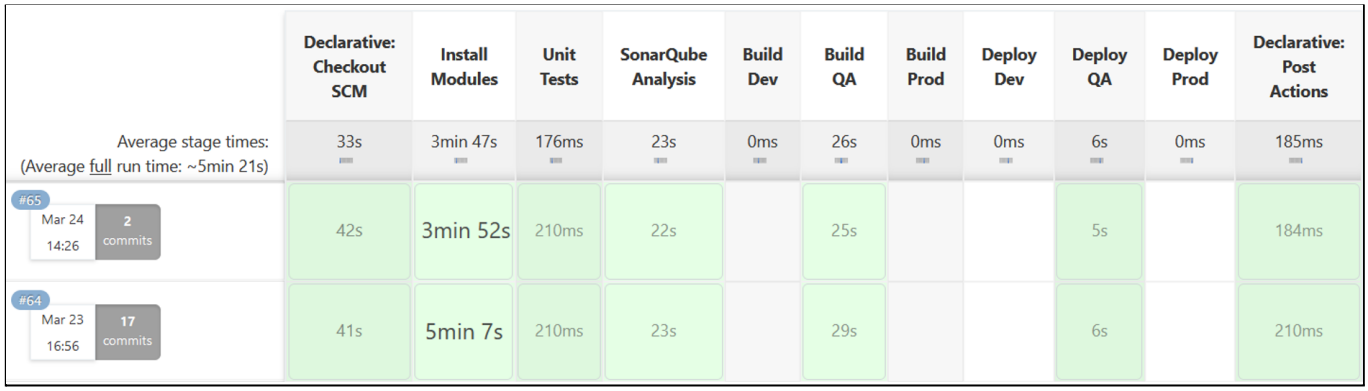


Figura 4: Métricas SonarQube proyecto Frontend después del formateo de código fuente y adopción de pre-commit por parte de los desarrolladores

Esta verificación del código se agregó como un *Stage* al pipeline de CI/CD del proyecto, el cual está implementado en Jenkins, tal como se puede identificar en la Figura 6 con el nombre de Lint-Syntax. Se debe resaltar que en este punto se configuró dicho *Stage* como no bloqueante, es decir, no influía en el flujo de CI/CD directamente debido a un acuerdo de equipo que se estableció, mientras que los desarrolladores se acostumbraban a los nuevos estándares de codificación a seguir; sin embargo, esta condición se eliminó posteriormente. También en la Figura 6, se aprecia una mejora sustancial en el tiempo que toma en ejecutarse el *Stage* denominado *Install Modules* tardando 5 segundos en promedio, en contraposición al tiempo que se muestra en la Figura 5 para el mismo *Stage,* en el que toma entre 4 y 5 minutos; reduciendo así el tiempo global de ejecución del pipeline. Este mejoramiento se alcanzó empleando algunos recursos técnicos que provee Docker, que es la herramienta utilizada para ejecutar todos los *Stages* del pipeline. Más específicamente, se tomó ventaja de los recursos de almacenamiento a través de volúmenes para contenedores Docker, con el objetivo de *cachear* las dependencias del proyecto, es decir, guardarlas tal que estas no tendrían que ser instaladas cada vez que se ejecuta el pipeline. Esto se traduce en un decremento significativo del tiempo que toma para los desarrolladores incorporar su código con otros desarrolladores, y por lo tanto, en un incremento de la eficiencia del proceso de integración continua.

Adicionalmente, comparando nuevamente las Figuras 5 y 6, se puede apreciar que se realizó una unificación de los *Stages Build* y *Deploy,* que inicialmente estaban separados por ambientes, pero que así agregaban una complejidad innecesaria a la representación visual del pipeline de CI/CD, y al código de configuración de dicho pipeline en el Jenkinsfile. Por lo tanto, se hizo una refactorización del código de dicho archivo de configuración, el cual está implementado siguiendo una sintaxis declarativa, pero añadiendo bloques de código en sintaxis *Scripted.*



	Declarative: Checkout SCM	Install Modules	Unit Tests	SonarQube Analysis	Build Dev	Build QA	Build Prod	Deploy Dev	Deploy QA	Deploy Prod	Declarative: Post Actions
Average stage times: (Average full run time: ~5min 21s)	33s	3min 47s	176ms	23s	0ms	26s	0ms	0ms	6s	0ms	185ms
#65 Mar 24 14:26 — 2 commits	42s	3min 52s	210ms	22s		25s			5s		184ms
#64 Mar 23 16:56 — 17 commits	41s	5min 7s	210ms	23s		29s			6s		210ms

Figura 5: Pipeline de CI/CD Frontend antes de agregar Lint-Syntax Stage y refactorizar el Jenkinsfile

