UPN
UNIVERSIDAD PRIVADA DEL NORTE

FACULTAD DE INGENIERÍA

Carrera de Ingeniería de Sistemas Computacionales

“IMPLEMENTACIÓN DE AWS Y CI/CD PARA OPTIMIZAR EL PROCESO DE PROVISIONAMIENTO DE AGENTES DE CONSTRUCCIÓN EN EL ÁREA DE TI EN UNA EMPRESA DE LIMA 2024”

Trabajo de suficiencia profesional para optar al título profesional de:

Ingeniero de Sistemas Computacionales

Autor:

Fabrizzio Aron Aranda La Rosa

Asesor:

Mg. Josue Joel Rios Herrera

0000-0002-1157-0194

Lima - **Perú**

2025

Informe de Similitud

turnitin Página 2 de 123 - Descripción general de integridad Identificador de la entrega trn:oid:::1:3366381751

17% Similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para ca...

Fuentes principales

16% Fuentes de Internet
2% Publicaciones
12% Trabajos entregados (trabajos del estudiante)

Marcas de integridad

N.º de alertas de integridad para revisión

No se han detectado manipulaciones de texto sospechosas.

Los algoritmos de nuestro sistema analizan un documento en profundidad para buscar inconsistencias que permitirían distinguirlo de una entrega normal. Si advertimos algo extraño, lo marcamos como una alerta para que pueda revisarlo.

Una marca de alerta no es necesariamente un indicador de problemas. Sin embargo, recomendamos que preste atención y la revise.

Dedicatoria

El presente trabajo va dedicado a mis padres, ya que por su esfuerzo tengo el privilegio de poseer educación y comodidad, a ellos les debo todo lo que soy.

También va a dedicado a mi fiel compañero de aventuras Argos, cuya compañía incondicional ha sido reconfortante en el tiempo que duró la realización de este.

Agradecimiento

Agradezco a la empresa por permitirme realizar el presente trabajo con los datos de la implementación.

También a mi alma mater, por brindarme los conocimientos necesarios para poder realizar el presente trabajo.

Contenido

Índice de tablas

Índice de Figuras

Índice de ecuaciones

(El presente índice se fijará en función a la naturaleza del trabajo. Las ecuaciones se emplean habitualmente en investigaciones en ingeniería)

RESUMEN EJECUTIVO

El presente trabajo se desarrolló en una empresa de Lima que tenía la problemática de tener agentes de Teamcity encendidos todo el día todos los días que brindaban colas de construcción largas para sus proyectos, así como también, costos de operación tanto en el hecho de que estén disponibles 24/7 así como también en el tiempo que se utilizaba para actualizar el software en cada una de las maquinas existentes. Es por ello por lo que se realiza una migración de los agentes en la nube, de Azure hacia AWS, utilizando herramientas como Terraform y complementándolo con instancias EC2 de AWS y Teamcity para conseguir agentes de construcción a demanda, con el fin de aligerar las colas de construcción.

Con esta implementación, se redujo los costos generados por los agentes de Teamcity debido a que estos se crean a demanda y se destruyen después de 15 minutos de inactividad; gracias a esto, nos permitió con ese ahorro obtener un mayor número de agentes disponibles para los proyectos, agilizando así la construcción de los artefactos de sus diferentes proyectos.

CAPÍTULO I. INTRODUCCIÓN

1.1. Contexto de la empresa

El presente trabajo con el título "Implementación de AWS y CI/CD para optimizar el proceso de provisionamiento de agentes de construcción en el área de TI en una empresa de Lima 2024" describe como fue el proceso de implementación de nuevos agentes de Teamcity para la mejora en la consistencia del mantenimiento de los agentes, la disponibilidad y resiliencia de estos, así como también la mejora en su costo de mantenimiento.

Previamente, se mantenían la cantidad de 3 agentes de Teamcity en máquinas virtuales alojadas en la nube de Azure, encendidas las 24 horas del día los 7 días de la semana y únicamente puestas fuera de línea para labores de mantenimiento, tales como actualización de software o parches de seguridad, los cuales se tenían que realizar manualmente mediante acceso SSH a la instancia, por lo que, realizar una actualización y/o aplicar un parche de seguridad se consideraba un proceso tedioso, repetitivo y susceptible al error humano. Otro punto para tener en consideración es el hecho que al tener tan pocos agentes disponibles, las colas de espera para la utilización de estos se podían volver largas, dado que existían algunos trabajos de construcción que demoraban horas y demandaban que sus dependencias también sean ejecutadas para que el pipeline pueda considerarse como terminado.

1.2. La Empresa

La empresa fue fundada en Perú en el año 2012, registrada como una sociedad anónima cerrada.

1.3. Misión

Ser reconocida como la principal plataforma de soluciones de marketing, tecnología y ventas que impulsa el crecimiento de las principales marcas de seguros en Estados Unidos y otros mercados, maximizando la satisfacción y el retorno de inversión de sus clientes mediante servicios innovadores y de alta calidad.

1.4. Visión

Convertirse en el socio estratégico más confiable y líder en la transformación digital de la industria de seguros, revolucionando la forma en que las marcas llegan, interactúan y fidelizan a los consumidores, y estableciendo nuevos estándares en marketing, tecnología y ventas en el sector asegurador.

1.5. Servicios

Presenta 3 principales servicios: marketing, ventas y tecnología, los cuales se detallan a continuación.

1.5.1. *Servicio De Marketing*

Ofrece una completa cartera de soluciones de orientadas a las empresas de seguros, con asistencia en todas las fases y medios de promoción. Esto incluye estrategias de SEO para mejorar los resultados de búsqueda orgánica y atraer a más visitantes relevantes al sitio web, campañas de publicidad basadas en datos para aumentar el retorno de la inversión y marketing en redes sociales mejorado para llegar de forma eficaz y oportuna al público adecuado.

Además, el objetivo del diseño y la construcción de sitios web no es solo la funcionalidad, sino también mejorar las tasas de conversión; las campañas de correo electrónico personalizadas contribuyen a mejorar el compromiso de los clientes y a aumentar las tasas de apertura y conversión.

1.5.2. *Servicio De Tecnología*

Se encarga de desarrollar soluciones innovadoras y basadas en datos que responden a las necesidades de sus clientes y socios comerciales. Dado que el proceso de compra de seguros está en constante evolución, el área tecnológica ayuda a sus socios a mantenerse a la vanguardia mediante la creación e implementación de herramientas tecnológicas avanzadas. Dentro de sus responsabilidades, desarrolla sitios web optimizados y adaptables a dispositivos móviles, con el objetivo de maximizar la eficiencia y aumentar la cantidad de clientes potenciales cualificados.

En el ámbito de la ciencia de datos, la empresa elabora algoritmos de aprendizaje automático para la optimización de ofertas publicitarias digitales, la compra y el enrutamiento de clientes potenciales, permitiendo maximizar el ROI (Return Of Investment) en ventas y marketing.

Además, en el desarrollo de productos, construye y mantiene aplicaciones que se integran con los sistemas de sus socios, utilizando análisis predictivo para mejorar la eficiencia operativa. También, se dedica a la publicación de contenido digital, creando aplicaciones web que facilitan la publicación de contenido en sitios internos, de clientes y medios, lo que contribuye al incremento de leads cualificados.

1.5.3. *Servicio De Ventas*

El servicio de ventas proporcionar un apoyo especializado para educar a los clientes y

devolverles el poder de decisión en el proceso de compra. Esta área busca no solo cerrar ventas, sino asegurar una experiencia positiva y satisfactoria para el cliente durante todo el proceso. Entre sus principales funciones, el área de ventas incluye la asistencia en la obtención de licencias de seguros. La empresa apoya a los profesionales de ventas elegibles en la obtención de licencias de seguros de vida y/o salud, lo cual es fundamental para la capacitación y legalidad del equipo de ventas.

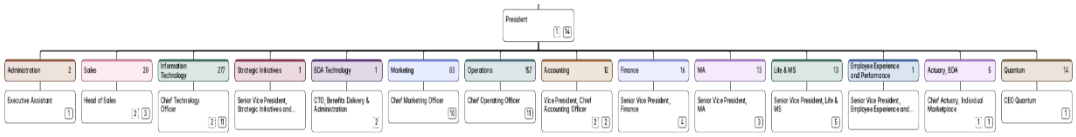
El área también se encarga del compromiso posterior a la venta, ofreciendo un soporte especializado para garantizar que las solicitudes de pólizas se completen correctamente y que las pólizas se emitan con éxito. Asimismo, el equipo de gestión de casos está dedicado a resolver cualquier problema que surja durante el procesamiento de las solicitudes, proporcionando un soporte personalizado al cliente potencial asegurado.

1.6. Organigrama de la empresa

La empresa está encabezada por el presidente y bajo el cual se dividen las áreas mostradas en la Figura 1.

Figura 1

Organigrama general de la empresa



Nota: Datos extraídos de Pingboard

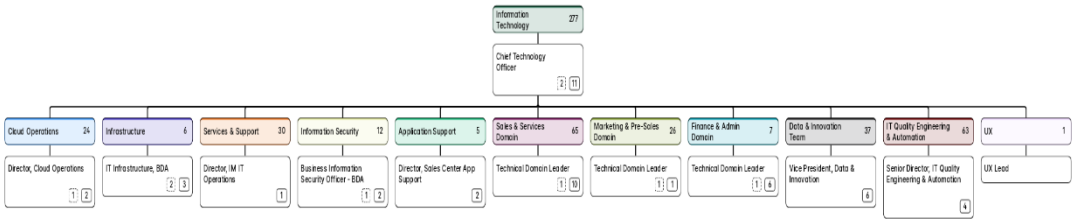
A continuación, se procede a detallar las áreas subsecuentes relacionadas al área en donde realizo mis funciones profesionales.

1.6.1. Information Technologies

Área responsable del uso que la organización hace de los sistemas y dispositivos informáticos para acceder a la información, procesarla y elaborar informes. Esta subdivido como se muestra en la Figura 2.

Figura 2

Organigrama del área de Information Technologies



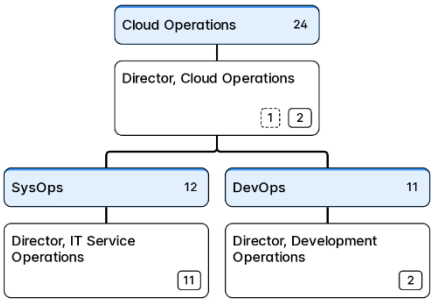
Nota: Datos extraídos de Pingboard

1.6.2. Cloud Operations

Como parte de la subdivisión, se encuentra el área de Cloud Operations, la cual se encarga de gestionar la entrega, el ajuste, la optimización y el rendimiento de las cargas de trabajo y los servicios de TI que se ejecutan en AWS y Azure. Su división se muestra en la Figura 3.

Figura 3

Organigrama del área Cloud Operations



Nota: Datos extraídos de Pingboard

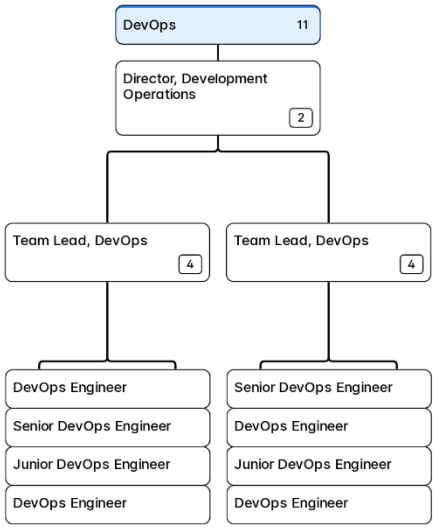
1.6.3. DevOps

Encargada de proveer a los equipos informáticos las herramientas, estrategias, normas y procesos que les faciliten la creación, implantación y supervisión de sus aplicaciones.

Como DevOps Engineer, me encuentro en esta área, la cual se encuentra divida como se puede apreciar en la Figura 4.

Figura 4

Organigrama del área DevOps



Nota: Datos extraídos de Pingboard

CAPÍTULO II. MARCO TEÓRICO

2.1 Antecedentes

2.1.1 Nacionales

Ofracio y Rey (2024), en su investigación "Diseño de infraestructura como servicio (IaaS) como solución para la entrega de servicios de TI para una empresa de seguros del mercado peruano" se enfocaron en resolver la ineficiencia existente con respecto a los costos y tiempos de entrega de los recursos de cómputo al querer implementar nuevos servicios de TI. Para lograrlo, diseñaron procesos identificando las actividades requeridas para la migración exitosa a la nube a través del modelo de Cloud Computing, tomando en consideración las recomendaciones de la ISO 27017 para el proveedor de servicios. Se utilizó la metodología Scrum. Como resultado se creó la landing zone y configuración de red de manera exitosa, además de identificar a AWS como líder en los cuadrantes de Gartner en los últimos 10 años, tomándolo como primera opción de proveedor de nube para realizar la migración. Se concluyó que la infraestructura como servicio (IaaS) debe ser considerada como un primer paso en la adopción de una arquitectura en la nube. Se recomienda considerar otras opciones como PaaS o SaaS, así como también considerar el uso de recursos en la nube como plan de contingencia.

Felices et al. (2024) en su investigación "Integración de servicios de AWS enfocado al uso de DevSecOps para mejorar el rendimiento y la seguridad en proyectos de Data & Anaytics" se enfrentan a la problemática de mejorar la calidad de gestión, centralización y gobernanza de los datos, dado que, en sus propias palabras,

presentan un entorno poco consistente, además de no presentar una solución capaz de adaptarse a diferentes problemáticas internas o de sus clientes. Sus objetivos fueron implementar una solución tecnológica modernas utilizando AWS y la metodología DevSecOps, así como también reducir el tiempo de despliegue de data lakes, mejorar la preparación de datos mediante herramientas de AWS y reducir las vulnerabilidades en el código. Utilizaron la metodología Scrum y Kanban con el fin de realizar la migración exitosa de una base de datos SQL hacia RedShift utilizando Cloudformation como medio para crear la infraestructura. Como resultados, mejoraron la consistencia operativa y redujeron errores manuales debido al uso de Cloudformation, además de hacer que la infraestructura sea reproducible y uniforme. También redujeron el tiempo de implementación de datos en 2 meses, así como también redujeron las vulnerabilidades en un 30% y obtuvieron una mejora en el suministro de datos del 40%. Recomiendan enfatizar más en la seguridad utilizando los diferentes servicios que ofrece AWS como AWS KMS, AWS WAF y AWS IAM, además de seguir con una evaluación continua de las herramientas y servicios ofrecidos por los diferentes proveedores de nube.

Llauce (2020) en su investigación “Implementación de una arquitectura de computación en la nube (cloud computing) diseñada para escalabilidad automática y alta disponibilidad basada en la plataforma de amazon web services (AWS) en la universidad de Lambayeque” nos menciona las dificultades que posee la infraestructura para escalar y manejar grandes cantidades de peticiones simultaneas, ocasionando caídas y fallas en los servidores y en la plataforma de la universidad. Su objetivo es implementar una arquitectura de computación en la nube diseñada para

escalabilidad automática, la optimización de costos y la alta disponibilidad basada en la plataforma AWS. Para ello, utilizó la metodología SCRUM y los servicios de AWS para diseñar una arquitectura robusta y escalable. Como resultados logró que la plataforma soporte un promedio de 80 usuarios de manera concurrente, además de conseguir un tiempo de recuperación del sistema promedio del 0.72 minutos por día. Recomienda que se cuente con personal capacitado para poder monitorear correctamente los servicios que se han implementado, así como también gestionar los accesos a los servicios mediante el servicio de IAM de AWS. Además, señala que el usuario no debe de preocuparse por el mantenimiento ni la infraestructura dado que es cubierto por AWS mediante el SLA.

Jara (2021) en su tesis “Implementación de una plataforma de virtualización usando Amazon Web Services, para soportar las aplicaciones de la empresa Programate S.A.C” presentan como desafío un problema en la virtualización de la plataforma de la empresa, dado que conlleva altos costos de mantenimiento el poseer y actualizar servidores propios con el fin de alojar sus aplicaciones, siendo estas cambiantes constantemente por el rápido avance de la tecnología. Tiene como objetivo determinar en qué medida la implementación de una plataforma de virtualización en AWS mejora el soporte de las aplicaciones de la empresa. Su investigación es de tipo aplicada y explicativa. Utilizaron la metodología Scrum y los servicios de AWS. Como resultados, se aprecia que debido a varios aspectos de medición como rapidez, seguridad y compatibilidad con las aplicaciones se considera mas eficiente utilizar la plataforma en AWS. Recomiendan poner en práctica el uso de la plataforma y activar alertas para monitorear las incidencias en caso de presentarse.

Rojas (2020) en la investigación “Proyecto de implementación de buenas prácticas de administración de recursos alojados en Amazon Web Services para controlar el consumo mensual” se enfrenta a la problemática de un sistema con un consumo mensual de aproximadamente 4 mil dólares, sumado a las alertas de bajo rendimiento y seguridad, asi como tambien cero tolerancias a fallos y limitaciones en el servicio. Sus objetivos fueron realizar la implementación de las buenas prácticas de gestión de recursos alojados en el entorno de nube del proveedor Amazon Web Services del cliente, reducir los costos mensuales de la plataforma a través de la optimización de recursos de cómputo, así como también optimizar la seguridad y rendimiento de los sistemas alojados en la plataforma de nube AWS. Se definió un plan de trabajo de 6 meses. Se identificaron los sistemas, regiones en donde estaban alojados y sus problemas respectivos. Mediante el uso de buenas prácticas referentes a la administración se obtuvo como resultados la disminución del costo mensual de nube en un 85% manteniendo la misma cantidad de sistemas. Además, se rediseñó la red e implementaron el servicio WAF para proteger las aplicaciones, permitiéndole además mejorar el rendimiento, identificar ataques y tomar acciones correctivas. Se recomienda conocer sobre las capas del modelo OSI y los diferentes tipos de ataques informáticos con el fin de incrementar la seguridad de los sistemas.

2.1.2 Internacionales

Garcia (2022) en la investigación “Propuesta de implementación de un proceso de despliegue automatizado, previo pruebas y análisis de código automatizado” nos indica las problemáticas de la metodología de desarrollo tradicionales, tales como inconsistencia entre los ambientes de desarrollo y

producción, limitaciones para tiempos de entrega y dificultades para administrar múltiples configuraciones y versiones de aplicaciones. Tuvo como objetivo elaborar una propuesta de implementación de un proceso de pruebas, análisis y despliegue automatizado de software mediante el uso de herramientas de integración, implementación y distribución continua con el fin de mejorar la calidad código y agilizar la entrega de funcionalidades en el departamento de TI de la PUCESE. Se utilizó la metodología SCRUM, flujo de trabajo GitFlow y las herramientas Sonarqube y Cypress. Sus resultados fueron la propuesta inicial del pipeline de integración y despliegue, además de reducir el tiempo de despligue y los errores. Se recomienda migrar a entornos en la nube, con los principales proveedores de la nube como opciones, aunque sin mencionar a alguno en específico.

Puolitaival (2024) en su investigación “Enhancing development efficiency with DevSecOps template: a design science approach for IaC and CI/CD implementation for AWS” se enfrenta a la falta conocimientos en las organizaciones con respecto a la digitalización y las tecnologías emergentes tales como inteligencia artificial. Tuvo como objetivos poder crear una plantilla del proyecto mediante infraestructura como código y automatización mediante CI/CD aplicando las mejores prácticas de desarrollo para facilitar el despliegue en la nube proyectos de software pequeños y medianos. Se aplicó la metodología de Ciencia de Diseño (Desing Science), la cual, según menciona Puolitaival (2024), consiste en desarrollar una teoría que luego se utiliza para diseñar y construir un artefacto en la práctica. Como resultado se determinó que se puede utilizar una plantilla que utilice Terraform y CI/CD pipelines mediante una implementación modular utilizando parámetros de

entrada, además se menciona que estandarizar los módulos provee a la infraestructura consistencia y buenas prácticas en el ámbito de seguridad. Recomienda iterar en mejorar la eficiencia de la plantilla con el tiempo para mejorar la eficiencia del desarrollo e investigar acerca de las mejores prácticas de código para diferentes tecnologías como React o Spring Boot.