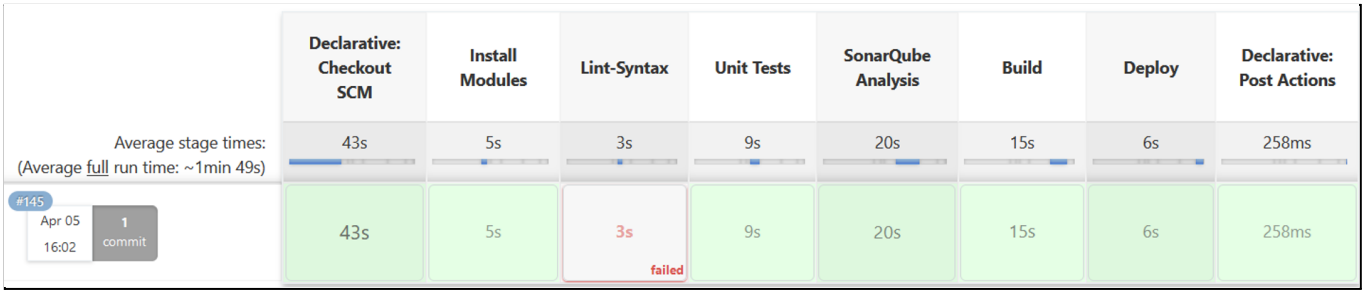


Figura 6: Pipeline de CI/CD Frontend después de agregar Lint-Syntax Stage y refactorizar el Jenkinsfile

Se debe hacer la salvedad de que, aunque se muestra un *Stage Unit Tests* en las Figuras 5 y 6, en ese punto no se estaban ejecutando dichas pruebas. No obstante, una vez que se empezaron a agregar pruebas unitarias al proyecto de Frontend por parte de los desarrolladores, y a mejorar las ya existentes y a adicionar las correspondientes en el proyecto de Backend, se configuró el *Stage Unit Tests* en el pipeline, y se llevó a cabo una reconfiguración del análisis de SonarQube; esto último con el objetivo de hacer un examen más preciso del código, y de agregar la cobertura de código (*Code Coverage*) de pruebas unitarias. Por ejemplo, en la Figura 7 se muestra el cambio en las métricas de SonarQube para el proyecto de Backend, que ya tenía pruebas unitarias implementadas, pero no contaba con la métrica del *Code Coverage*; mientras que en la Figura 8 se muestra dicha información para el proyecto de Frontend, donde no se tenían pruebas unitarias, y por lo tanto la cobertura de código evolucionó incrementalmente desde cero.

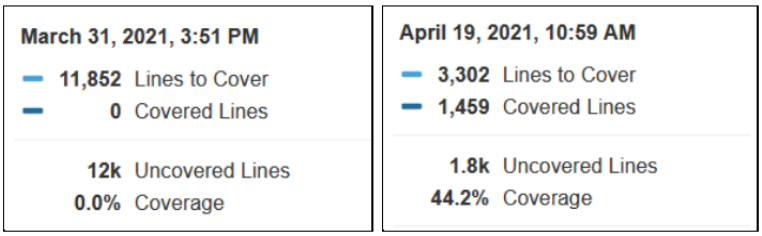


Figura 7: Cobertura de Código Backend

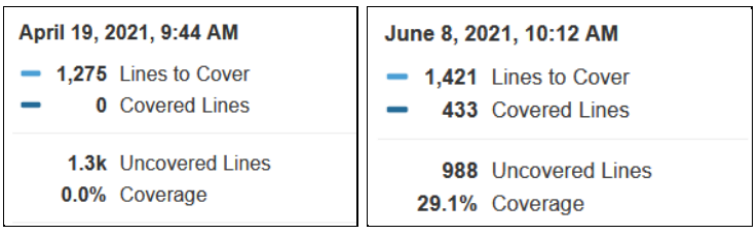


Figura 8: Evolución Cobertura de Código Frontend

Además de las pruebas unitarias que corresponden a la base de la denominada “Pirámide de Pruebas de Cohn” [8], también se adicionaron al proyecto pruebas automatizadas de orden

superior, más específicamente, pruebas de servicios o de API, y pruebas *end-to-end* de interfaz de usuario (UI por sus siglas en Inglés). Las primeras se implementaron utilizando la utilidad Newman de la herramienta Postman, y las segundas empleando el framework Cypress. El desarrollo de dichas pruebas fue realizado por el equipo de QA del proyecto, y en consecuencia, fueron añadidas al pipeline de CI/CD en cooperación con ellos. De este modo, se alcanzó un estado de madurez y solidez no presenciado hasta ese momento en el proyecto, en el cual el sistema contaba con pruebas automatizadas en todos los niveles. En consecuencia, fue posible evolucionar el estado del proyecto de entrega continua a despliegue continuo, donde no se requiere aprobación manual para desplegar al ambiente de producción, debido a que se confía plenamente en el conjunto de pruebas aplicadas en el pipeline. Las Figuras 9 y 10 muestran cómo quedaron estructurados finalmente dichos pipelines de CI/CD, después de las modificaciones y las mejoras llevadas a cabo.