Juconsa (2022) en su investigación "Platform for deploying a highly available, secure and scalable web hosting architecture to the AWS cloud with Terraform" busca resolver la problemática de la complejidad de crear recursos y/o servicios en la nube de manera manual, muchas veces involucrando a individuos con poco o nulo conocimiento acerca de la computación en la nube. Utiliza la metodología Agile, bajo el maco de trabajo Kanban. Su objetivo fue crear una herramienta que permita desplegar recursos en la nube de AWS utilizando Terraform como gestor y mejorar el trabajo de equipo utilizando GitHub como repositorio de control de versiones. Como resultado, se redujo los tiempos de creación promedio de recursos de entre el rango de 2 a 4 horas a un rango de 25 a 30 minutos. Recomienda implementar AWS Shield en su plataforma y automatizar el despliegue de las versiones de su proyecto mediante un pipeline de integración continua usando una herramienta como Jenkins.

Martinez (2022) en su investigación "CI/CD en infraestructura como código (IaC) Caso Practico en Amazon Web Services (AWS)" tuvo como objetivo encontrar una forma colaborativa de poder trabajar en la infraestructura de la empresa, de manera versionable e implementar un pipeline de integración continua en AWS. Utilizo la metodología SCRUM, realizando un POC con herramientas open-source tales como Jenkins, Git, Linux y Terraform, además de utilizar AWS. Como resultado

se implementó buenas prácticas de software a la infraestructura como código, tales como análisis de código, test unitarios y de integración, para después poder desplegar la infraestructura de AWS de manera confiable. Recomendaciones: Evaluar la implementación de un despliegue continuo en la empresa, ejecutar restauraciones automatizadas si es que fallan las pruebas unitarias o de integración.

Cîrlan (2024) en su investigación "Mining for Cost Awareness in Cloud Computing: A Study of AWS CloudFormation and Developer Practices" identifica como problemática la presencia de los costos ocultos para los desarrolladores al momento de crear infraestructura como código utilizando Terraform o CloudFormation en AWS, tales como costo de transferencia de datos o almacenamiento. Su objetivo es investigar que tan consientes están los desarrolladores de los costos cuando utilizan CloudFormation y compáralo con el uso de Terraform con el fin de mejorar la comprensión de las buenas prácticas relacionada con los costes en las herramientas de Infraestructura como Código (IaC). Se utiliza un minado de datos en repositorios públicos de GitHub conteniendo archivos de Terraform y/o Cloudformation, luego filtran los commits en base a palabras clave como bill, cheap, cost, siendo estos datos validados por 3 investigadores. Como resultados, se identificaron 826 commits de 446 repositorios conteniendo términos relacionados a costos, de los cuales se analizaron 262 commits de 205 repositorios. Se identifica que existe un interés en la concientización sobre los costes y el ahorro, con los desarrolladores realizando activamente modificaciones para optimizar los gastos. Además, identificaron que los desarrolladores que usan CloudFormation se enfocaron mas en estrategias financieras consientes y ajustes para seguimiento de costes, mientras que los que usan Terraform

se enfocan mas en las optimizaciones de recursos, como actualizaciones en la configuración de las instancias o el almacenamiento. Se recomienda ampliar la investigación considerando archivos JSON y Pull Request.

2.2 Definición de términos

2.2.1 AWS

Amazon Web Services (AWS) es la plataforma en la nube más utilizada y completa a nivel global, ofreciendo más de 200 servicios que incluyen desde infraestructura como cómputo, almacenamiento y bases de datos, hasta tecnologías avanzadas como IA, aprendizaje automático, análisis de datos y IoT. Esto facilita la migración de aplicaciones a la nube, haciéndola más rápida, sencilla y económica, y permite desarrollar prácticamente cualquier proyecto.

Diseñada para ser el entorno en la nube más seguro y adaptable disponible hoy, AWS integra una amplia gama de herramientas de seguridad con más de 300 funciones y servicios dedicados a la seguridad, conformidad y gobernanza, y cumple con 143 certificaciones y normas de seguridad. (Amazon Web Services, 2024).

2.2.2 Azure

Es la plataforma en la nube de Microsoft, la cual ofrece una amplia gama de servicios de computación en la nube que abarcan desde el desarrollo y la gestión de aplicaciones hasta el análisis de datos y la inteligencia artificial. Destaca por su solidez y adaptabilidad, al estar diseñada para soportar cualquier tipo de carga de trabajo y proporcionar a los desarrolladores herramientas integradas, plantillas y servicios de administración, facilitando así la creación y gestión de aplicaciones empresariales tanto para dispositivos móviles como para la web. (Microsoft Azure, 2024)

Además, cuenta con soluciones de nube híbrida que permiten gestionar entornos de TI existentes mediante conexiones privadas seguras, almacenamiento y bases de datos, todo ello con un enfoque en proteger la privacidad y los datos de los usuarios.

2.2.3 CI/CD Pipeline

Un CI/CD pipeline es un conjunto de pasos que optimiza el proceso de entrega de software (Gitlab, 2024). Mediante un enfoque DevOps o de ingeniería de fiabilidad del sitio, el pipeline de CI/CD optimiza el desarrollo de aplicaciones a través de la supervisión y la automatización, lo cual lo vuelve especialmente útil en la integración y las pruebas continuas, que a menudo son complejas, demandan mucho tiempo y requieren la creación de componentes temporales que simulan partes del sistema que no están disponibles o no están completamente implementadas.

Los pipelines automatizados ayudan a prevenir errores derivados de procesos manuales, facilitan iteraciones rápidas del producto y proporcionan datos consistentes a lo largo del ciclo de desarrollo.

Beneficios de usar pipelines automatizados

- Simplifica la construcción y las pruebas
- Mejora la calidad y la coherencia del código
- Agiliza la comunicación
- Automatiza gran parte del proceso de entrega de software
- Fomenta una retroalimentación más rápida
- Aumenta la visibilidad del producto

- Permite corregir rápidamente los errores manuales
- Reduce los costes de mano de obra
- Acelera el ciclo de vida del desarrollo
- Facilita un rápido bucle de retroalimentación entre ingenieros y clientes
- Las pruebas automatizadas ahorran dinero y minimizan los problemas para los usuarios finales

2.2.4 Build

Según Aranda (2023) un build es el resultado de un proceso de compilación en el que se toman los archivos de código fuente y se generan los archivos binarios necesarios para que el software funcione en un sistema operativo específico. Es un hito importante en el desarrollo de software, ya que marca el paso de un producto en desarrollo a un producto listo para ser probado y distribuido.

Se puede entender por "build" como la versión concreta del sofware en un momento dado y su creación puede automatizarse con herramientas como Make, Gradle, Ant o Maven, entre otras, para garantizar la repetibilidad y confiabilidad del proceso. Esta automatización es fundamental para mantener la eficiencia y calidad en el desarrollo.

2.2.5 Teamcity

Es una herramienta de Integración Continua y Despliegue Continuo (CI/CD) creada por JetBrains. Diseñada para optimizar el proceso de desarrollo de software, automatizando tareas como la construcción, prueba e implementación de cambios en el código. Proporciona una plataforma centralizada donde los desarrolladores pueden

gestionar compilaciones, ejecutar pruebas automatizadas y monitorear la salud general de sus proyectos en tiempo real. Su flexibilidad la hace adecuada tanto para proyectos pequeños de startups como para sistemas empresariales de gran escala.

Además de ser una herramienta de CI/CD, se le puede considerar como una solución completa que cubre todo el ciclo de vida del desarrollo de software, desde el compromiso del código hasta la implementación.

De acuerdo con StarAgile (2024) las características de Teamcity son:

- Interfaz de Usuario Intuitiva: Ofrece una interfaz fácil de usar que facilita la configuración, administración y visualización de las tuberías de CI/CD. Su panel de administración proporciona una visión general completa de la salud del proyecto, incluido el estado de las compilaciones, los resultados de las pruebas y las tareas pendientes.
- Capacidades de Integración Versátiles: Es compatible con una amplia gama de herramientas y plataformas; además, admite varios sistemas de control de versiones, herramientas de compilación, marcos de prueba y servicios de notificación, lo que garantiza una integración perfecta en los flujos de trabajo existentes.
- Configuraciones de compilación personalizables: Permite a los equipos adaptar las configuraciones de compilación a sus requisitos específicos. Desde scripts simples hasta tuberías complejas que involucran múltiples dependencias
- Monitoreo y Reportes de Compilación en Tiempo Real: Permite a los equipos seguir las compilaciones en tiempo real, ofreciendo registros detallados e

informes de progreso que están disponibles de inmediato.

- Escalabilidad y Alta Disponibilidad: Diseñado para crecer con las necesidades de su proyecto, permite gestionar múltiples proyectos, crear configuraciones personalizadas, distribuir las cargas de compilación entre varios agentes y asegurar una alta disponibilidad mediante una configuración con múltiples nodos.
- Amplia Personalización y Plugins: La plataforma es muy flexible, con una amplia gama de Plugins que extienden sus capacidades.

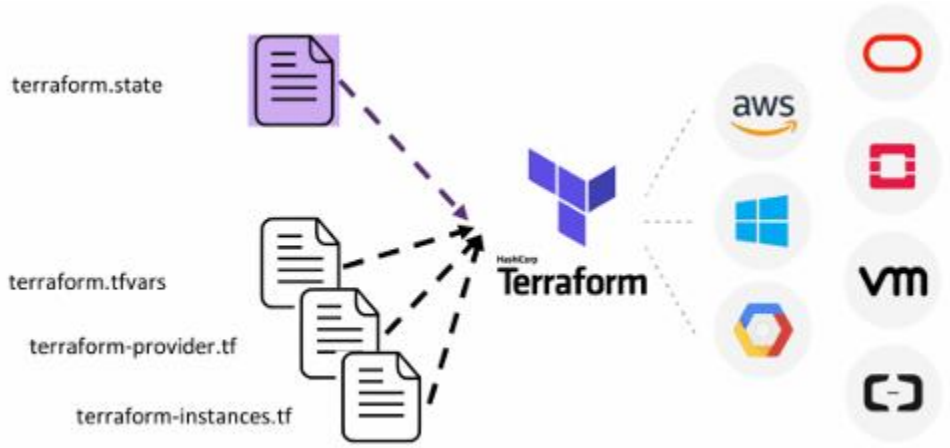
2.2.6 Terraform

Es una herramienta de manejo de infraestructura que nos permite de manera eficiente y segura, construir, modificar y versionar nuestra infraestructura. Permite también manejar estados locales y remotos, lo cual lo vuelve una herramienta ideal para trabajar en equipo en un proyecto sin importar la ubicación geográfica de sus integrantes.

Tiene también muchos casos de uso, como construir un único servidor o un datacenter completo. Puede a su vez, interactuar con varios proveedores de nubes, como lo son AWS, Azure, GCP, entre otros.

Figura 5

Arquitectura de Terraform multi nube



Nota: Adaptado de Architecting AWS with Terraform (p. 41) por E. Kavas, 2023

La arquitectura de Terraform se compone de los siguientes elementos:

- Proveedores

  Para interactuar con varios servicios en la nube, Terraform se apoya en plugins conocidos como proveedores. Son la interfaz entre Terraform y la infraestructura objetivo, permitiéndole, mediante estos, crear o modificar la infraestructura deseada. Cada uno de estos proveedores posee una documentación detallada de cómo debe ser usado

- Módulos

  Un módulo es esencialmente un conjunto de archivos de plantilla que residen en un mismo directorio. Pueden ser constituidos por uno o más archivos de plantilla.

  El beneficio que conlleva organizar la infraestructura en módulos es que nos permite la reusabilidad de recursos a lo largo de diferentes proyectos siguiendo

un estándar predefinido, consiguiendo así que el código sea escalable y mantenible, además de brindar mayor velocidad y capacidad de mantenimiento con el tiempo.

- Recursos

  Terraform utiliza bloques de recursos para administrar diferentes tipos de infraestructura, como redes virtuales, instancias de cómputo y servicios de alto nivel, como registros DNS. Estos bloques representan uno o más objetos de infraestructura definidos en la configuración de Terraform.

  Cada proveedor de Terraform ofrece una variedad de recursos que corresponden a las API necesarias para gestionar un tipo específico de infraestructura.

  Con características avanzadas, como la capacidad de crear múltiples recursos similares a partir de una sola declaración, los usuarios pueden explorar y consultar toda la información detallada en la documentación oficial de cada proveedor en el registro de Terraform.

- Plantillas

  Una plantilla de Terraform es un conjunto de archivos que describe cómo debe configurarse la infraestructura deseada. Estos archivos pueden contener definiciones de variables, recursos y módulos. Dependiendo de tus necesidades y preferencias, puedes almacenar toda la configuración en un único archivo o distribuirla en varios dentro de un directorio

Según Kavas (2023), el flujo de trabajo de Terraform se compone de cinco pasos fundamentales:

- Write

  Escribir el código de Terraform en el editor de texto preferido, para Kavas, es una buena práctica mantener el código versionado en un repositorio, aun si se está trabajando solo

- Init

  El comando Terraform init inicializa el directorio en donde es ejecutado. Según Kavas, es recomendado ejecutar este comando incluso antes de comenzar a escribir el código de Terraform o después de clonar un repositorio de Terraform existente.

  No hay problema en ejecutar este comando múltiples veces, dado que es seguro, incluso si algunas ejecuciones posteriores presenten algún fallo, no eliminara la configuración o el estado existente.

  Este comando puede ser usado para múltiples propósitos, como la instalación de plugins, instalación de módulos hijos y la inicialización del backend

- Plan

  El comando Terraform Plan nos permite ver las modificaciones que Terraform está planeando hacer en base al estado existente y las diferencias con el código. Una vez que el comando Terraform plan ha sido ejecutado, Terraform lee el estado de los objetos remotos para verificar los estados actuales, Luego compara el estado actual con el que se está planeando ejecutar, identificando

discrepancias si es que existen, por último, sugiere un listado de acciones que pueden alinear el estado actual de los recursos al estado deseado según el código. Cabe recalcar que no se realiza ningún cambio al ejecutar el plan, simplemente es una previsualización de los que se podría cambiar. Si no detecta diferencias, Terraform indicara que ningún cambio es requerido para la infraestructura actual.

- Apply

  El comando Terraform apply nos permite aplicar los cambios sugeridos por el comando Terraform plan. La forma más sencilla es ejecutarlo sin argumentos, lo que generará automáticamente un nuevo plan de ejecución (similar a Terraform plan) y solicitará confirmación antes de llevar a cabo las acciones propuestas.

  A menos que se indique lo contrario, Terraform apply siempre pedirá tu aprobación antes de realizar cambios en la infraestructura mediante la API del proveedor, claro que se puede forzar la ejecución de cambios mediante el argumento –auto-approve.

  Si no hay diferencias entre la configuración actual y el estado registrado por Terraform, no se aplicarán cambios.

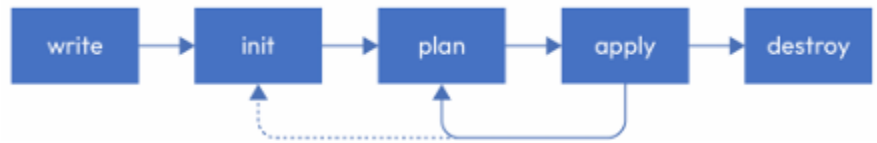
- Destroy

  El comando Terraform destroy se usa para destruir fácilmente todos los objetos remotos gestionados por una configuración específica de Terraform. Al igual que Terraform apply, siempre te pedirá una aprobación antes de hacer el

cambio y de igual manera, se puede saltar esta petición mediante el argumento –auto-aprobe, aunque es recomendado realizarlo con sumo cuidado, ya que se puede eliminar infraestructura necesaria sin querer.

Figura 6

Flujo de trabajo de Terraform



Nota: Adaptado de Architecting AWS with Terraform (p. 45) por E. Kavas, 2023

2.2.7 Infraestructura como Código

La infraestructura como código es un enfoque de la automatización de infraestructura basado en las practicas del desarrollo del software, en donde se hacen cambios en el código y luego se aplica la automatización para probar y aplicar los cambios en los sistemas. (Morris, 2021)

Las prácticas de infraestructura como código han evolucionado desde la gestión de servidores a la gestión de pilas completas, sin embargo, esta mayor potencia se logra a expensas de una mayor complejidad. (Morris, 2021, p. 3)

Existen dos principios fundamentales en la infraestructura como código:

- Idempotencia:

  Se refiere a la capacidad de ejecutar la misma operación múltiples veces y obtener el mismo resultado, sin alterarlo más que para hacer copias idénticas.

  En el contexto de Infraestructura como código, Según Kavas, significa que sin

importar la cantidad de veces que se ejecute la infraestructura como código, el estado final permanece siendo el mismo, minimizando la probabilidad de resultados inesperados, y permitiendo así hacer un roll back en caso de algún fallo

- Inmutabilidad

  Según Kavas (2023), la infraestructura inmutable es una técnica que permite construir y manejar la infraestructura de una manera dependiente, repetible y previsible. En lugar de alterar la infraestructura existente, la infraestructura inmutable involucra en reemplazarla con una nueva, de esta manera se evitan los conflictos en las configuraciones y se puede reproducir múltiples veces (p. 45)

En resumen, para Morris (2021), las organizaciones que implementan Infraestructura como Código para gestionar infraestructuras dinámicas buscan obtener beneficios como:

- Utilizar la infraestructura de TI para facilitar la entrega rápida de valor.
- Reducir el esfuerzo y los riesgos asociados con los cambios en la infraestructura.
- Permitir que los usuarios de la infraestructura accedan a los recursos que necesitan, en el momento que los requieren.
- Ofrecer herramientas comunes para los equipos de desarrollo, operaciones y otras partes interesadas.
- Crear sistemas que sean confiables, seguros y económicos.

- Hacer visibles los controles relacionados con la gobernanza, seguridad y cumplimiento.
- Mejorar la velocidad de detección y resolución de fallos. (p. 32)

2.2.8 Agente de Teamcity

Un agente de construcción de TeamCity es un software que recibe órdenes del servidor TeamCity para iniciar los procesos de compilación. Cuando se tiene una configuración de producción de TeamCity, es necesario instalar agentes adicionales en máquinas dedicadas. Estos agentes pueden correr sobre él un sistema operativo indiferente al cual fue instalado el servidor.

Según la documentación de JetBrains (s.f.), el agente se conecta al servidor de TeamCity mediante la URL configurada en la propiedad "serverUrl" del agente. Esta es una conexión unidireccional entre el agente y el servidor. A través de esta conexión, el agente establece una conexión HTTP(S) con el servidor TeamCity y lo consulta de forma periódica para recibir nuevos comandos.

2.2.9 Script

Un script es un conjunto de instrucciones en código diseñado para realizar funciones específicas dentro de aplicaciones, servidores o sitios web. Es una parte esencial del software, ya que puede representar tanto la totalidad del código de una aplicación como una función en particular.

¿Para qué sirven los scripts?

Según Abad (2024), los scripts tienen una amplia variedad de usos y aportan numerosas ventajas como se detalla a continuación:

- Automatización de tareas: Pueden simplificar tareas repetitivas, como actualizar contenidos o gestionar formularios y bases de datos, ahorrando tiempo y reduciendo errores.
- Optimización del rendimiento: Ayudan a mejorar la velocidad de carga mediante técnicas como la carga diferida de imágenes o la compresión de archivos.
- Personalización de la experiencia: Adaptan el contenido y las funcionalidades según las preferencias o el comportamiento del usuario.
- Conexión con servidores: Permiten el intercambio de datos con servidores, esencial para aplicaciones que requieren acceso a bases de datos o servicios en la nube.

2.2.10 Integración Continua (Continuous Integration)

Según Shanin et al. (2017) La integración continua es una práctica que consiste en incorporar de manera temprana y frecuente todos los cambios de código en la rama principal de un repositorio compartido de código fuente. (p. 3910) Cada modificación se prueba automáticamente al confirmarse o fusionarse, y se genera una compilación de forma automática. Esta metodología permite detectar y corregir errores y problemas de seguridad con mayor facilidad y en etapas tempranas.

Los beneficios según Gitlab (2024) al implantar integración continua son:

- Usuarios y clientes más satisfechos: Al dismunuir el número de errores y fallos que llegan a producción, los usuarios y clientes disfrutan de una experiencia mejorada. Esto contribuye a aumentar su nivel de satisfacción y confianza,

además de mejorar la reputación de la organización.

- Mayor rentabilidad: La capacidad de implementar cambios en cualquier momento permite que los productos y nuevas funciones se lancen al mercado con más rapidez, implicando una reduccion del costo durante el desarollo y en una respuesta más ágil que libera al equipo para otras tareas. Los clientes reciben resultados más rápidamente, lo que proporciona a la empresa una ventaja competitiva.
- Menos manejo de crisis: Realizar pruebas de código con mayor frecuencia, en bloques más pequeños y en etapas del desarrollo temranas, disminuye significativamente la necesidad de resolver problemas de última hora. Como resultado, el ciclo de desarrollo se vuelve más fluido, el equipo experimenta menos estrés y los resultados son más predecibles, facilitando la detección y corrección de errores.
- Cumplir plazos de manera más confiable: Eliminar los cuellos de botella y hacer que las implementaciones sean más predecibles reduce considerablemente la incertidumbre sobre el cumplimiento de plazos importantes. El dividir el trabajo en tareas más pequeñas y manejables, facilita completar cada etapa a tiempo y seguir el progreso de forma más clara.

2.2.11 Entrega Continua (Continuous Deployment)

La entrega continua es una práctica de desarrollo de software que se complementa con la integración continua (CI) para automatizar tanto la provisión de infraestructuras como el proceso de despliegue de aplicaciones.

Una vez que el código ha sido probado y construido durante el proceso de CI, la entrega continua se encarga de las últimas etapas para asegurarse de que se empaquete adecuadamente con todo lo necesario para ser desplegado en cualquier entorno, en cualquier momento. Esto puede incluir desde la provisión de la infraestructura hasta el despliegue de la aplicación en entornos de prueba o producción en cualquier momento.

Luego, se puede activar el despliegue de manera manual o pasar a un proceso de despliegue continuo, en el cual también está automatizado.

Para Gitlab (2024), los beneficios de la entrega continua son:

- Creación de valor más rápida: Al poder desplegar en cualquier momento, se pueden lanzar productos y nuevas características al mercado más rápidamente. Esto reduce los costes de desarrollo y una respuesta más ágil permite que el equipo se enfoque en otras tareas. Los clientes obtienen resultados más rápido, lo que brinda a la empresa una ventaja competitiva.
- Menos agotamiento: Los estudios muestran que el despliegue continuo ayuda a disminuir los problemas de despliegue y reduce el agotamiento del equipo. Los desarrolladores sienten menos frustración y estrés al trabajar con procesos de CI/CD, lo que se traduce en empleados más felices y menos fatigados.
- Recuperación más rápida: CI/CD facilita la solución de problemas y la recuperación después de incidentes, reduciendo el tiempo medio de resolución (MTTR). Las prácticas de despliegue continuo implican actualizaciones pequeñas y frecuentes, por lo que cuando surgen errores, es más fácil

identificarlos.

2.2.12 ***DevOps***:

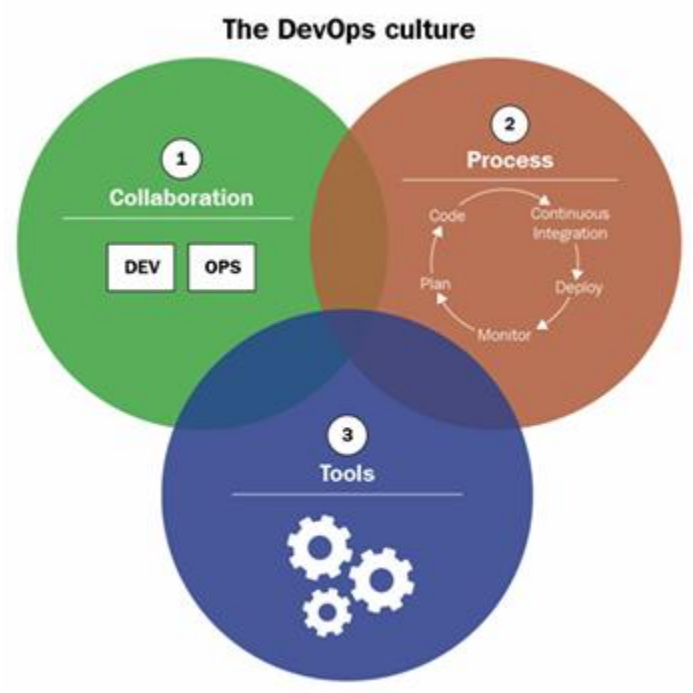
Según Krief, el término DevOps representa la combinación de Desarrollo (Dev) y Operaciones (Ops). En ha dado lugar a un movimiento que aboga por reunir a desarrolladores y operaciones dentro de los equipos. De este modo, se ofrece valor añadido empresarial a los usuarios con mayor rapidez, lo que lo hace más competitivo en el mercado.

Krief (2022) nos menciona que la cultura DevOps consiste en prácticas que facilitan la colaboración entre los desarrolladores, cuyo objetivo es innovar y acelerar la entrega de software, y los equipos de operaciones, responsables de garantizar la estabilidad y la calidad en los cambios aplicados a los sistemas de producción.
El enfoque de DevOps en el desarrollo de software se estructura en cuatro dimensiones: colaboración, automatización, medición y monitoreo (Lwakatare et al., 2015). También amplía las prácticas ágiles, organizándolas en tres fases (Ebert et al., 2016): la fase de construcción, que incluye prácticas de integración continua; la fase de despliegue, con un fuerte enfoque en la gestión de la configuración; y la fase de operación, que abarca monitoreo y registro de logs.

En otras palabras, como menciona Krief (2022) citando a Donovan Brown, “DevOps es la unión de las personas, los procesos y productos para establecer una entrega continua de valor hacia los usuarios”.

Figura 7

La cultura DevOps



Nota: Adaptado de Learning DevOps Second Edition (p. 31) por M. Krief, 2022

Los beneficios de implementar DevOps en la organización son los siguientes

- Mejor comunicación y colaboración entre los equipos
- Menores tiempo de despliegue a producción
- Reducir los costos de la infraestructura mediante infraestructura como código
- Un ahorro considerable de tiempo gracias a los ciclos iterativos que minimizan los errores en la aplicación y a las herramientas de automatización que eliminan las tareas manuales, permitiendo que los equipos se enfoquen más en desarrollar nuevas funcionalidades que aporten valor al negocio.

Por lo tanto, podemos decir que incorporar la gestión de lanzamientos

dentro de la cultura DevOps permite entregar software de forma ágil, segura y satisfactoria, lo que aumenta la satisfacción del cliente, fomenta la colaboración entre equipos y acelera el crecimiento empresarial.

Figura 8

Ciclo DevOps

Nota: Adaptado de Embracing DevOps Release Management (p. 41) por J. Kruger, 2024

2.2.13 Agile

Antes de Agile, las organizaciones habían creado procesos detallados de cómo debía ser creado el software, en donde todo estaba altamente controlado y en teoría no podían existir errores.

Shore et al (2021), mencionan que el proceso consistía en que los analistas de negocio se entrevistarían con las partes interesadas para documentar las necesidades del sistema, luego los arquitectos de software leerían estos documentos y crearían documentos de diseño especificando cada componente del sistema y como se relacionaba con otros, para que, posteriormente, los programadores tomen estos

documentos y los transformen en código. Mientras tanto, los lideres de pruebas utilizarían los mismos documentos para generar planes de prueba, con los cuales, una vez el desarrollo esté terminado, los encargados del área de QA realizarían las pruebas manualmente y reportarían los defectos encontrados. Después de cada fase, todo debía estar cuidadosamente documentado, revisado y firmado. Esta metodología de trabajo es considerada como Waterfall.

Debido a toda esta burocracia, varias personas crearon métodos de trabajo más ligeros y simples, entre los cuales destacan los nombres como “Extreme Programing” y “Scrum”

Posteriormente todos estos métodos livianos se encapsularían bajo un solo nombre: Agile, y junto con esto, nacería el Agile Manifesto

Figura 9

Manifiesto de la cultura Agile

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- **Individuals and interactions** over process and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas

© 2001, the above authors
This declaration may be freely copied in any form, but only in its entirely through this notice.

Nota: Adaptado de The Art of Agile Development (p. 5) por J. Shore et al., 2022

Como mencionan Shore et al (2021), Agile es una filosofía, no una metodología o algo que se pueda hacer. Está más orientado a las personas que a los procesos.

La filosofía ágil es altamente efectiva, ya que sus ciclos de desarrollo cortos permiten a los equipos detectar problemas desde etapas tempranas. No obstante, depender demasiado del feedback de los clientes puede llevar a un aumento descontrolado del alcance del proyecto. Este enfoque es especialmente adecuado para proyectos de desarrollo de software que necesitan ajustarse y evolucionar con el tiempo.

Figura 10

El modelo ágil de gestión de versiones

Nota: Adaptado de Embracing DevOps Release Management (p. 40) por J. Kruger, 2024

2.2.14 Scrum

Scrum es un marco de trabajo ligero que ayuda a las personas, equipos y organizaciones a generar valor mediante soluciones adaptativas a problemas complejos. (Schwaber & Sutherland, 2020).

Se puede entender a Scrum como un marco de trabajo que combina principios del empirismo y el pensamiento lean para mejorar la eficiencia y la adaptabilidad en proyectos complejos. Basado en la experiencia y el análisis de resultados observables, fomenta decisiones fundamentadas y elimina actividades innecesarias para centrarse en lo realmente importante. Este enfoque incremental e iterativo permite a los equipos gestionar riesgos y alcanzar una mayor predictibilidad en sus entregas.

A su vez, promueve la colaboración entre personas con las habilidades

necesarias para completar el trabajo, apoyando el aprendizaje continuo y la adquisición de nuevas capacidades cuando es necesario. Su estructura gira en torno al Sprint, un ciclo de trabajo definido que incluye cuatro eventos clave para garantizar la transparencia, evaluar el progreso y ajustar los planes según sea necesario, impulsando así la mejora constante.

Según Schwaber y Sutherland (2020), Scrum se sustenta en cinco valores fundamentales que, como se interpreta en este trabajo, guían el comportamiento y la colaboración del equipo:

• **Compromiso**: El Equipo Scrum se dedica plenamente a alcanzar sus objetivos y a ofrecer apoyo mutuo, asegurando que todos trabajen de manera alineada hacia un propósito compartido.

• **Enfoque**: El equipo dirige toda su atención al trabajo del Sprint, priorizando las actividades esenciales para avanzar significativamente hacia las metas establecidas.

• **Apertura**: Tanto el Equipo Scrum como los interesados adoptan una actitud transparente, compartiendo de forma sincera los avances logrados y los obstáculos enfrentados durante el proceso.

• **Respeto**: Los integrantes del equipo valoran las habilidades, la independencia y las aportaciones de sus compañeros, fomentando un ambiente de respeto mutuo que también se extiende a las personas para las que trabajan.

• **Coraje**: El equipo demuestra valentía al tomar decisiones éticas y necesarias, enfrentando con resolución los problemas complejos y desafiantes que puedan surgir en su camino.

El Equipo Scrum: Es un equipo pequeño y multifuncional compuesto por un

Scrum Master, un Product Owner y desarrolladores, sin sub-equipos ni jerarquías. Su objetivo principal es trabajar de manera cohesionada para lograr un único propósito a la vez: el Objetivo del Producto.

Características de los equipos Scrum.

- Los equipos de Scrum son autogestionados, deciden internamente quien hace que, cuando y como.
- Son multifuncionales, se componen de diferentes individuos con habilidades necesarias para crear valor en cada Sprint
- Generalmente constan de 10 personas o menos, logrando así optimizar la agilidad, productividad y favoreciendo una comunicación efectiva. En el caso de que sobrepase el número máximo recomendado de 10 personas, se podrían dividir en sub-equipos compartiendo el mismo backlog

Responsabilidades del equipo Scrum. El equipo es responsable de todas las actividades relacionadas con el producto, incluyendo:

- Colaboración con los interesados.
- Verificación, mantenimiento y operación del producto.
- Investigación, experimentación y desarrollo.

Entrega de Valor en Cada Sprint. La finalidad de la metodología Scrum es el conseguir un entregable incremental, valioso y funcional al finalizar cada Sprint. Los equipos Scrum por lo general se componen de las siguientes funciones clave

- Desarrolladores

  Son los responsables de crear los aspectos utilizables de los entregables en cada Sprint. Su trabajo incluye:

  - Planificar el Sprint mediante el Sprint Backlog.
  - Garantizar la calidad adhiriéndose a la Definición de Terminado.
  - Ajustar el plan diariamente para cumplir el Objetivo del Sprint.
  - Mantener la responsabilidad mutua como profesionales.

- Product Owner:

  Se encarga de maximizar el valor del producto derivado del trabajo del equipo.

  Sus responsabilidades incluyen:

  - Definir y comunicar claramente el Objetivo del Producto.
  - Crear, ordenar y garantizar la transparencia del Product Backlog.

  Para tener éxito, necesita que la organización respete sus decisiones, las cuales son visibles en el Product Backlog y los resultados de los entregables. El Product Owner actúa como único responsable y puede representar a múltiples interesados, quienes deben persuadirlo si desean cambios en el Backlog.

- Scrum Master:

  Es el líder servicial que asegura la correcta implementación de Scrum, promoviendo la comprensión de su teoría y práctica tanto dentro del equipo como en la organización.

  Su apoyo al equipo Scrum consiste en:

  - Fomentar la autogestión y multifuncionalidad.

- Ayudar a entregar entregables valiosos que cumplan la Definición de Terminado.
- Eliminar impedimentos que bloqueen el progreso.
- Facilitar eventos Scrum productivos y dentro del tiempo establecido.

También brinda apoyo al Product Owner de la siguiente manera:

- Definir objetivos claros y gestionar eficazmente el Product Backlog.
- Promover una planificación empírica en entornos complejos.
- Facilitar la colaboración con los interesados.

Por último, su apoyo a la organización consiste en:

- Liderar y capacitar en la adopción de Scrum.
- Planificar implementaciones de Scrum en la organización.
- Promover un enfoque empírico para el trabajo complejo.
- Eliminar barreras entre los interesados y los equipos Scrum.

Ceremonias De Scrum. La metodología Scrum establece las siguientes ceremonias como marco de trabajo:

Sprint. Son eventos de duración fija, pueden ser de un mes o menos, dependiendo de cómo se organice el equipo. Cada Sprint comienza una vez se acabe el anterior.

Las demás ceremonias como el Sprint Planning, Daily Scrum, Sprint Review y Sprint Retrospective tienen lugar en el Sprint.

Durante el Sprint no se realizan cambios que puedan poner en peligros los objetivos de este, y se puede aclarar y renegociar el alcance con el Product Owner.

Existen varias métricas para monitorear el desarrollo del Sprint, con las cuales se puede tomar decisiones para los desarrollos futuros. Se puede cancelar un Sprint si el objetivo de este se vuelve obsoleto, y solo el Product Owner tiene la potestad para cancelarlo.

Sprint Planning. Se establece el trabajo que se realizará durante el Sprint en un entorno colaborativo, en donde todos los integrantes del equipo participan. Se debe definir el objetivo del Sprint antes de finalizar el Sprint Planning, el cual puede tener una duración de máximo 8 horas para un Sprint de un mes, aunque es una referencia ya que cada equipo lo puede adaptar a sus necesidades.

En el Sprint Planning, se abarcan los temas de

- ¿Por qué es valioso el Sprint?

El Product Owner propone como el producto podría mejorar en el Sprint actual, todo el equipo colabora para definir un objetivo del Sprint en donde se responde a la pregunta. El objetivo del Sprint debe definirse antes de terminar el Sprint Planning.

- ¿Qué se puede hacer en este Sprint?

El equipo, junto con el Product Owner, seleccionan los elementos del Backlog para incluirlos en el Sprint actual, los cuales se pueden refinar durante el proceso.

- ¿Cómo se realizará el trabajo escogido?

Los desarrolladores planifican el trabajo necesario para crear un entregable que cumpla con la definición de terminado. Por lo general se consigue descomponiendo los elementos del Backlog en tareas más pequeñas

Daily Scrum. Según Schwaber y Sutherland (2020), el propósito de la Daily

Scrum es inspeccionar el progreso hacia el Objetivo del Sprint y adaptar el Sprint Backlog según sea necesario, ajustando el trabajo planificado entrante.

Es un evento diario, por lo general de 15 minutos, en donde se revisa como va el desarrollo del Sprint, se comunican los avances y si es que existe algún bloqueante o impedimento con las tareas asignadas.

Sprint Review. Durante esta parte de la ceremonia, el equipo y los interesados revisan los logros obtenidos durante el Sprint y los cambios que ocurrieron en este. En base a esa información, se toman decisiones sobre como continuar en el futuro. Es la penúltima ceremonia de Scrum y tiene un límite de cuatro horas para un Sprint de un mes.

Sprint Retrospective. Según Schwaber y Sutherland (2020), el propósito de la Sprint Retrospective es planificar formas de aumentar la calidad y la efectividad.

En esta ceremonia se analiza cómo le fue al equipo durante el Sprint, como fueron las interacciones con otros equipos, así como también como les fue con las herramientas proporcionadas y la definición de terminado establecida. Se identifican además los problemas que existieron durante el Sprint y se analiza sus orígenes y como podrían resolverse para un Sprint futuro.

Con esta ceremonia se concluye el Sprint, por lo general tiene un tiempo de 3 horas para un Sprint de un mes, aunque en equipos más pequeños el tiempo puede ser menor.

2.2.15 Puppet

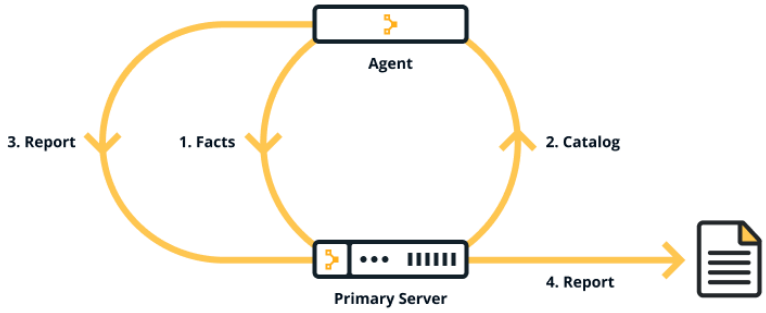
Puppet (s.f.) nos menciona que es una herramienta para automatizar y gestionar la configuración de servidores. A través de su código declarativo, se define el estado

deseado de los sistemas, sin detallar los pasos específicos para alcanzarlo.

Utiliza un servidor principal para almacenar el código y un agente que ejecuta los comandos en los sistemas especificados, automatizando así el proceso de configuración y mantenimiento.