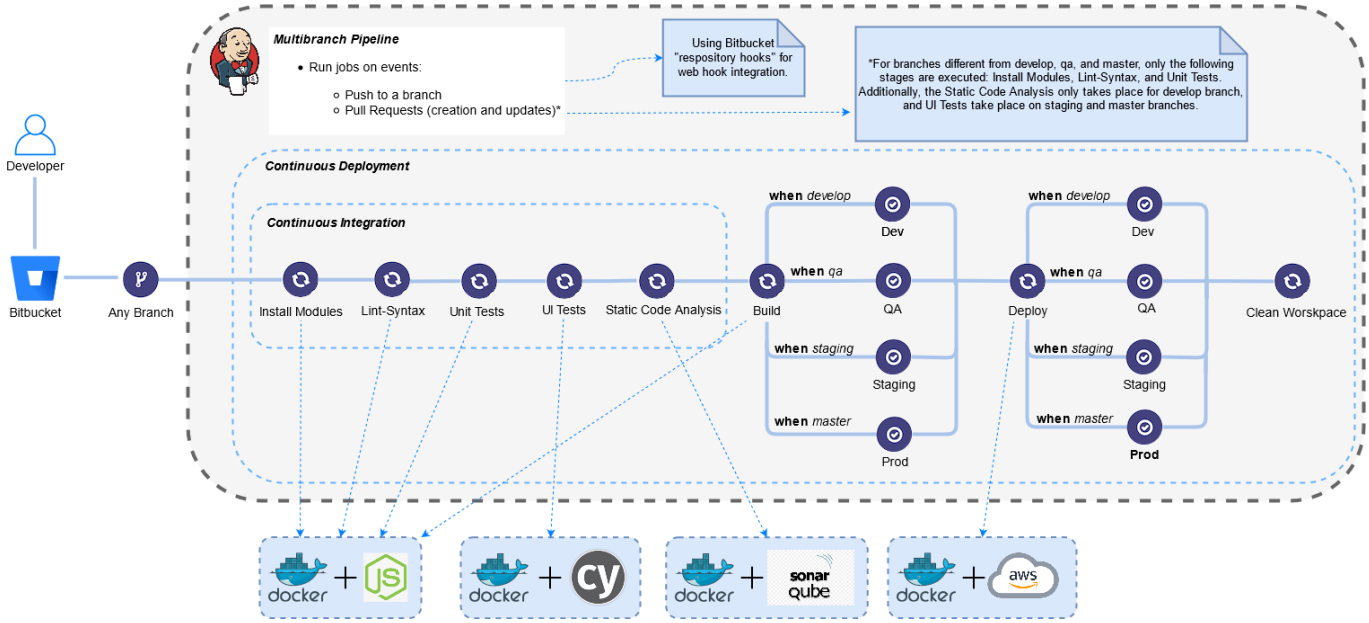


Figura 9: Pipeline de CI/CD Frontend

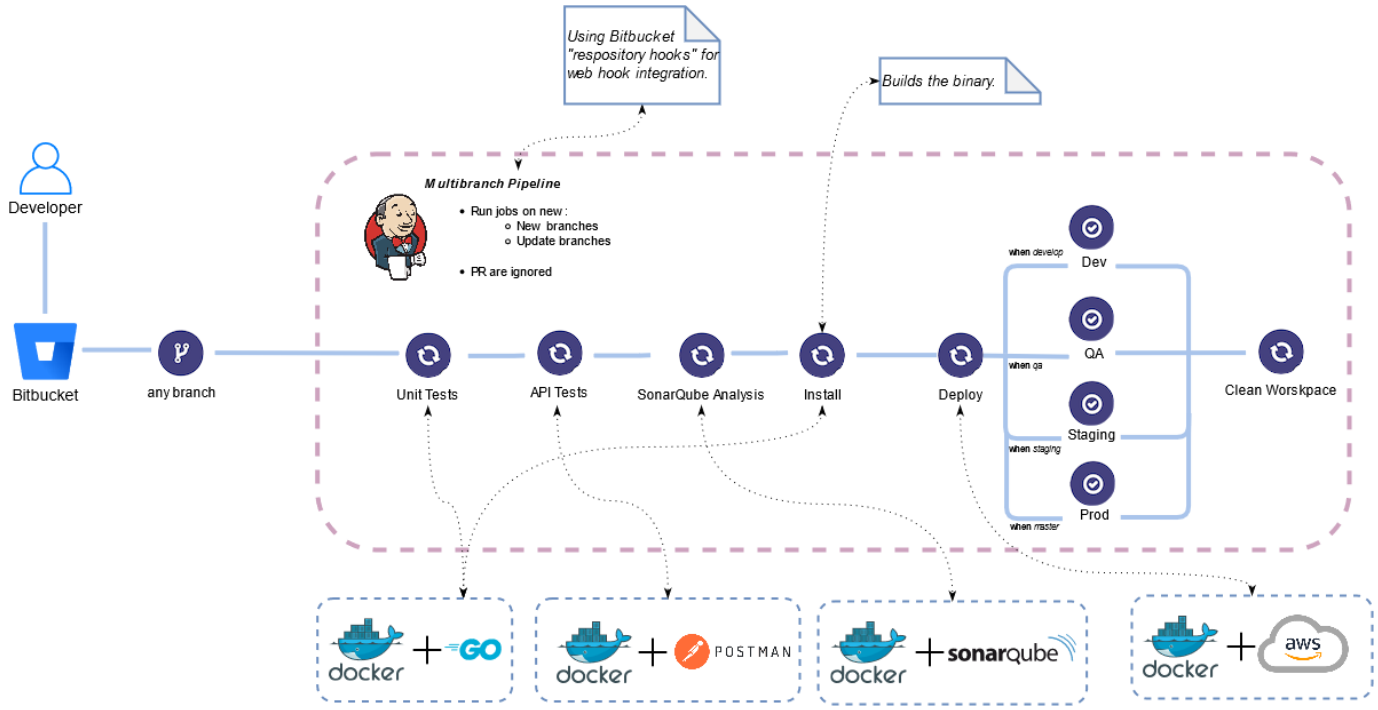


Figura 10: Pipeline de CI/CD Backend

Como se mencionó anteriormente, todas estas mejoras estuvieron determinadas, en gran medida, por el entendimiento obtenido producto de realizar el ejercicio de migración entre nubes. En la Figura 11, se muestra un esquemático de la arquitectura resultante después de replicar los componentes del proyecto que se muestran en la Figura 1. En este punto es importante resaltar que, durante la ejecución del ejercicio de migración, se identificó que la implementación del código de la API correspondiente al Backend, no es adecuada para ser desplegada en una arquitectura serverless compuesta de varias funciones pequeñas y granulares (tal como expone dicho paradigma). En efecto, el despliegue de este componente sobre AWS corresponde a una única "súper función" que recibe y procesa todas las peticiones; aunque quizá no fue la mejor decisión de diseño en la concepción inicial del proyecto, algunos integrantes que participaron en dichas etapas manifestaron que se tomó dicha decisión con el objetivo de explorar el paradigma de computación serverless (una práctica que es usual en proyectos de tipo *bench*). Así, al momento de intentar replicar dicho escenario sobre Microsoft Azure, empleando el servicio Azure Functions [17], se evidenció que se agregaría una complejidad innecesaria a la infraestructura, teniendo en cuenta las particularidades de dicho servicio en comparación al de AWS. Por consiguiente, se consideró desplegar la API siguiendo una estrategia de contenerización, utilizando Docker y sobre el servicio Azure App Service [18] (que es de categoría PaaS -Platform as a Service-) en conjunción con el servicio Azure Container Registry, tal como se ilustra en la Figura 11. De lo anterior, es de destacar que debió seguirse una estrategia de contenerización debido a que Go, el lenguaje en el que está escrito el Backend, no es directamente soportado por el servicio Azure App Service. Eventualmente, los demás

componentes involucrados (base de datos, monitoreo, entrega de estáticos para el Frontend, almacenamiento de credenciales y funciones auxiliares) sí pudieron replicarse en los servicios homólogos en ambas plataformas, como se planteó inicialmente en el marco teórico.