Figura 11

Arquitectura servidor agente de una ejecución de Puppet



Nota: Adaptado de ¿Que es Puppet? [Ilustración] s.f. Puppet (https://www.puppet.com/docs/puppet/7/what_is_puppet.html)

2.2.16 Amazon Elastic Compute Cloud (EC2)

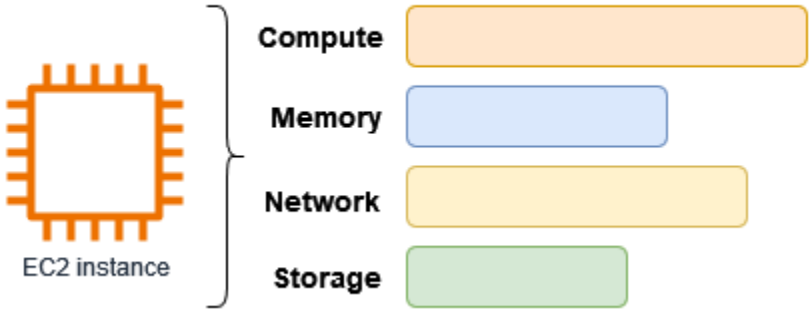
Es un servidor en la nube de AWS, donde el tipo de instancia que seleccione determina los recursos de hardware disponibles. Brinda una combinación distinta de características dependiendo del tipo de instancia que se elija, además de computación escalable que ayuda a minimizar los costos de hardware y acelerar el desarrollo y despliegue de aplicaciones.

Puede ser usado para lanzar servidores virtuales según sea necesario, configurar seguridad y redes, y gestionar almacenamiento. Además, permite escalar la capacidad (aumentando verticalmente) para manejar tareas intensivas en computación, como

procesos mensuales o picos de tráfico, y reducirla (disminuyendo verticalmente) cuando el uso disminuye.

Figura 12

Componentes de una instancia EC2



Nota: Adaptado de ¿Qué es Amazon EC2? [Ilustración] s.f. Amazon Web Services (https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html)

2.3 Variables a medir

- Costo de los agentes en AWS
- Tiempo de creación y de provisionamiento de los agentes

2.4 Limitaciones

- El trabajo tenía que ser bajo la nube de AWS ya que se estaba pasando por una migración de Azure a AWS,
- Los agentes de despliegue no contaban con ninguna herramienta como Ansible, Chef o Puppet.
- La versión de Teamcity en ese momento era 2023.05, la cual no solo poseía la integración de los Cloud Profiles con AWS, Kubernetes y VMWare. La integración con Azure y otros proveedores de la nube eran manejado por terceros.

- Los agentes de construcción necesitan tener Docker disponible para los proyectos.
- Todo debía ser manejado bajo el esquema de infraestructura como código
- Soporte y/o mantenimiento del área de IT por el ancho de banda
- Dado que la versión de Teamcity actual es antigua, se necesitaba crear llaves de acceso específicas para permitir la comunicación entre Teamcity y AWS

CAPÍTULO III. DESCRIPCIÓN DE LA EXPERIENCIA

Ingresé a la empresa en el año 2022, precisamente en el mes de mayo como un DevOps Engineer con experiencia en AWS, Azure, Kubernetes, CI/CD y herramientas de monitoreo para apoyar al equipo en el manejo y soporte de la infraestructura y las aplicaciones. Se me asignó al subproyecto llamado SITH, el cual se encarga principal, mas no exclusivamente, de brindar soporte al equipo de automatización, encargado de las pruebas automatizadas para verificar el correcto funcionamiento de las diferentes aplicaciones de la empresa.

Al momento del ingreso éramos 3 personas en el equipo y si bien se manejaban los despliegues mediante herramientas centrales como Octopus Deploy, algunos de las herramientas utilizadas por el equipo estaban desplegadas en Kubernetes directamente mediante manifiestos de Kubernetes gestionados desde la misma herramienta de despliegue. Así como también se encontraban los lanzamientos de Helm ejecutados a mano en el clúster de Kubernetes. Mi función al inicio fue de automatizar estos despliegues mediante Helm charts y volver a desplegar o importar según sea el caso las herramientas desplegadas a mano, mediante la gestión del código en un repositorio remoto y la intervención de herramientas de despliegue centralizadas para dicho fin.

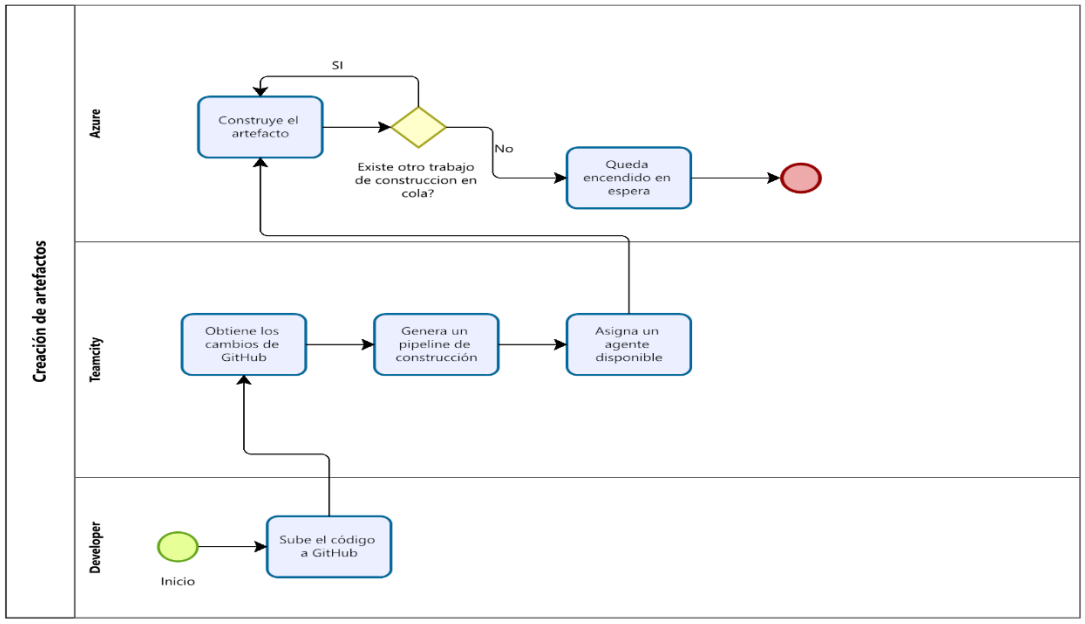
Luego de alrededor de un año de ingresado a la empresa, el jefe del área se comunicó conmigo dado que había identificado una falencia en el pipeline de integración continua que era ejecutado en Teamcity, dado que los agentes en ese momento permanecían encendidos las 24 horas del día, los 7 días de la semana. Él ya había investigado o sido informado de una funcionalidad de la herramienta Teamcity que

permitía crear agentes a demanda y que tenía dos opciones nativas de implementación: Kubernetes y AWS. Otra problemática que también se buscaba solucionar era el hecho de tener que manejar las versiones de los programas en los agentes manualmente mediante acceso remoto, por lo que el requerimiento era que la solución se realice utilizando infraestructura como código, en este caso con la herramienta Terraform.

En la Figura 13 se puede apreciar cómo era el proceso de construcción de imágenes utilizando los agentes durante la problemática.

Figura 13

Proceso de construcción de imágenes antes de los Cloud Profiles



Es así como al ser uno de los conocedores de ambas tecnologías, se me asigna la tarea de realizar la investigación e implementación de dicha característica, con el fin de aligerar los costos de mantenimiento que implicaba tener los agentes disponibles, mejorar el proceso

de mantenimiento de versiones y/o parches de seguridad de las aplicaciones instaladas en los agentes y manejar el provisionamiento de los agentes mediante infraestructura como código en un sistema de control de versiones centralizado.

Objetivos:

Objetivo General:

Optimizar el proceso de provisionamiento de agentes de construcción en el área de TI implementando AWS y CI/CD en la empresa.

Objetivos Específicos:

- Reducir los costos de los agentes de construcción alojando la infraestructura en AWS para optimizar el proceso de provisionamiento de agentes de construcción en el área de TI implementando AWS y CI/CD en la empresa.
- Reducir el tiempo de provisionamiento y despliegue de los agentes de Teamcity para optimizar el proceso de provisionamiento de agentes de construcción en el área de TI implementando AWS y CI/CD en la empresa.
- Reducir las vulnerabilidades de seguridad para optimizar el proceso de provisionamiento de agentes de construcción en el área de TI implementando AWS y CI/CD en la empresa.

¿Por qué Amazon y no Microsoft Azure?

Se eligió el proveedor de la nube AWS debido a que es líder en el mercado, como se puede apreciar en la Figura 14, así como también que cumple en la mayoría de sus servicios con

los ISO/IEC 27001 Gestión de seguridad de la información, 27017 Controles de seguridad específicos para servicios en la nube y 27018 Protección de datos personales en la nube.

Además de esto, vimos más provechosas sus herramientas de seguridad, con una interfaz más amigable al usuario y una mayor facilidad de uso. Con la ayuda de IAM y Terraform, podríamos gestionar los permisos de una manera eficiente. Al poseer mayores regiones y zonas de disponibilidad, encontramos en AWS un aliado bastante flexible con respecto a las ubicaciones de los recursos en general. Como punto influyente, mediante la herramienta AWS Systems Manager Session Manager, podemos acceder a las instancias sin necesidad de utilizar una llave privada o una conexión SSH, con lo cual, obtenemos una mayor seguridad dado que las maquinas no necesitan exponer puertos específicos y se puede controlar quien puede interactuar con Systems Manager mediante permisos de IAM.

Otra de las características que se tomó en cuenta fue la integración que la versión de Teamcity ofrecía para AWS. Las comparaciones que se tomaron en cuenta se encuentran en la Tabla 1.

Tabla 1

Comparación entre los proveedores de nube AWS y Microsoft Azure

Aspecto	AWS	Microsoft Azure	Ventaja
Certificaciones de Seguridad	Amplia gama, incluyendo ISO 27001, 27017, 27018, SOC 1/2/3, PCI DSS, FedRAMP, HIPAA, RGPD.	Cumple con estándares similares, pero menor cobertura por región para algunos servicios.	AWS: Mayor cobertura global.
Regiones y Zonas	31 regiones, 100+ zonas de disponibilidad, fuerte redundancia regional.	60+ regiones, menos zonas de disponibilidad por región.	AWS: Más zonas por región
Herramientas de Seguridad	IAM, Security Hub, Macie, Shield, WAF, soporte para FIPS 140-2 y herramientas basadas en IA.	Azure Security Center, Sentinel, soporte de herramientas similares, pero menos IA avanzada.	AWS: Herramientas más avanzadas.
Innovación	Líder en ritmo de lanzamientos de nuevos servicios y características.	Innovador, pero históricamente más lento en implementar nuevas características.	AWS: Mayor ritmo de innovación.
Cálculo de Costos	Facturación predecible y	Facturación menos transparente, menor	AWS: Facturación más clara.

	herramientas avanzadas para optimizar costos (ej. Instancias Spot).	flexibilidad en opciones de ahorro.

Figura 14

Cuadro comparativo de proveedores en la nube 2023

Nota: Adaptado de Magic Quadrant for Strategic Cloud Platform Services [Ilustración] por S. Stormacq, 2024 (https://aws.amazon.com/es/blogs/aws/read-the-2023-gartner-magic-quadrant-for-strategic-cloud-platform-services)

Ejecución del trabajo

El trabajo se realizó siguiendo la metodología Scrum, en donde se asignaban las tareas correspondientes en un sprint de 2 semanas de duración. El plan de trabajo se puede ver reflejado en la Figura 15 y la distribución de las tareas con su peso en la Tabla .

Figura 15

Diagrama de Gantt

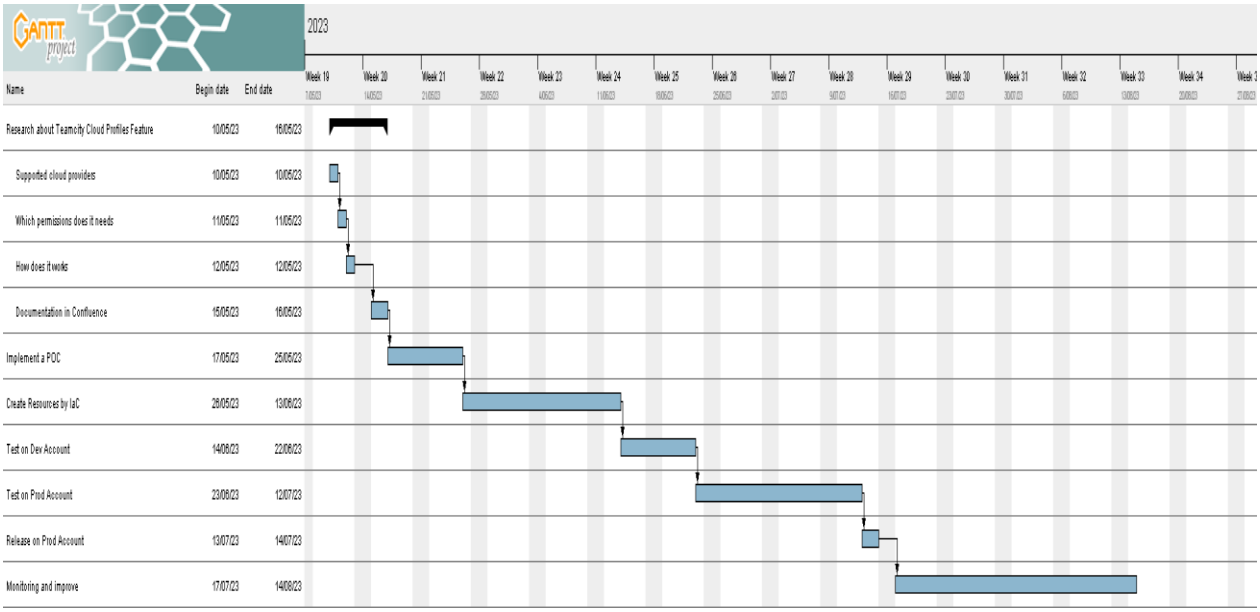


Tabla 2

Detalle de actividades por Sprint

Sprint	Tarea	Peso
Sprint 1: Investigación y preparación inicial	Investigar sobre Teamcity Cloud Profiles	3
	Identificar proveedores de Cloud compatibles	3
	Determinar los permisos necesarios en AWS	5

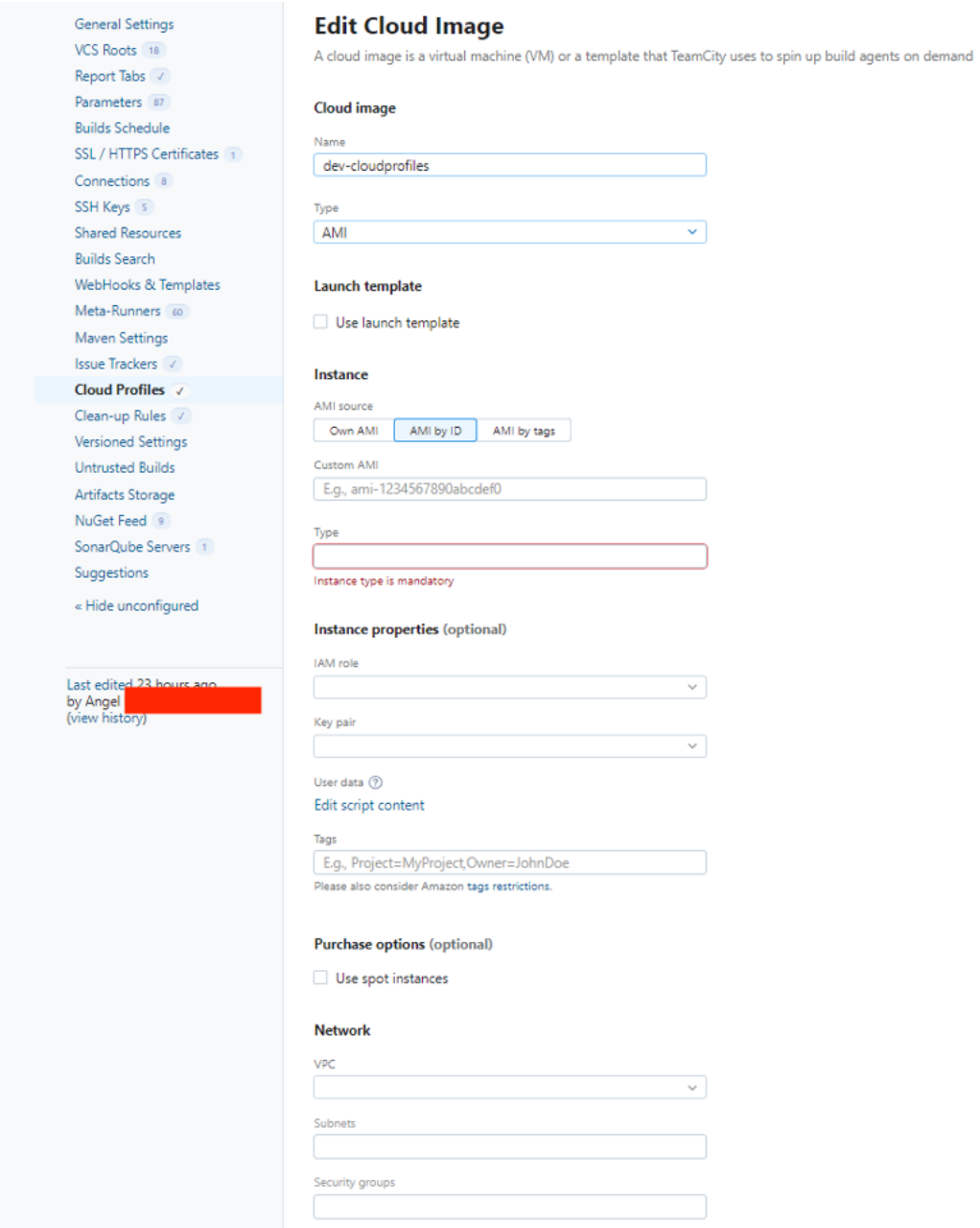
Total Sprint		11
Sprint 2: Diseño y prototipado	Definir cómo funciona la integración	5
	Crear documentación en Confluence	2
	Definir el prototipo inicial	3
Total Sprint		10
Sprint 3: Implementación inicial	Implementar el prototipo en el ambiente de desarrollo	5
	Automatizar la creación de los recursos mediante IaC (terraform)	8
	Realizar pruebas en el entorno de desarrollo	3
Total Sprint		16
Sprint 4: **Pruebas avanzadas y ajustes**	Implementar la solución en el entorno de producción para pruebas	8
	Ajustar configuraciones y políticas en producción	3
	Documentar en Confluence	2
Total Sprint		13
Sprint 5: Lanzamiento y monitoreo	Desplegar en producción	5
	Configurar monitoreo y alertas	5
	Monitorear, mejorar y optimizar	5

Total Sprint	**15**

Realizando la investigación sobre los requerimientos necesarios, nos dimos cuenta que la característica de los Cloud Profiles de la herramienta Teamcity requería una AMI para poder crear las instancias a demanda, además de la configuración necesaria como en que subnet y zona de disponibilidad estaría, cuál sería el tamaño de la instancia, si se usaría una instancia del tipo spot o no, si usaría un perfil de instancia, si usaría una plantilla de lanzamiento, que grupo de seguridad usaría, si se le desea agregar más tags a la instancia o alguna configuración extra al momento de ser iniciada, como se puede ver en la Figura 16.

Figura 16

Campos requeridos por los Cloud Profiles de Teamcity



Así como el diagrama de Gantt mostrado en la Figura 15, también se crearon tarjetas de Jira relacionadas a las tareas especificadas para un mejor seguimiento de las tareas como se

muestra en las Figuras 17, 18 y 19.

Figura 17

Ticket de Jira para la documentación de los Cloud Profiles en Confluence

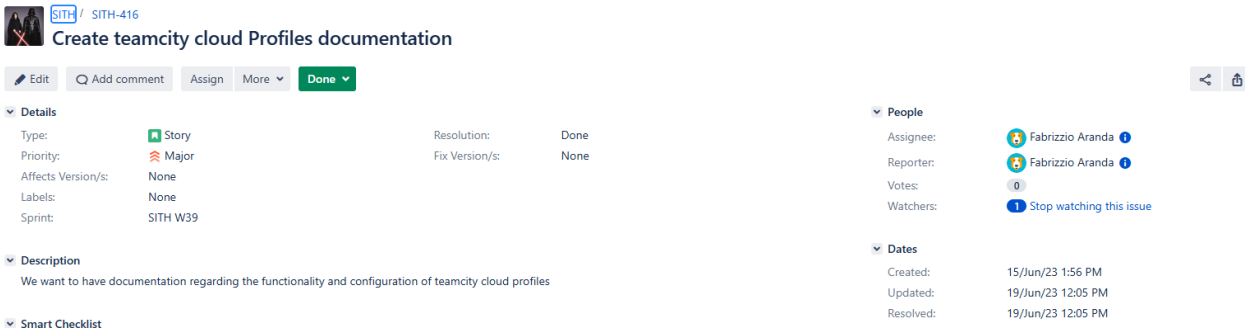
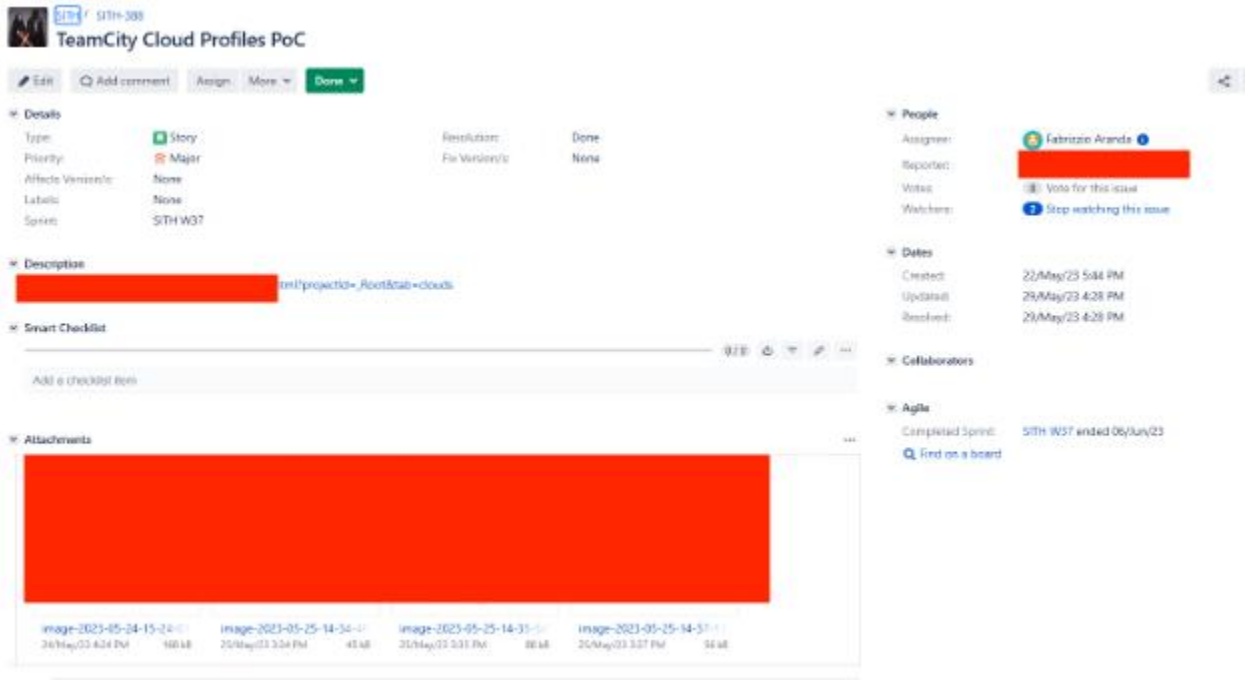


Figura 18

Ticket de Jira para la implementación del PoC de Cloud Profiles

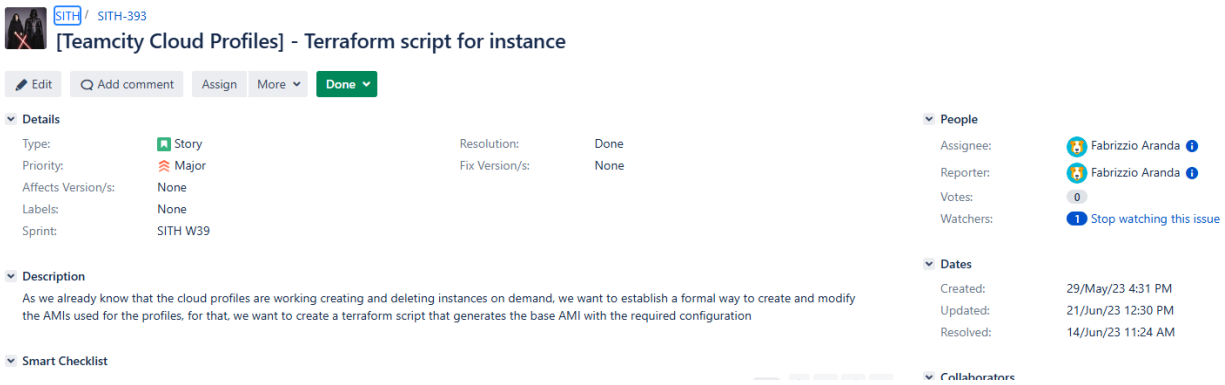


En base a la prueba de concepto implementada, y con el fin de manejar todos los mediante Terraform, nos dimos cuenta de que muy aparte del provisionar el agente con las herramientas necesarias mediante el script de Bash, se tenía que detener y crear una AMI una vez

que se haya terminado la configuración. Esto se logró de igual manera utilizando dependencias anidadas como se puede apreciar en el Figura 24.

Figura 19

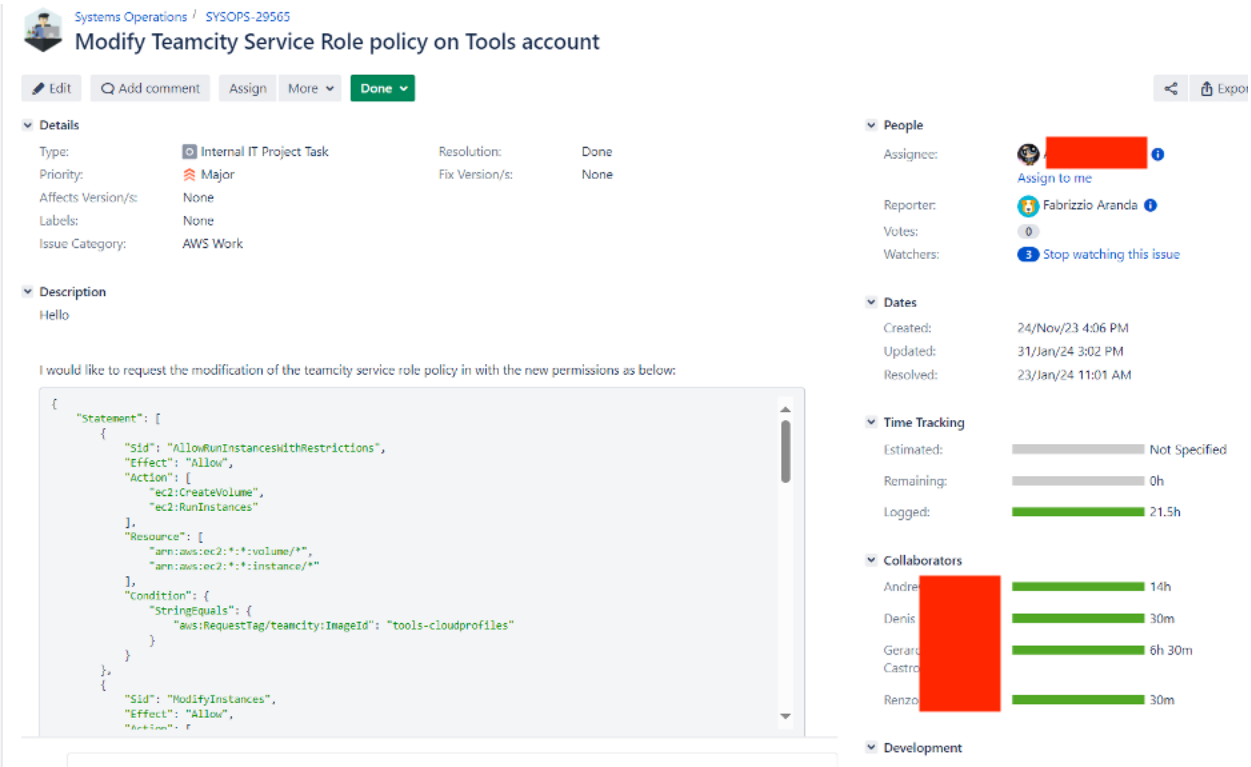
Ticket de Jira para la creación de los recursos mediante Terraform



Otro requerimiento de Teamcity para poder crear las instancias EC2 en AWS fue que se necesitaban un usuario IAM con permisos específicos, según la documentación de la característica. Dado que nuestra área no crea directamente usuarios ni roles en los ambientes de AWS que maneja la empresa, se solicitó al área de SysOps la creación de estos mediante un ticket de Jira, adjuntando los permisos requeridos por el usuario, indicado en la Figura 20.

Figura 20

Ticket de Jira solicitando la creación de permisos requeridos



En adición a esto, otro requerimiento que se encontró fue el que los agentes puedan descargar los repositorios desde el sistema de control de versiones centralizado, en este caso GitHub. La problemática con respecto a esto fue que al estar los repositorios como internos en GitHub, no eran visibles sin estar autenticados, ocasionando así que no se puedan descargar los repositorios correctamente

Para solucionar esto, se utilizó un `*remote exec*` provider en Terraform para obtener acceso remoto la instancia tiempo de creación y mediante un comando de bash o Powershell, especificar el token mediante el comando `*git config –global url*` como se puede apreciar en la Figura 21. Esto con el fin de permitir al usuario bajo el cual corre el agente de Teamcity tener las

credenciales de GitHub en su perfil de manera predeterminada.

Figura 21

Adición de las credenciales de GitHub a las instancias mediante Terraform

```
provisioner "remote-exec" {
  connection {
    type            = "ssh"
    user            = var.provisioner_user
    password        = aws_instance.ec2_cloud_profiles.get_password_data ? rsadecrypt(aws_instance.ec2_cloud_profiles.password_data, var.private_key) : null
    host            = aws_instance.ec2_cloud_profiles.private_ip
    insecure        = true
    private_key     = length(regexall("windows", var.name)) > 0 ? null : var.private_key
    target_platform = length(regexall("windows", var.name)) > 0 ? "windows" : "unix"
    timeout         = "10m"
  }
  ## IF computer name contains windows
  inline = length(regexall("windows", var.name)) > 0 ? [
    "powershell.exe while (-not (Test-Path \"C:\\Users\\Administrator\\user-data-status.txt\")) { Start-Sleep -Seconds 6;};",
    "C:\\\"Program Files\"\\Git\\mingw64\\bin\\git config --global url.\"https://oauth2:${var.github_token}@github.com\".insteadOf https://github.com"
    ] : [
    # else
    "until [ -f /var/lib/cloud/instance/boot-finished ]; do sleep 5; done;",
    "sudo -u devops sh -c 'git config --global url.\"https://oauth2:${var.github_token}@github.com\".insteadOf https://github.com'"
  ]
}
```

Se muestra el script para ambos sistemas operativos dado que el módulo de Terraform permite la creación de máquinas tanto Linux como Windows, los agentes de Windows aún están en proceso de implementación.

Luego de configurada la instancia, se procede a detener y obtener una AMI (Amazon Machine Image) de la misma, lo cual se logra mediante el código mostrado en las Figuras 22, 23 y 24.

Figura 22

Código de Terraform con el cual detenemos la instancia

```
resource "aws_ec2_instance_state" "stop_instance" {
  instance_id = aws_instance.ec2_cloud_profiles.id
  state       = "stopped"
}
```

Cabe recalcar, que la creación de la imagen de Amazon depende de la instancia que haya sido provisionada, a su vez, se debe primero crear la nueva imagen de Amazon para que se pueda

destruir la anterior, esto se logra mediante el *lyfecycle create_before_destroy = true*, y se realiza de esta manera para contar con una resiliencia a fallos en el caso de que la nueva instancia y/o imagen presente un error al momento de su creación.

Figura 23

Código de Terraform para la creación de la AMI

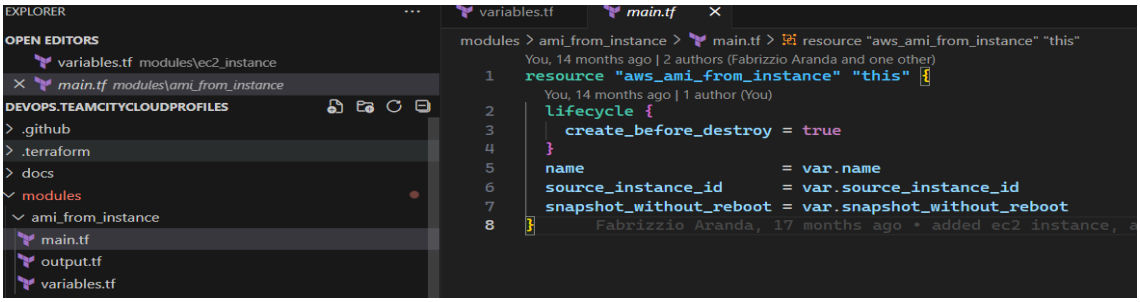
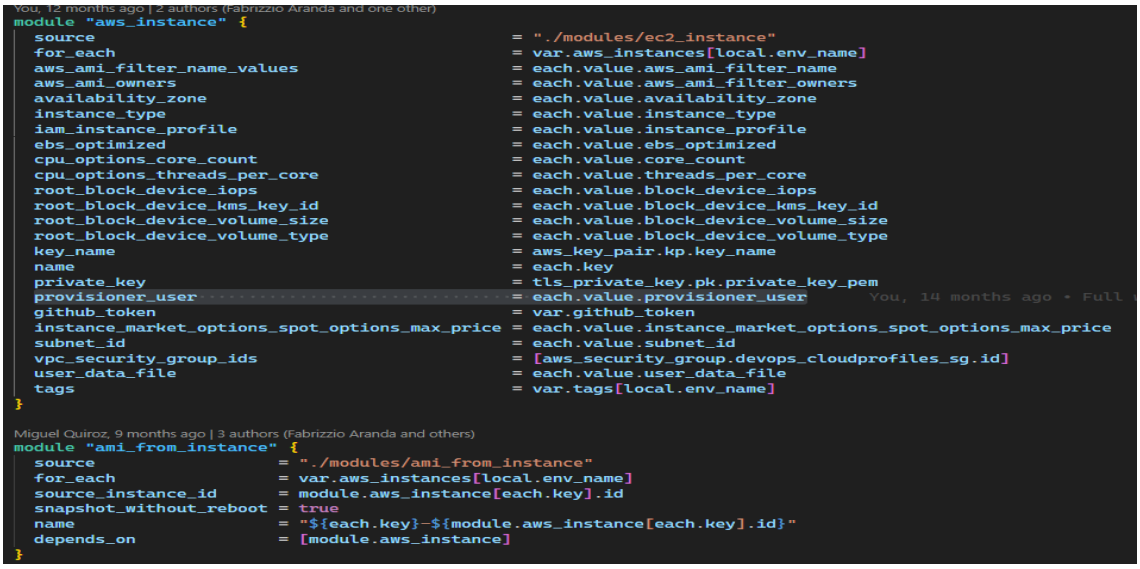


Figura 24

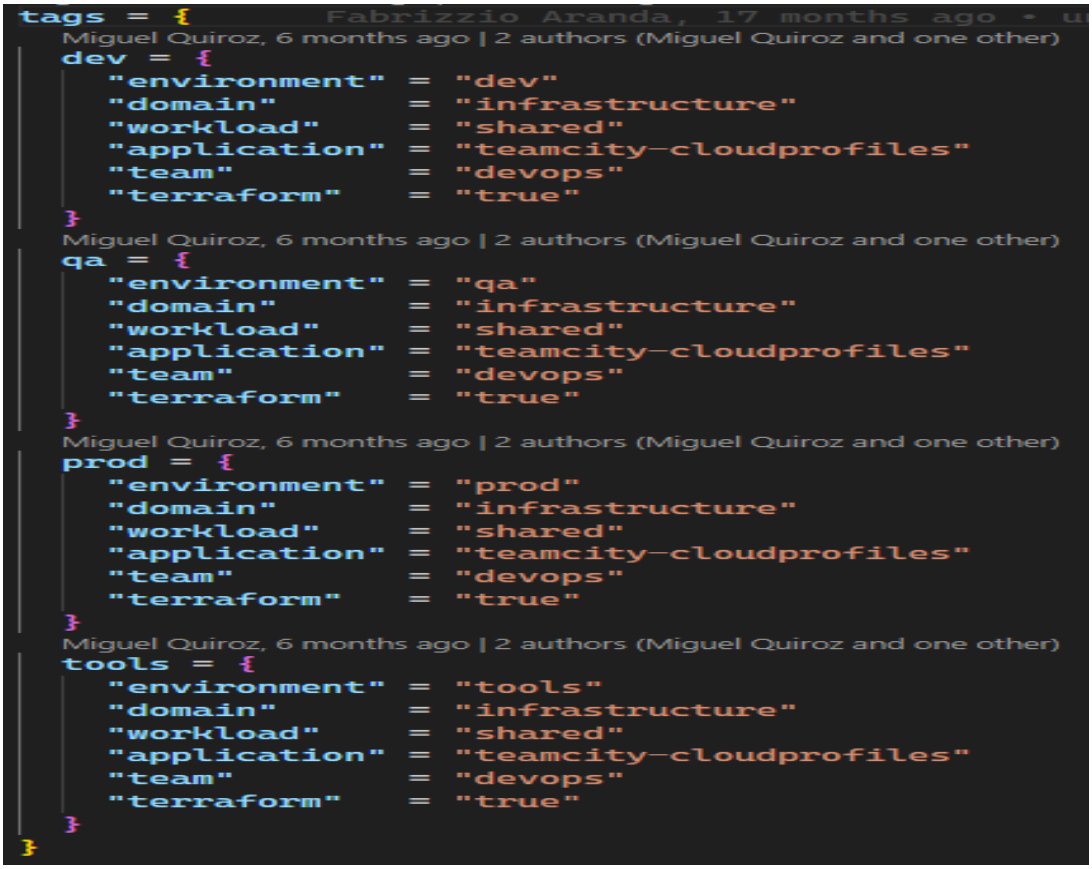
Código de Terraform que correlaciona la instancia creada con la AMI



Asimismo, creamos las AMI por ambiente, dado que de esta manera nos es más sencillo probar algún cambio nuevo en el código primero sin dañar nuestras operaciones o causar tiempo de espera. Una vez que la imagen haya sido creada y probada en el ambiente de desarrollo, se procede a crear para el ambiente productivo. Los valores se manejan a través del archivo *terraform.tfvars*, y para identificar cada ambiente se utiliza la característica de Terraform llamada workspaces, que nos permite separar el Terraform state de acuerdo con el workspace en el que nos encontremos, dándole aún más granularidad al desarrollo. En la Figura 25 se muestra un ejemplo de los valores empleados por ambiente.

Figura 25

Código de Terraform que ejemplifica la separación de valores por ambiente



Por otra parte, en la Figura 26 podemos apreciar un extracto del código de Bash usado para provisionar la instancia Linux.