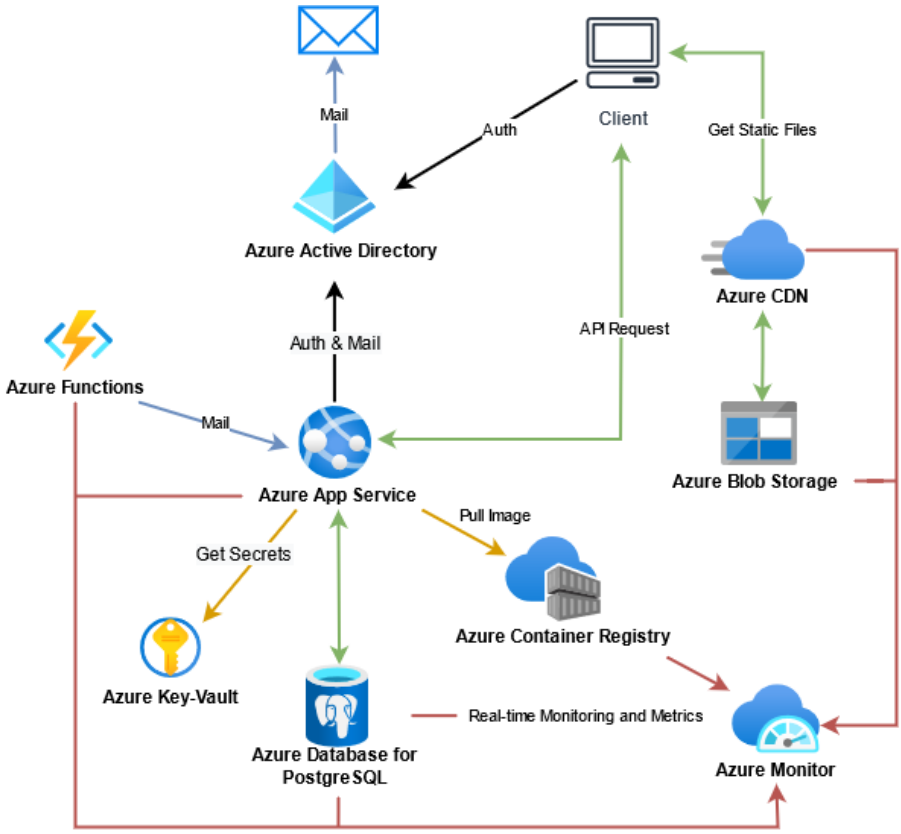


Figura 11: Componentes del proyecto EndavaCares sobre la plataforma Microsoft Azure

Sabiendo que el ejercicio de migración entre nubes respondía a una actividad cuyo objetivo era adquirir experiencia y explorar nuevas herramientas y conceptos, ya que el proyecto EndavaCares seguiría alojado en un ecosistema de AWS, se generó una entrada de documentación explicando, paso a paso, cómo se realiza el proceso de aprovisionamiento y despliegue de cada uno de los componentes: esta guía quedó almacenada en el sitio oficial de documentación del proyecto en la empresa. En las Figuras 12 y 13 se exponen evidencias de los componentes principales del sistema funcionando sobre servicios de la plataforma Microsoft Azure (para que el backend se ejecutara es necesario que existiera una conexión exitosa a la base de datos), y que después del ejercicio fueron retirados de la nube.