Figura 26

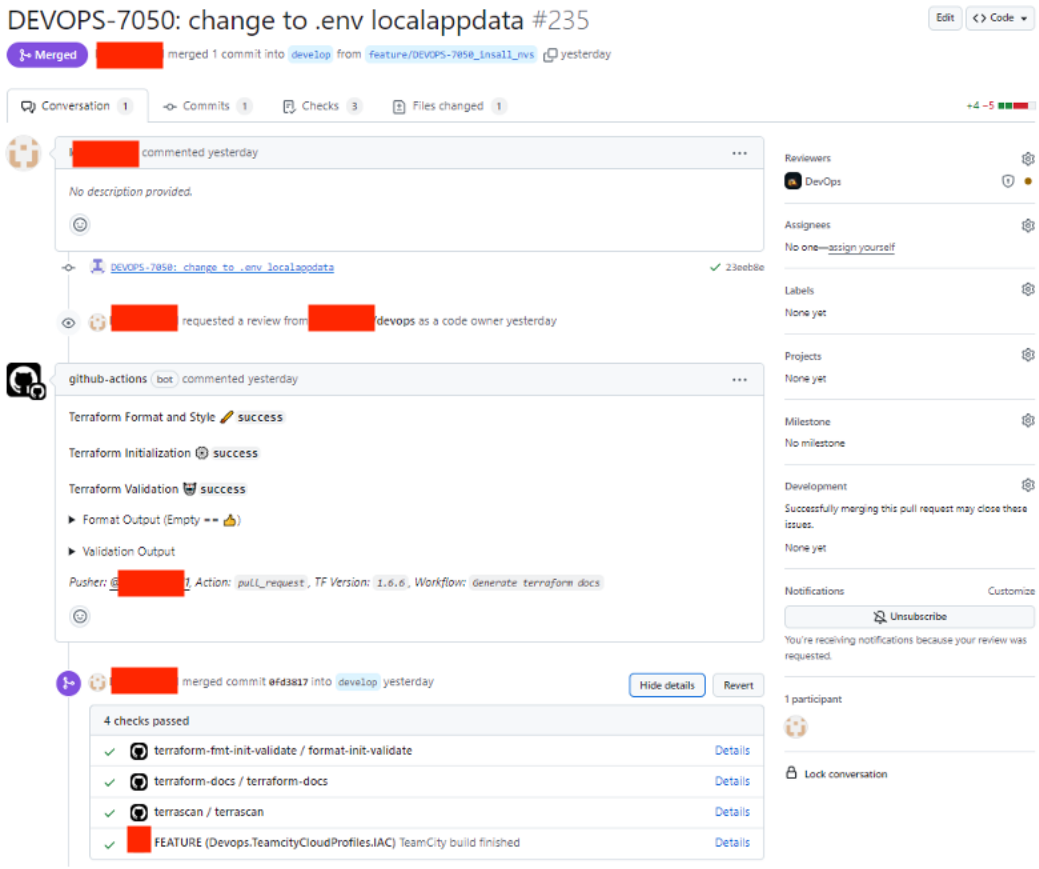
Extracto de código en bash para provisionar las instancias

```
user-data.sh
# Execute command and handle errors
execute_command() {
    "$@" || handle_error "Error executing the command: $*"
}

AZ_REPO=$(lsb_release -cs)
NODE_MAJOR=18

execute_command sudo apt-get update
execute_command sudo apt-get install -y ca-certificates python2 curl gnupg dirmngr lsb-release zip default-jre wget apt-transport-https software-properties-common || handle_error "Error downloading zip,mono and/or java"

execute_command sudo install -m 0755 -d /etc/apt/keyrings
execute_command sudo mkdir -p /home/devops
execute_command curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg || handle_error "Error downloading Docker key"

execute_command sudo chmod a+r /etc/apt/keyrings/docker.gpg
execute_command echo \
  "deb [arch=\"$(dpkg --print-architecture)\" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null || handle_error "Error al agregar el repositorio de Docker"

execute_command wget -q "https://packages.microsoft.com/config/ubuntu/$(lsb_release -rs)/packages-microsoft-prod.deb"
execute_command sudo dpkg -i packages-microsoft-prod.deb || handle_error "Error downloading Microsoft package"

execute_command rm packages-microsoft-prod.deb

execute_command curl -sLS https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor | sudo tee /etc/apt/keyrings/microsoft.gpg > /dev/null
if [ $? -ne 0 ];then
 handle_error "Error downloading Microsoft key"
fi
execute_command sudo chmod go+r /etc/apt/keyrings/microsoft.gpg

execute_command echo "deb [arch=\"$(dpkg --print-architecture)\" signed-by=/etc/apt/keyrings/microsoft.gpg] https://packages.microsoft.com/repos/azure-cli/ $AZ_REPO main" | sudo tee /etc/apt/sources.list.d/azure-cli.list || handle_error "Error adding the Azure CLI rep

execute_command curl -sSfL https://apt.octopus.com/public.key | sudo apt-key add -
execute_command sudo sh -c "echo deb https://apt.octopus.com/ stable main > /etc/apt/sources.list.d/octopus.com.list" || handle_error "Error adding octopus repository"

execute_command curl -fsSL https://deb.nodesource.com/gpgkey/nodesource-repo.gpg.key | sudo gpg --dearmor -o /etc/apt/keyrings/nodesource.gpg
execute_command echo "deb [signed-by=/etc/apt/keyrings/nodesource.gpg] https://deb.nodesource.com/node_$NODE_MAJOR.x nodistro main" | sudo tee /etc/apt/sources.list.d/nodesource.list

execute_command sudo apt-get update || handle_error "Error updating repositories info"
#### Install Docker #####
execute_command sudo apt-get install -y docker-ce=5:25.0.2-1~ubuntu.20.04~focal docker-ce-cli=5:25.0.2-1~ubuntu.20.04~focal containerd.io || handle_error "Error installing Docker"
######################

##### Install Docker compose ############
execute_command curl -L "https://github.com/docker/compose/releases/download/v2.24.5/docker-compose-$(uname -s)-$(uname -m)" -o docker-compose || handle_error "Error downloading docker compose"
execute_command chmod +x docker-compose
execute_command mv docker-compose /usr/bin/ || handle_error "Error adding docker-compose to env PATH"
########################################

execute_command sudo useradd devops -d /home/devops

execute_command groupadd docker
execute_command usermod -aG docker devops
execute_command newgrp docker

####### Install powershell 7.2.1
execute_command sudo wget https://github.com/PowerShell/PowerShell/releases/download/v7.2.1/powershell-lts_7.2.1-1.deb_amd64.deb || handle_error "Error download package powershell 7.2.1"
execute_command sudo dpkg -i powershell-lts_7.2.1-1.deb_amd64.deb || handle_error "Error installing powershell version 7.2.1"
execute_command sudo rm -rf powershell-lts_7.2.1-1.deb_amd64.deb || handle_error "Error deleting package powershell 7.2.1"
#######
```

Con lo cual, después de definir una nueva versión de software a instalar en el agente o algún cambio requerido en la configuración de la creación de la imagen base, se procede a crear un Pull Request en el repositorio como se muestra en la Figura 27.

Figura 27

Vista general del Pull Request con los cambios



Una vez que el Pull Request haya sido revisado y aprobado, se hace la fusión hacia la rama develop, en donde el pipeline de integración continua se encarga de verificar que no haya errores en el código y genera un artefacto, que luego será usado por el pipeline de despliegue con el fin de crear los recursos en la nube de AWS, así como puede ser apreciado en las Figuras 28, 29 y 30.

Figura 28

Pipeline de integración continua para el ambiente de desarrollo

Figura 29

Pipeline de despliegue para el ambiente de desarrollo

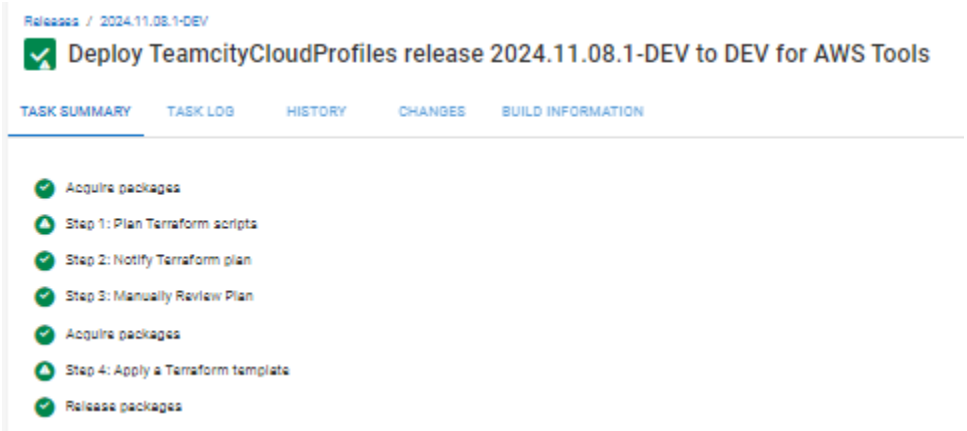
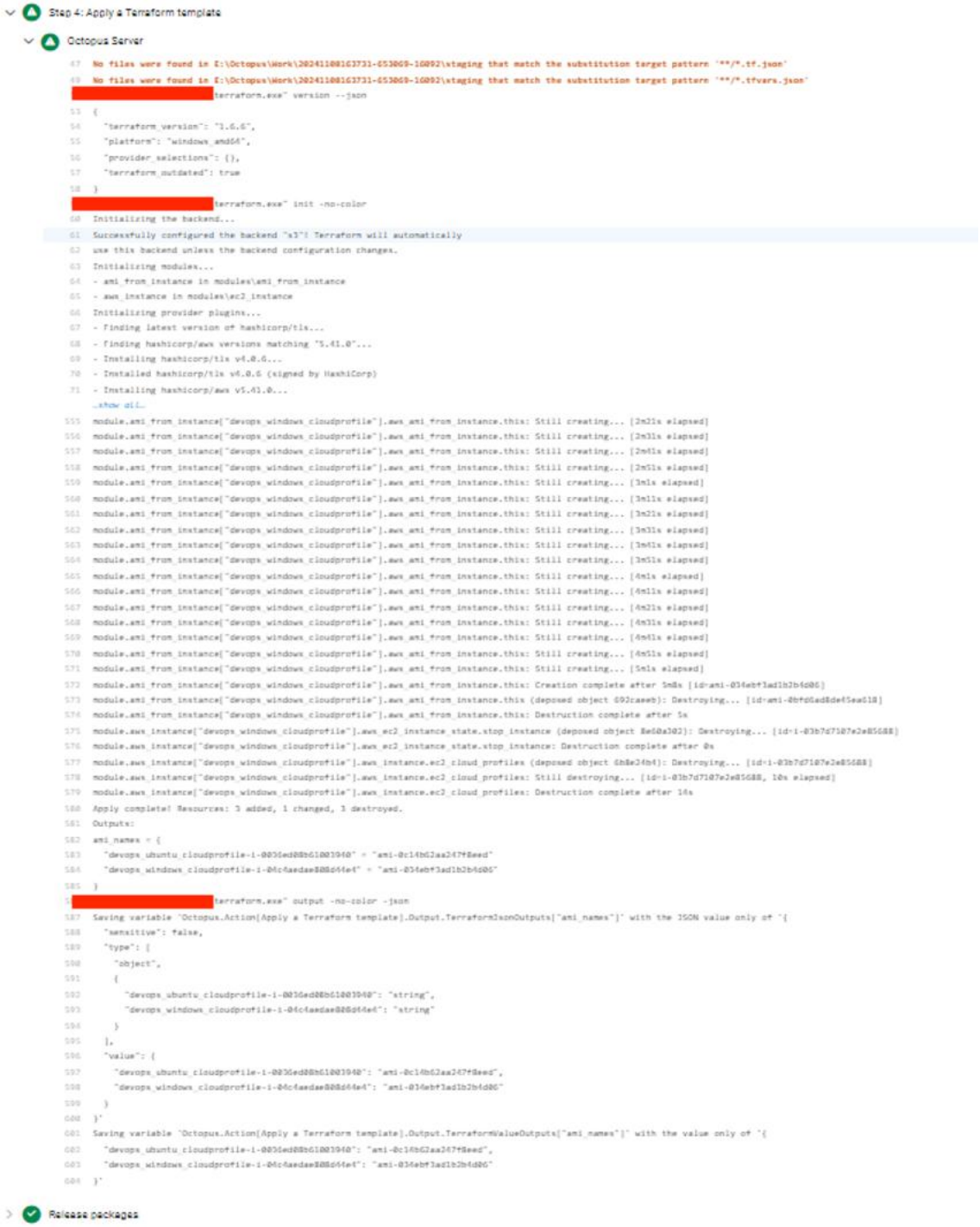


Figura 30

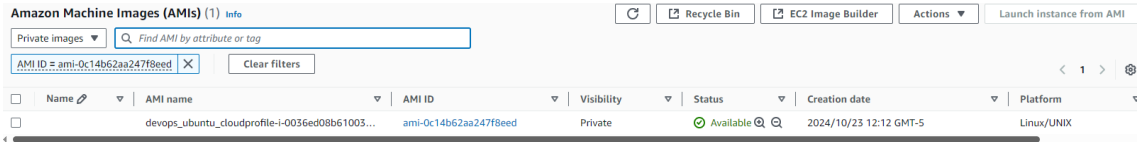
Logs del pipeline de despliegue confirmando la creación de los recursos en AWS



Después de la ejecución sin errores de los pipelines de integración y entrega continua, se procede a revisar que la imagen haya sido creada correctamente en la nube de AWS, mostrado en la Figura 31.

Figura 31

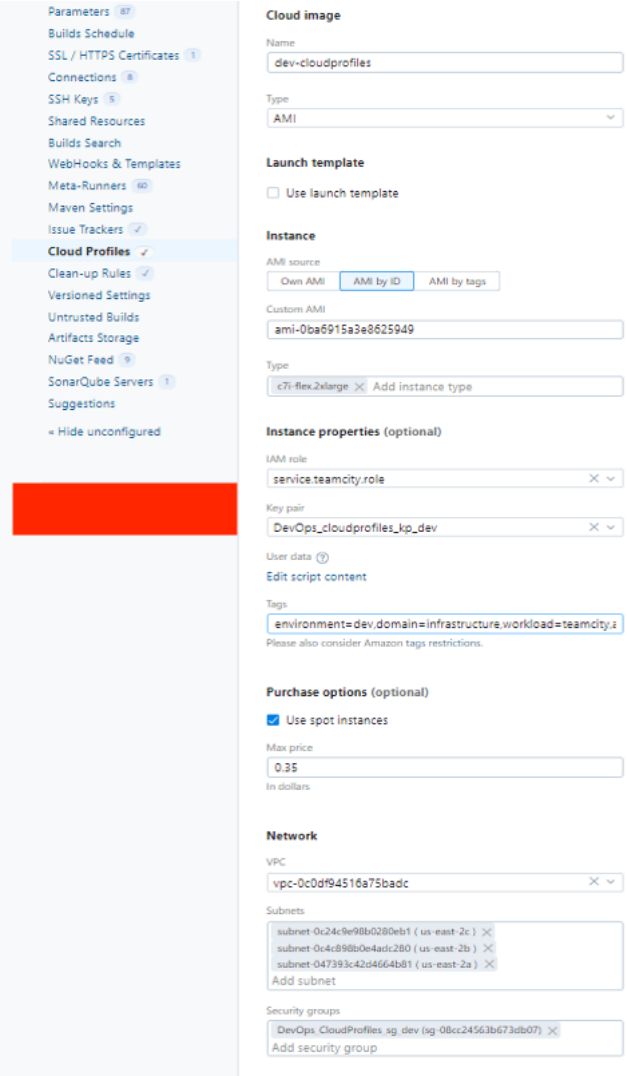
Verificación de la AMI creada en AWS



Posteriormente, en la Figura 32, se completa la información requerida por la característica Cloud Profiles de Teamcity para poder crear agentes a demanda utilizando la imagen creada mediante Terraform.

Figura 32

Configuración de la característica Cloud Profiles con la AMI creada

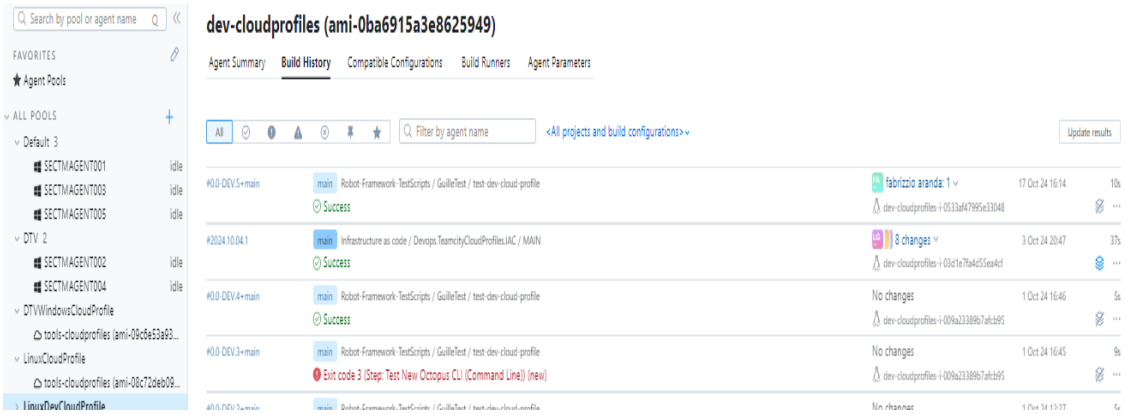


En consecuencia, una vez guardado los cambios se procede a probar diferentes trabajos de construcción con la nueva configuración de los Cloud Profiles para verificar que la instancia esté funcionando correctamente. Luego de que una los trabajos de construcción pasen satisfactoriamente, como se muestra en la Figura 33, se puede considerar como estable el cambio en la AMI de Amazon, y se procede a realizar el mismo procedimiento para el entorno

productivo.

Figura 33

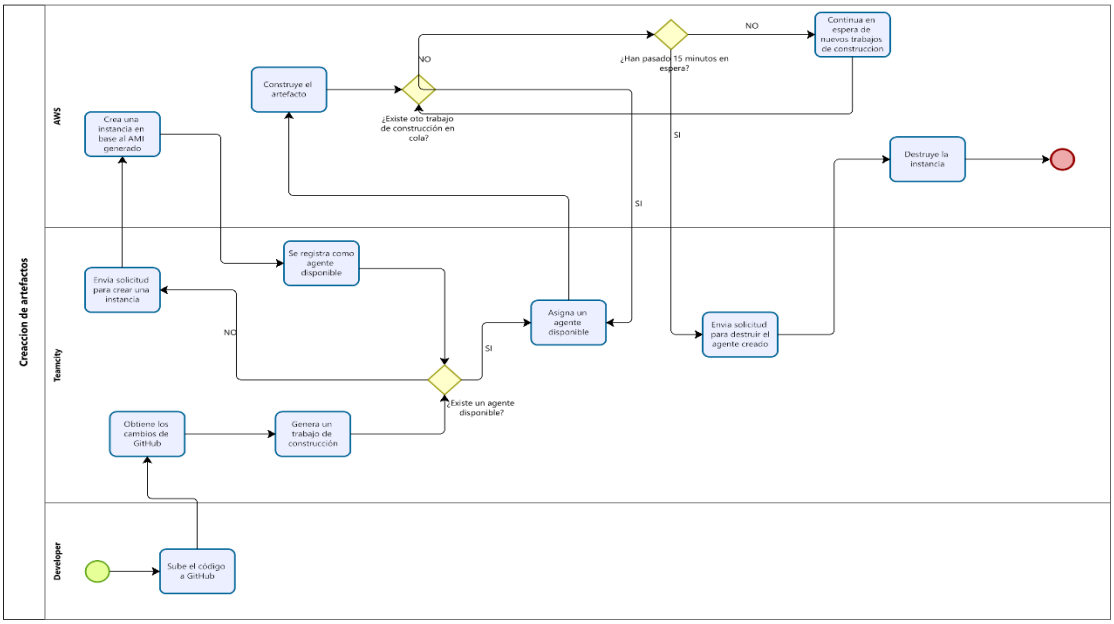
Ejecución de agentes en la nube mediante la característica de Cloud Profiles



Por lo que el flujo de trabajo final, bajo la implementación de la característica de Cloud Profiles de Teamcity, quedaría retratado en la Figura 34.