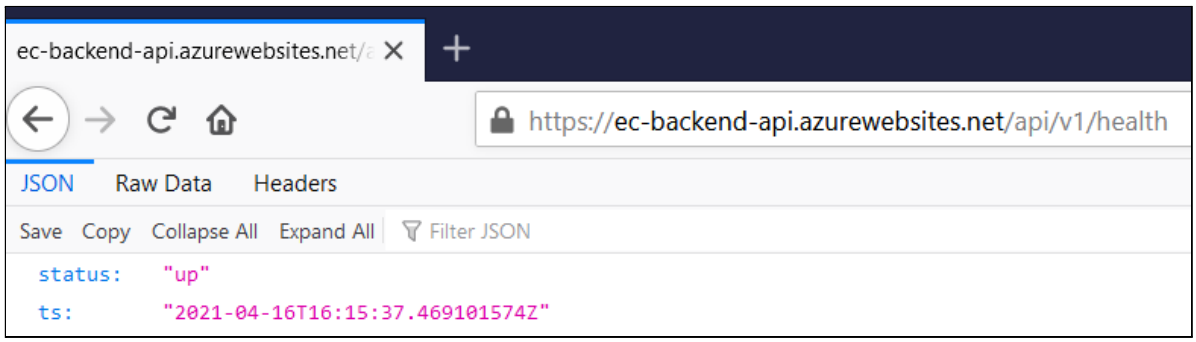


Figura 12: Backend Funcional sobre una Instancia de Azure App Service

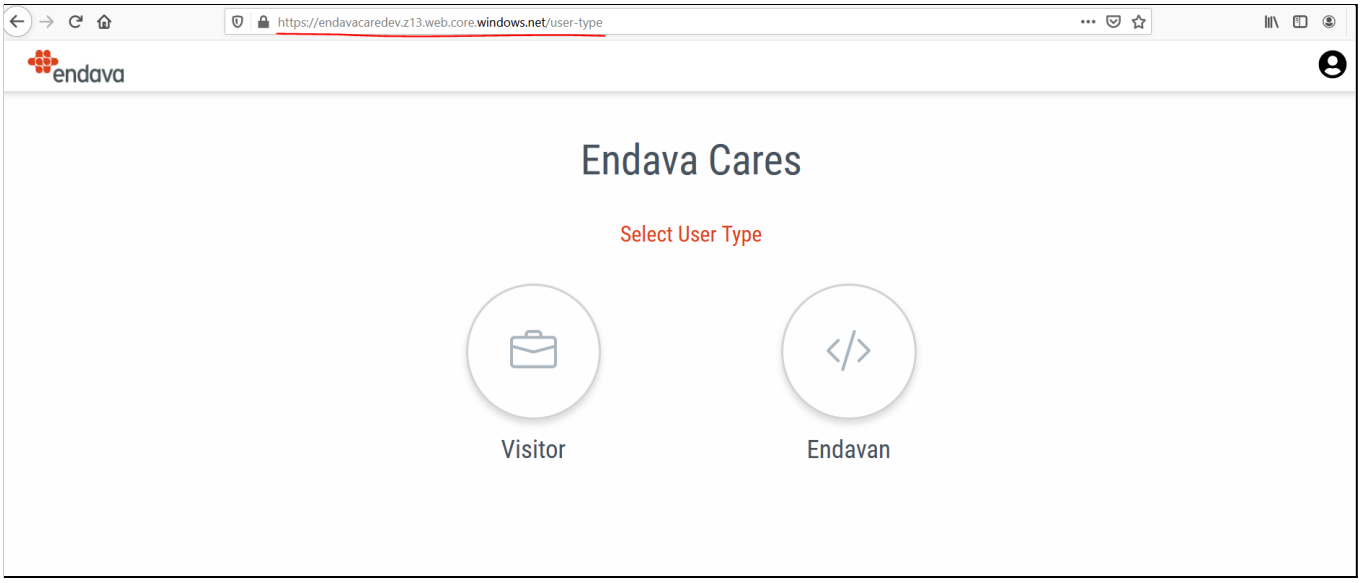


Figura 13: Frontend Funcional sobre una Instancia de Azure Blob Storage

Finalmente, teniendo en cuenta la persistencia del proyecto sobre un ecosistema de AWS, se realizó la implementación de la arquitectura mostrada en la Figura 1, siguiendo el paradigma de aprovisionamiento de infraestructura como código (IaC), y empleando la herramienta de uso libre Terraform para dicho objetivo. La Figura 14 muestra un esquema del funcionamiento de dicha herramienta: la infraestructura es expresada como código en un lenguaje declarativo simple, se leen los archivos de configuración y se provee un plan de ejecución, que luego es aplicado automáticamente sobre alguno de los proveedores de nube (en este caso AWS). Así, en la Figura 15 se presenta la salida generada como resultado del proceso de aprovisionamiento llevado a cabo (el cual es puesto en marcha sencillamente ejecutando el comando *terraform apply*). Allí se señala que se agregaron 44 recursos, los cuales corresponden a lo siguiente: la configuración de la red privada virtual en la nube que incluye subredes y grupos de seguridad, roles y políticas, la base de datos, máquinas virtuales necesarias, funciones lambda para el Backend y de soporte (triggers), y servicios de almacenamiento y entrega de archivos estáticos (Frontend). Sin embargo, la

característica más notable es el tiempo total empleado para realizar el despliegue y configuración de los recursos de manera automática: aproximadamente 7 minutos. Este tiempo representa una reducción sustancial con respecto a lo que tomaría a un ingeniero realizar este proceso manualmente, a través del portal web y la CLI del proveedor: un estimado de entre 7 y 8 horas de trabajo. En consecuencia, una vez que se tiene implementada la infraestructura como código, dicho proceso de aprovisionamiento y configuración se hace bastante eficiente y eficaz al emplear Terraform.

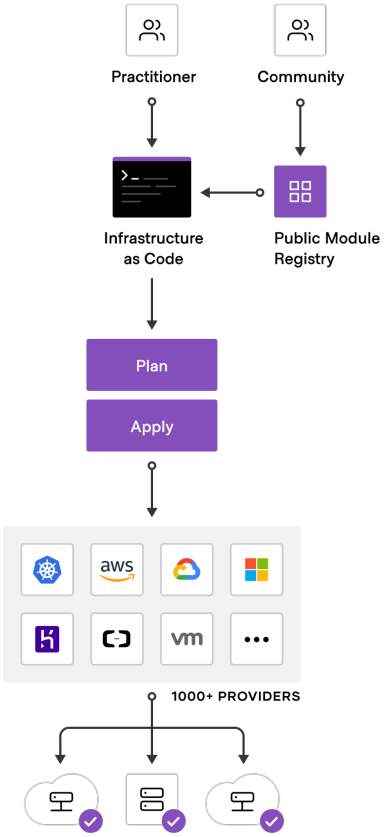


Figura 14: IaC usando Terraform [15]

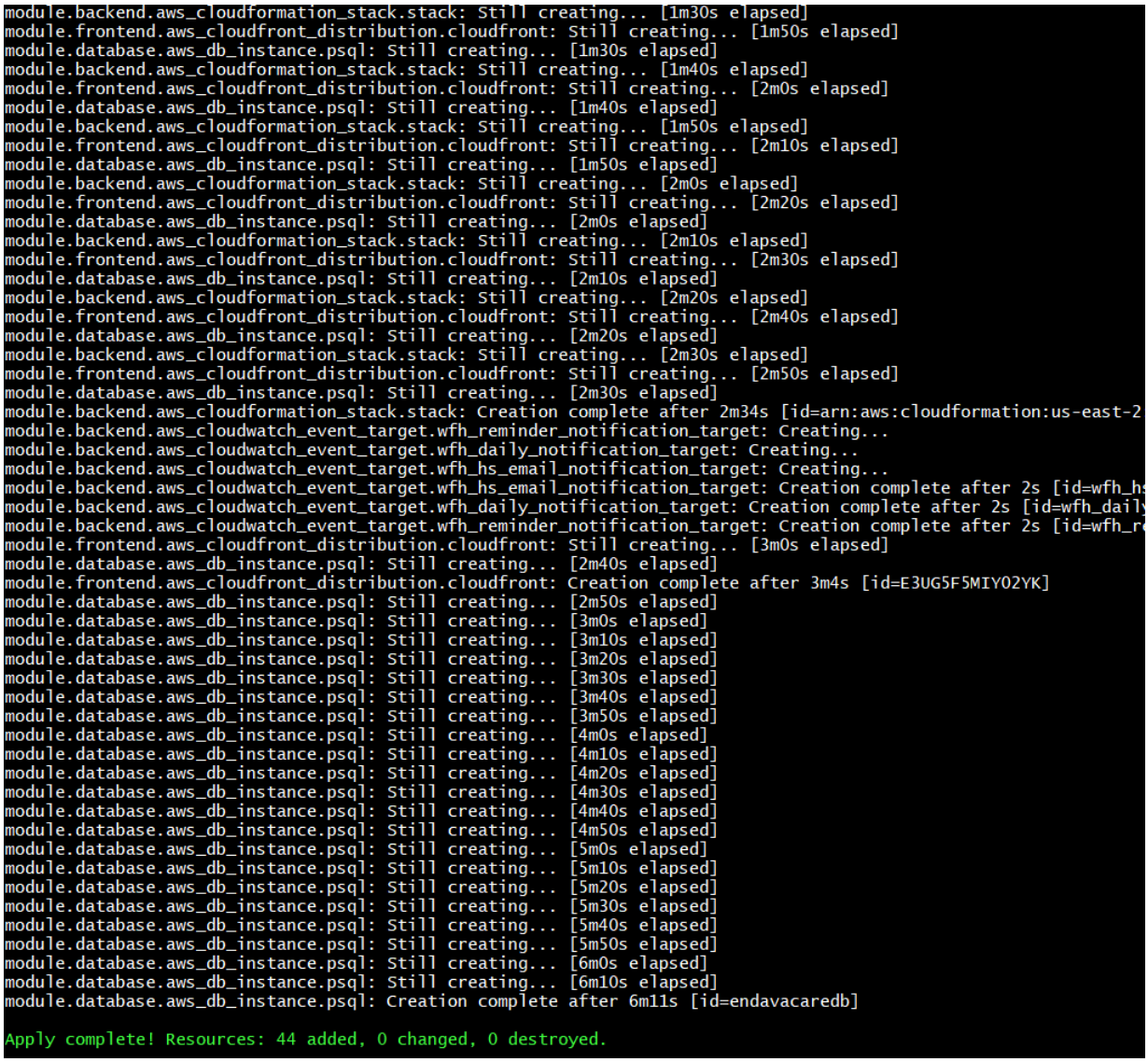


Figura 15: Aprovisionamiento de Recursos en la Plataforma AWS empleando Terraform

CONCLUSIONES

En este proceso de práctica académica se comprobó que, en aras de aplicar exitosamente las prácticas de la cultura DevOps en un proyecto de software, se requiere de una comunicación constante y asertiva entre los miembros del equipo; estos, además, deberán ser personas actitudinalmente dispuestas a adoptar dicha cultura progresivamente, a través de la consecución de acuerdos mutuos. En lo referente al componente técnico, la herramienta Jenkins permitió materializar el flujo de CI/CD a través de un pipeline, el cual a su vez se sirvió de las características de Docker para implementar los *stages* de manera fiable y eficiente utilizando contenedores. Por ejemplo, uno de los *stages* del pipeline

corresponde a la ejecución de un análisis de código estático, usando un contenedor con la herramienta SonarQube; así, fue posible obtener métricas de valor para el equipo y el proyecto (e.g., *code smells, tech debt, code coverage*), las cuales representaron datos útiles para la toma de decisiones, de cara a las acciones de mejoramiento. También, la flexibilidad de ejecución de aplicaciones con Docker, permitió agregar al pipeline pruebas automatizadas de servicios y de orden superior (pruebas de API y de interfaz de usuario), aumentando la calidad del software; esto se tradujo en un estado de robustez del proyecto tal que fue posible evolucionar desde la entrega continua al despliegue continuo.

Adicionalmente, se evidenció que utilizando Terraform se disminuyen significativamente los tiempos empleados en el aprovisionamiento de servicios en la nube, y se reducen las probabilidades de cometer errores en dicho proceso, en comparación con la implementación desde los portales web de las plataformas, y el uso de comandos a través de la CLI respectiva. El paradigma de IaC permitió automatizar eficientemente el proceso de despliegue y configuración de la infraestructura, lo cual es muy valioso para la cultura DevOps, donde la automatización de procesos es uno de los pilares fundamentales.