Figura 34

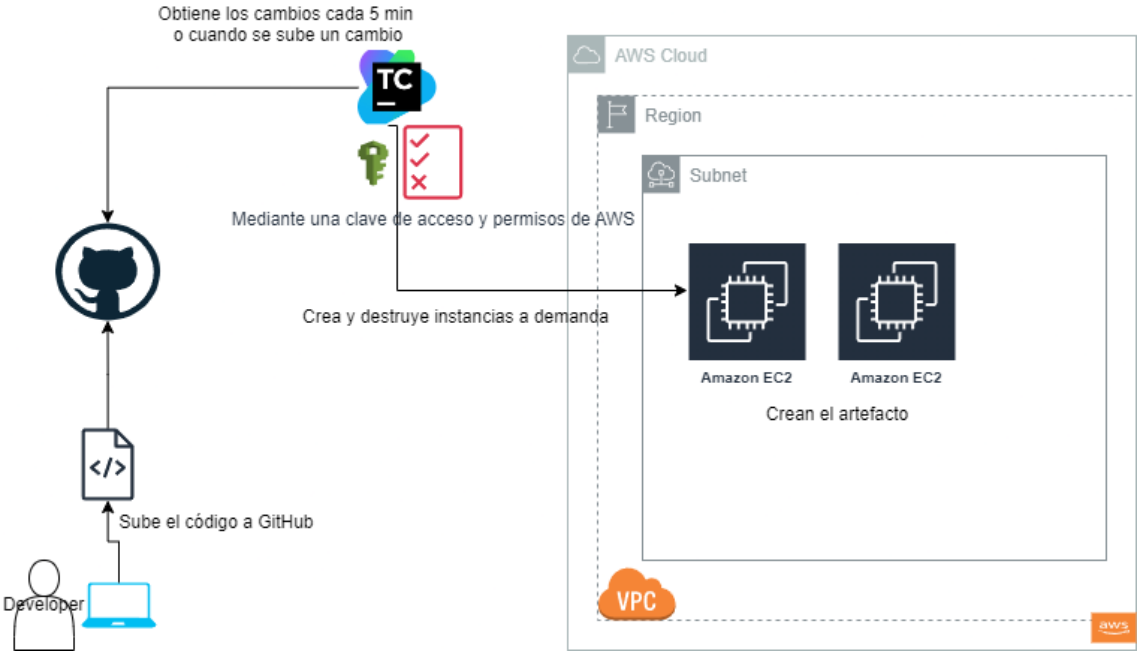
Flujo de trabajo luego de implementar la característica Cloud Profiles



A su vez, en la Figura 35, se refleja la arquitectura final utilizada para el proceso de creación de agentes desde el punto de vista del desarrollador, en donde se puede apreciar que una vez el código con los nuevos cambios ha sido subido al repositorio, en este caso GitHub, se encuentra disponible para que Teamcity obtenga dichos cambios y genere las instancias en caso de ser necesario para la construcción del artefacto.

Figura 35

Arquitectura final de los Cloud Profiles



CAPÍTULO IV. RESULTADOS

Resultado 1: Reducir los costos de los agentes de construcción alojando la infraestructura en AWS para optimizar el proceso de provisionamiento de agentes de construcción en el área de TI implementando AWS y CI/CD en la empresa.

Antes de la implementación, el costo de los agentes de construcción alojados en la nube de Microsoft Azure era elevado dado que estaban disponibles todo el tiempo, generando un gasto innecesario a la empresa dado que, generalmente no solían usarse fuera del horario laboral o en los fines de semana.

Se realiza la comparativa de los costos previos existentes que conllevaban mantener los agentes de construcción operativos en Azure contra el costo actual gracias a la implementación de AWS y los Cloud Profiles de Teamcity.

Tabla 3

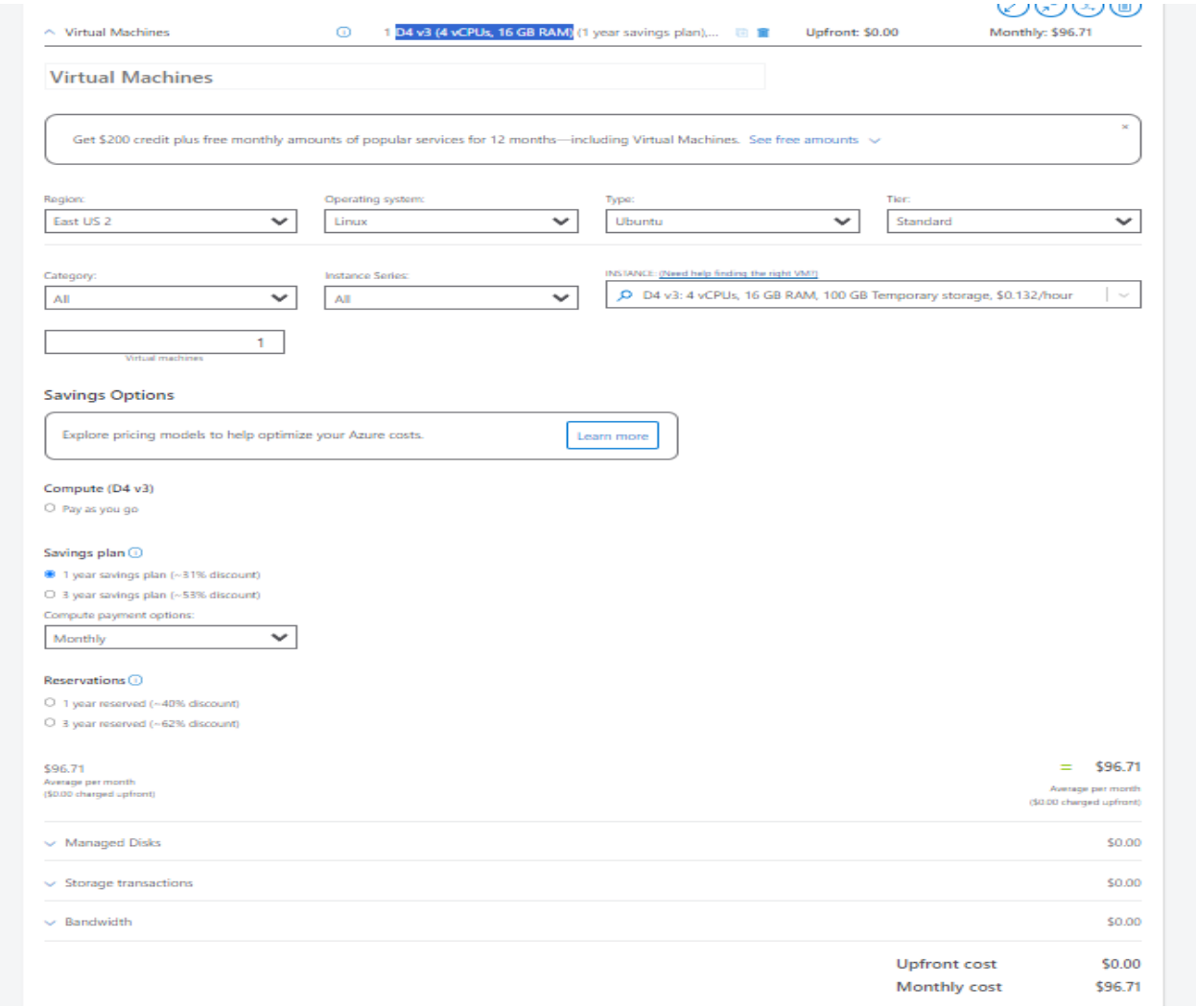
Comparativa en base a las instancias en los proveedores de nube

Características\Proveedores	Azure	Amazon Web Services
Tipo de instancia	Standard D4 v3 (1-year savings plan)	C5.4xLarge (Spot instances)
Capacidad de disco	100 GB	100 GB
Horas activo	24	A demanda
Cantidad de instancias	3	6

Anteriormente, en Azure, para cada agente se utilizaba una máquina virtual del tipo Standard D4 v3(4vcpu, 16 GB RAM) con un almacenamiento de 100GB usando un plan de 1 año de reserva, sí utilizamos la calculadora de Azure, un aproximado del costo por mes sería de $96.71 USD por instancia, como se puede apreciar en la Figura 36.

Figura 36

Cálculo estimado del costo mensual de los agentes de Teamcity en Azure

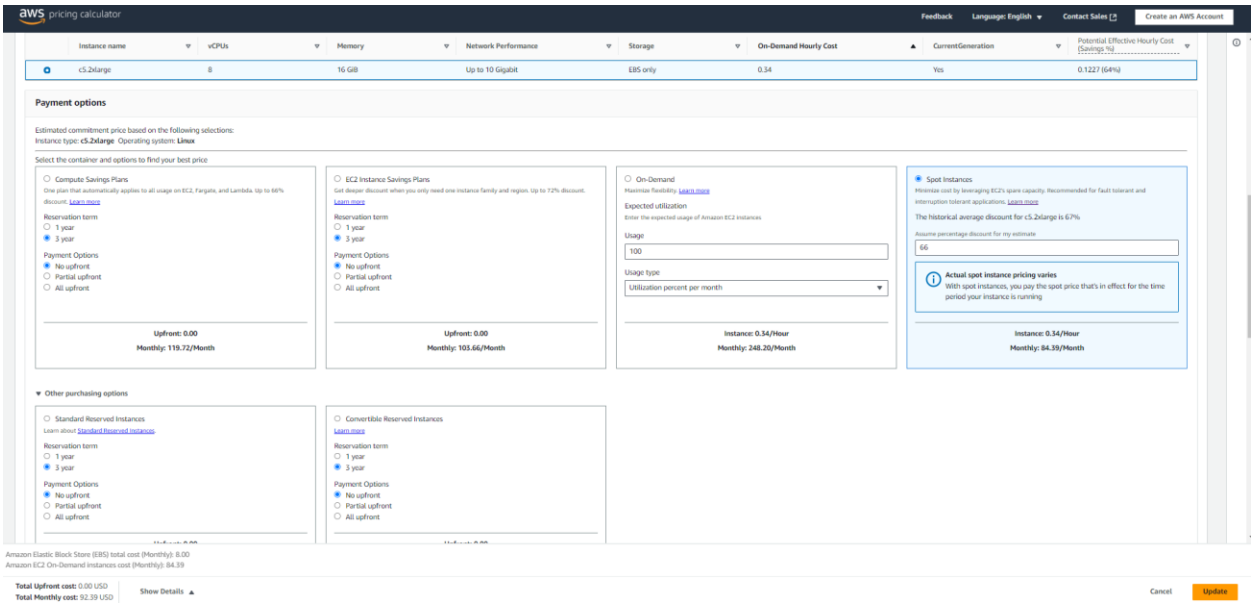


Realizando una comparativa con el tipo de instancia Spot equivalente en Amazon Web

Services, C5.2xlarge (8vcpu 16GB), aun estando encendida todo el mes, se puede apreciar que su costo aproximado es de $92.39 USD por instancia, siempre y cuando se use las 24 horas del día, todos los días del mes.

Figura 37

Cálculo estimado del costo mensual de los agentes en AWS



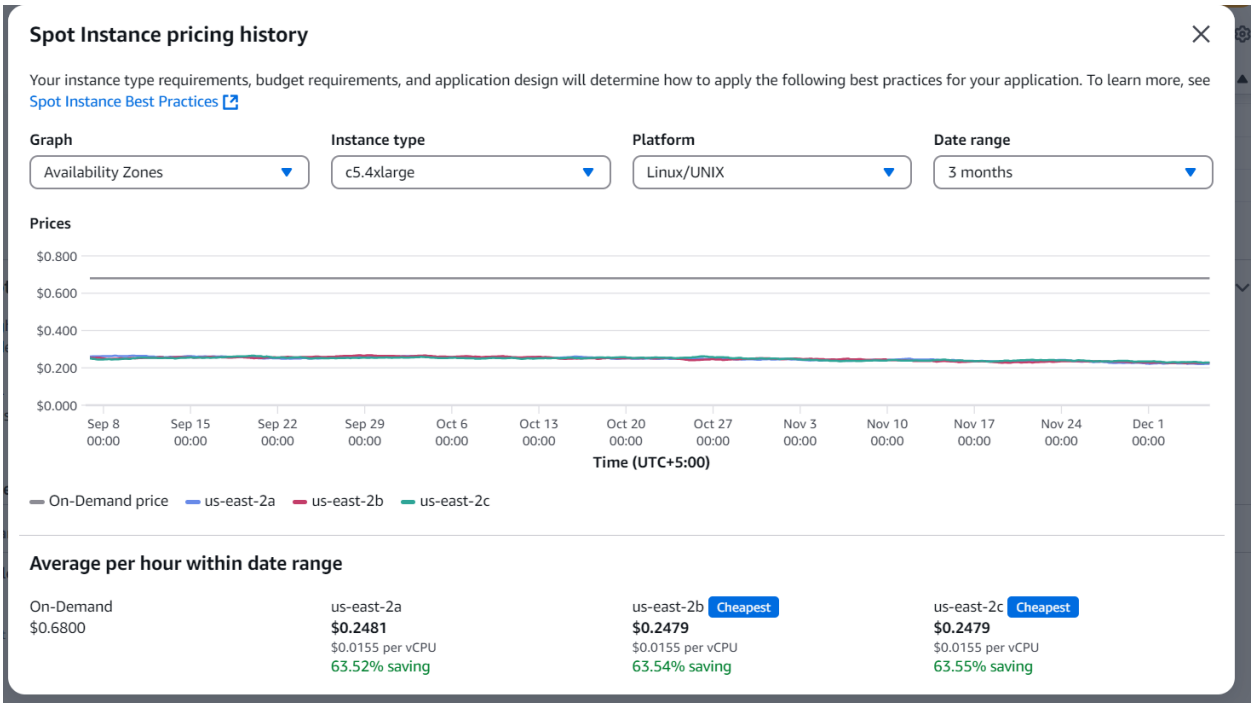
Sin embargo, este no es el caso dado que estas instancias son a demanda, por lo cual, en un caso hipotético de mucha demanda, se asume que podrían estar encendidas 12 horas seguidas, con lo cual, el número hipotético de horas por mes sería de 365 horas. Por consiguiente, el cálculo tomando como referencia la zona de disponibilidad más cara de las instancias spot: 365 x 0.1195 + 8(almacenamiento), nos daría un total de $51.61 USD, aun mucho más barata como estaba alojada en Azure.

Debido a la diferencia en la eficiencia de precio, optamos por ampliar aún más la capacidad de nuestras instancias, pasando al tipo de instancia C5.4xlarge, la cual posee una

capacidad de 16vcpus y 32 GB de RAM. El costo de las instancias spot para este tipo de máquinas es de $0.2481/h la más cara y $0.2479/h la más barata, como se puede apreciar en la Figura 38.

Figura 38

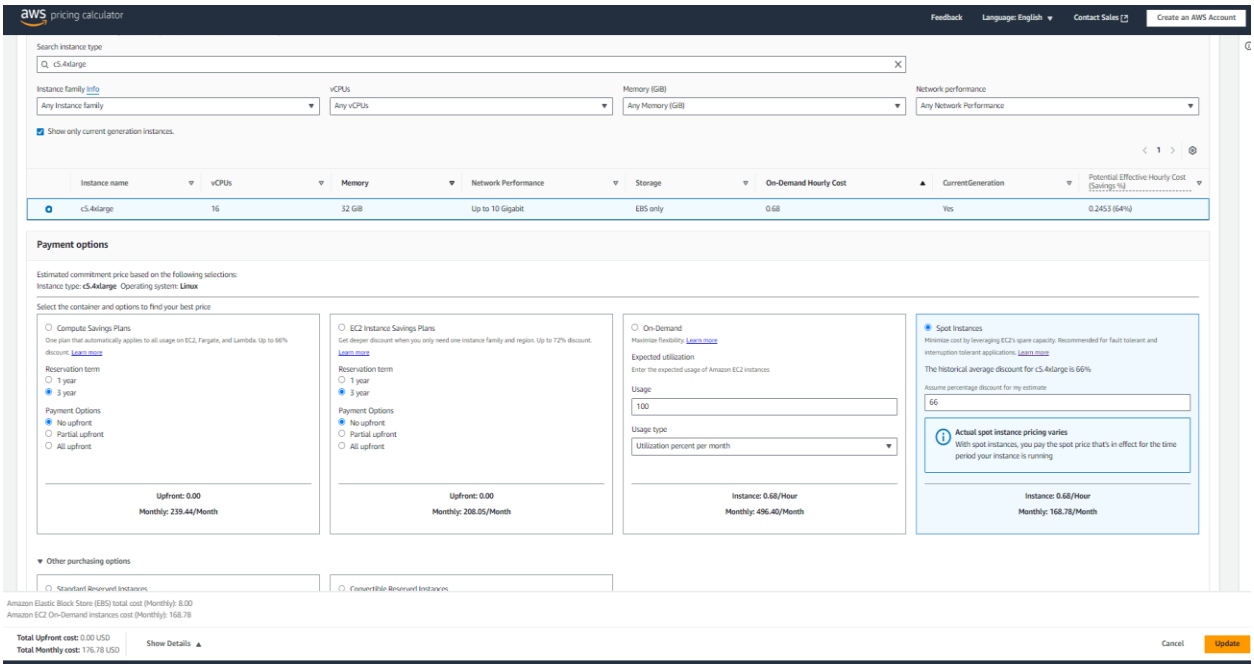
Histórico de precios de las instancias spot en AWS por zona de disponibilidad



Según esta premisa, este tipo de instancias nos debería de costar mensualmente un aproximado de $176.78 USD

Figura 39

Cálculo aproximado del coste mensual para una instancia spot en AWS

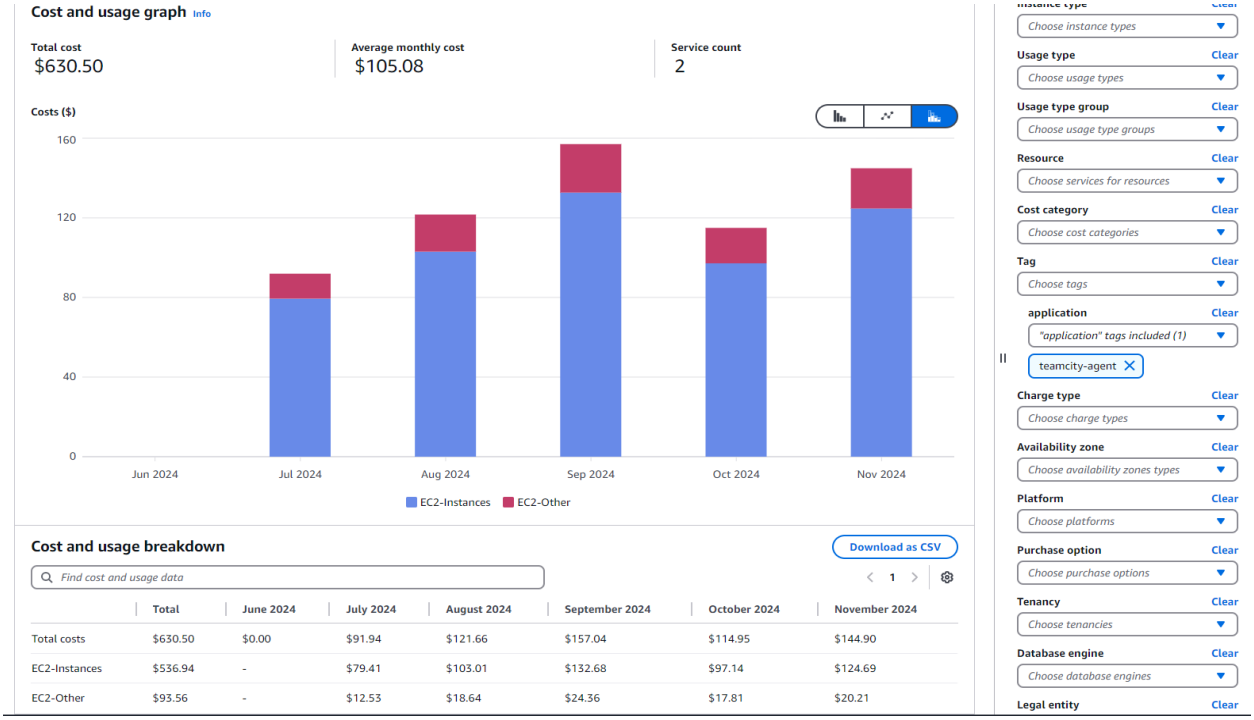


Sin embargo, en base a nuestros datos de uso en la empresa y mediante la herramienta de exploración de costos de AWS, podemos determinar que no se cumple dicho precio

Tomando en cuenta los costos de los 6 últimos meses, se aprecia que, en promedio, por una cantidad máxima de 6 agentes en simultaneo, nos genera un coste de $105 USD mensuales, esto quiere decir, que en promedio el mantenimiento de un agente en AWS nos debería estar costando $17.5 USD mensuales, siempre y cuando se consideren los 6 agentes en ejecución.