Finalmente, de la participación en el proyecto se concluye que, los proyectos tipo *bench* en las empresas de desarrollo de software, se presentan como oportunidades muy beneficiosas para explorar nuevas tecnologías y prácticas innovadoras, al tiempo que se genera un producto de valor para la empresa. Por ejemplo, en este caso se llevó a cabo un ejercicio de migración desde la plataforma AWS hacia Microsoft Azure, lo que permitió adquirir aptitudes en el paradigma de computación en la nube, y más específicamente en los servicios ofrecidos por dichas plataformas, así como una visión holística del proyecto para identificar puntos de mejora. Se puede aprender mucho mediante el ejercicio de comparación de los diferentes servicios a utilizar, en un contexto de software real, con el objetivo de replicar todos los componentes exitosamente. Con todos los retos que pueden emerger en este proceso, los conocimientos adquiridos son muy valiosos para el practicante o ingeniero, y para la empresa, de cara a futuros proyectos que involucren proveedores de nube específicos.

REFERENCIAS

[1] A. Goswami, "Increased bench size in IT: Strength or liability?", *HR Katha*, 2020. [Online]. Available: https://www.hrkatha.com/features/increased-bench-size-in-it-strength-or-liability/. [Accessed: 02- Jul- 2021].

[2] Resolución No. 666 de 2020. Colombia: Ministerio de Salud y Protección Social, 2020. [Online]. Available: https://www.minsalud.gov.co/Normatividad_Nuevo/Resoluci%C3%B3n%20No.%20666%20de%202020.pdf

[3] M. Pastorino, "Frontend vs Backend: What's The Difference?", *Pluralsight.com*, 2021. [Online]. Available: https://www.pluralsight.com/blog/software-development/front-end-vs-back-end. [Accessed: 02- Jul- 2021].

[4] "DevOps: Principles, Practices, and DevOps Engineer Role", AltexSoft, 2021. [Online]. Available: https://www.altexsoft.com/blog/engineering/devops-principles-practices-and-devops-engineer-role/ . [Accessed: 13- Jun- 2021].

[5] "AWS | Technology Radar | ThoughtWorks", Thoughtworks.com, 2021. [Online]. Available: https://www.thoughtworks.com/radar/platforms/aws . [Accessed: 13- Jun- 2021].

[6] "AZURE | Technology Radar | ThoughtWorks", Thoughtworks.com, 2021. [Online]. Available: https://www.thoughtworks.com/radar/platforms/azure . [Accessed: 13- Jun- 2021].

[7] S. PITTET, "Continuous integration vs. continuous delivery vs. continuous deployment", *Atlassian*, 2021. [Online]. Available: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment . [Accessed: 16- Jun- 2021].

[8] M. Cohn, "The Forgotten Layer of the Test Automation Pyramid", Mountain Goat Software, 2009. [Online]. Available: https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid . [Accessed: 13- Jun- 2021].

[9] H. Vocke, "The Practical Test Pyramid", martinfowler.com, 2018. [Online]. Available: https://martinfowler.com/articles/practical-test-pyramid.html . [Accessed: 13- Jun- 2021].

[10] IBM Education, "What is Docker?", *Ibm.com*, 2021. [Online]. Available: https://www.ibm.com/cloud/learn/docker . [Accessed: 16- Jun- 2021].

[11] "Empowering App Development for Developers | Docker", *Docker*, 2021. [Online]. Available: https://www.docker.com/ . [Accessed: 16- Jun- 2021].

[12] "Jenkins", *Jenkins*, 2021. [Online]. Available: https://www.jenkins.io/ . [Accessed: 16- Jun- 2021].

[13] "Code Quality and Code Security | SonarQube", *Sonarqube.org*, 2021. [Online]. Available: https://www.sonarqube.org/ . [Accessed: 16- Jun- 2021].

[14] "TERRAFORM | Technology Radar | ThoughtWorks", Thoughtworks.com, 2021. [Online]. Available: https://www.thoughtworks.com/radar/tools/terraform . [Accessed: 16- Jun- 2021].
[15] "Terraform by HashiCorp", *Terraform by HashiCorp*, 2021. [Online]. Available: https://www.terraform.io/ . [Accessed: 16- Jun- 2021].
[16] "Manifesto for Agile Software Development", Agilemanifesto.org, 2001. [Online]. Available: https://agilemanifesto.org/ . [Accessed: 13- Jun- 2021].
[17] "Azure Functions Serverless Compute | Microsoft Azure", *Azure.microsoft.com*, 2021. [Online]. Available: https://azure.microsoft.com/en-us/services/functions/ . [Accessed: 16- Jun- 2021].
[18] "App Service | Microsoft Azure", *Azure.microsoft.com*, 2021. [Online]. Available: https://azure.microsoft.com/en-us/services/app-service/ . [Accessed: 16- Jun- 2021].