Figura 40

Costo de los agentes de Teamcity en AWS de los últimos 6 meses

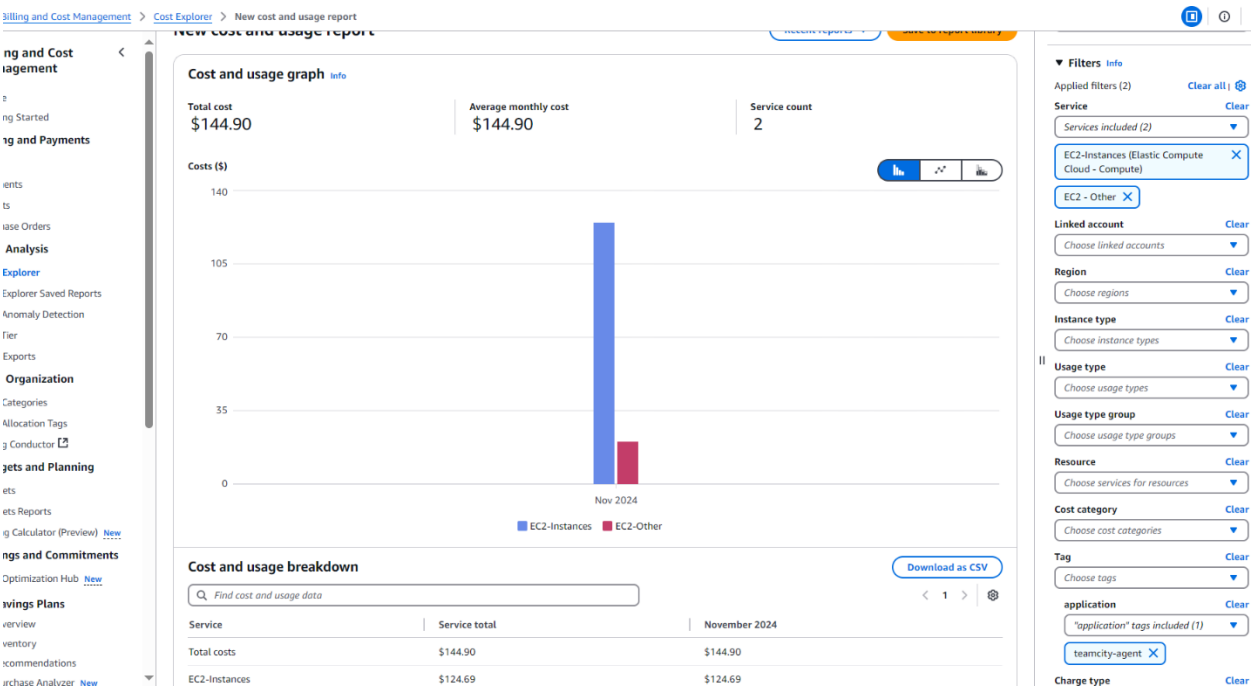


Por supuesto que esto puede variar de acuerdo con el uso de cada agente, la cola de ejecución y el tiempo de construcción que demore el proyecto asignado al agente, y si hay otros proyectos en cola que programen su ejecución en ese mismo agente. Para desglosarlo aún más, tomaremos las medidas del último mes efectivo

Para el mes de noviembre, el costo total de los agentes fue de $144.90 USD por el total, mientras que, si tomamos en consideración los costos de Azure, por el total de 3 agentes deberíamos estar pagando un aproximado de $290.13 USD. Con lo cual podemos afirmar que, gracias a la implementación de los agentes a demanda en AWS, para el mes de noviembre se ha logrado ahorrar $145.23.

Figura 41

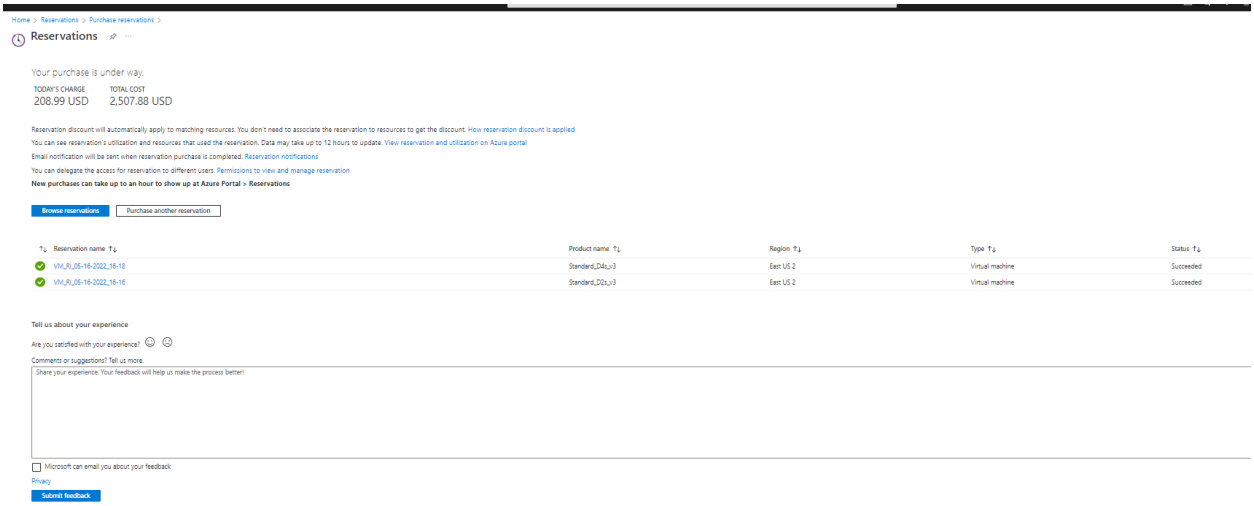
Cálculo de los agentes de TeamCity en AWS para el mes de noviembre



Si lo llevamos a una perspectiva semestral, el aproximado de uso de los agentes de Azure por 6 meses para 3 agentes sería un total de 96.71 x 6 x3 = $1740.78 USD, considerando que los 3 sean el mismo tipo de instancia. Se puede apreciar una correlación en base a un estimado en el cálculo del precio anual por las instancias reservadas como se observa en la Figura 42.

Figura 42

Costo anual estimado para los agentes de Teamcity en Azure

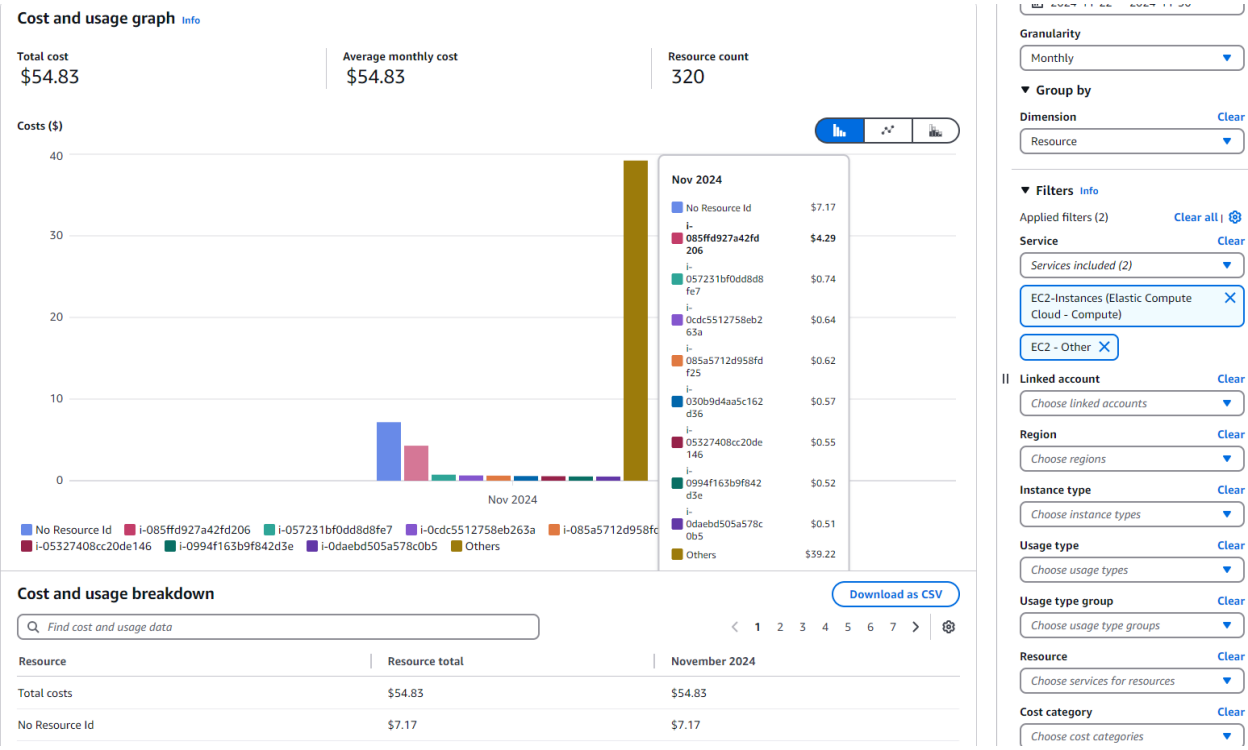


Por lo que, comparándolo con el costo de 6 meses de los agentes en AWS como se muestra en la Figura 40, la estimación anual para los agentes en AWS sería de $1261 USD, por lo cual, se puede afirmar que se obtuvo aproximadamente una reducción de $1246.98 USD.

También podemos afirmar que esta reducción de costos es debido a la reducción del tiempo de actividad de cada uno de los agentes, como se puede apreciar en la Figura 43, los agentes son creados y destruidos, generando costos mínimos a su paso.

Figura 43

Cálculo de los costos de los agentes de Teamcity en los últimos 14 días en AWS



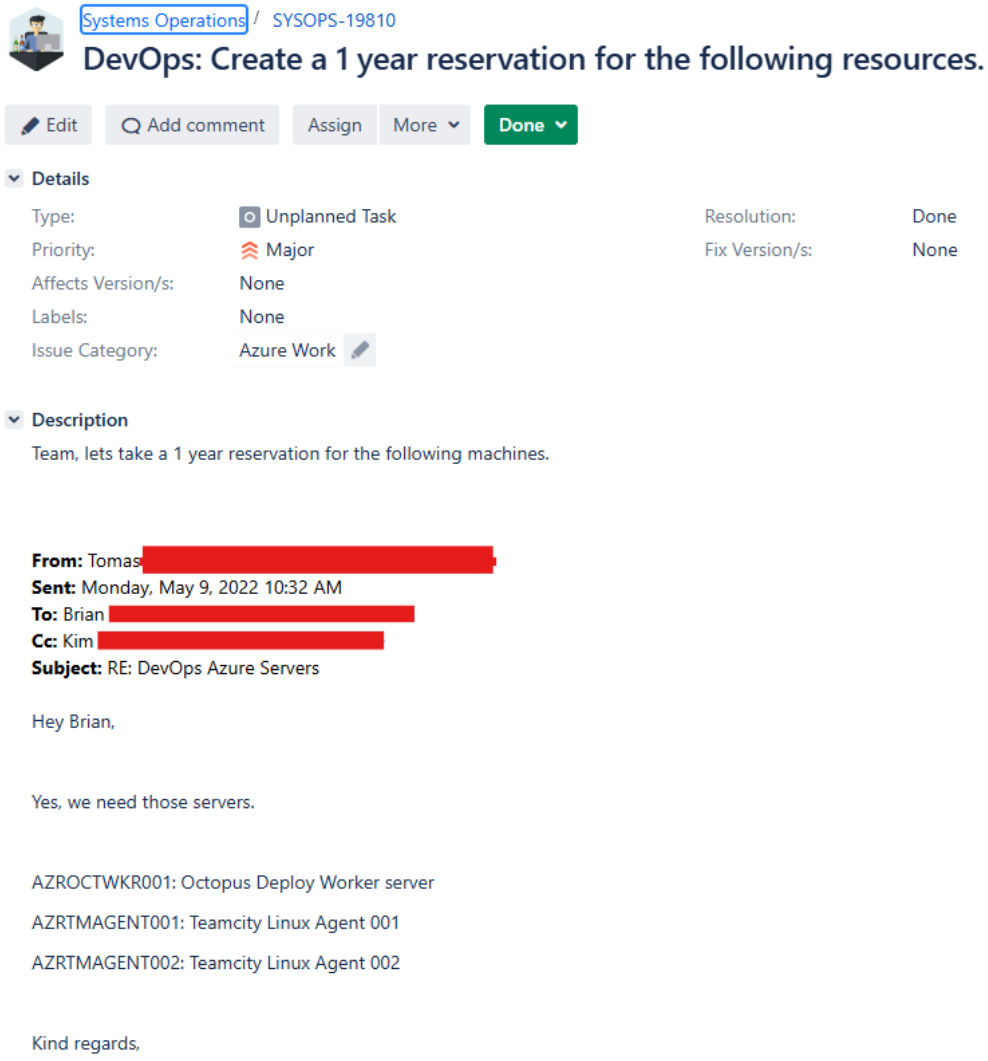
Resultado 2: Reducir el tiempo de provisionamiento y despliegue de los agentes de Teamcity para optimizar el proceso de provisionamiento de agentes de construcción en el área de TI implementando AWS y CI/CD en la empresa.

Previo a la implementación, el equipo de DevOps debía crear un ticket de Jira hacia el equipo de SysOps para que puedan crear una máquina virtual en Azure como se puede apreciar en la Figura 44, para luego obtener acceso mediante SSH y provisionarla con las herramientas necesarias para la construcción de artefactos de los diferentes proyectos, así como también, instalar y registrar el agente de Teamcity en la instancia, lo cual, en palabras del jefe del área de

DevOps, podría llegar a tomar hasta 1 día laboral por instancia.

Figura 44

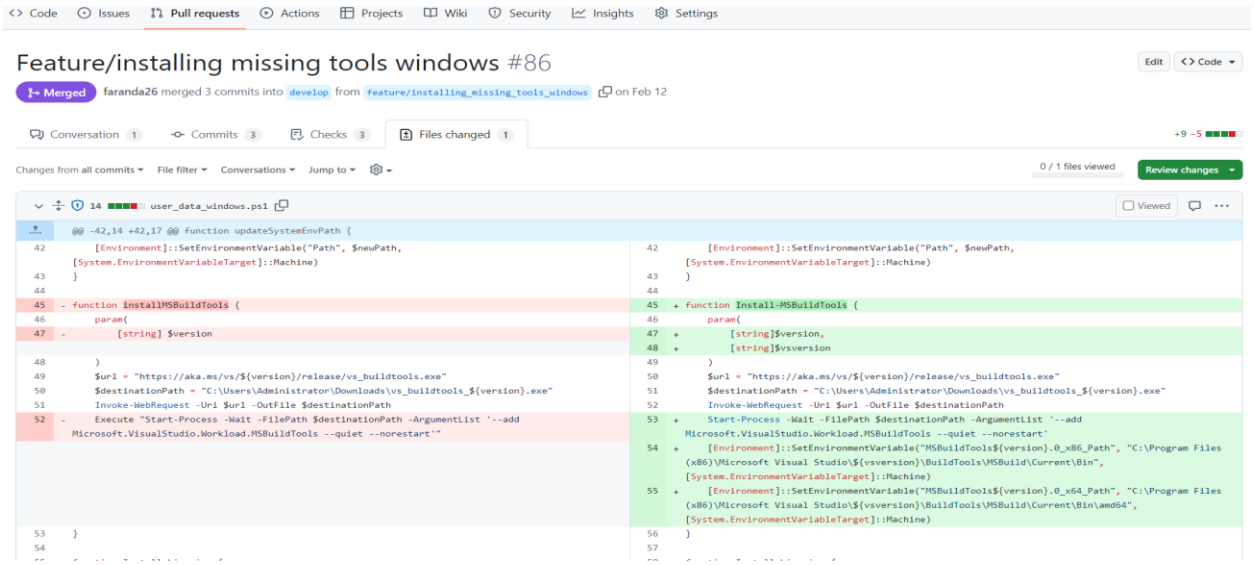
Ticket para la creación de nuevas instancias a SysOps



En la actualidad, solo el equipo de DevOps se encarga completamente de la creación y mantenimiento de la instancia mediante infraestructura como código y el repositorio de GitHub, como se observa en la Figura 45.

Figura 45

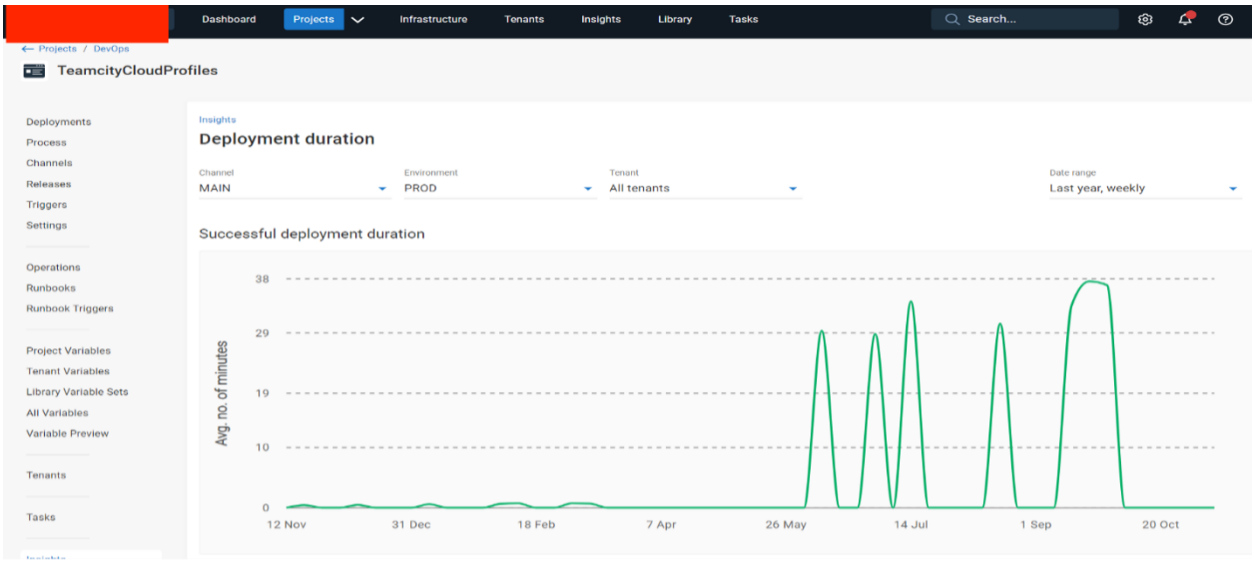
Registro de los cambios realizados en el código alojado en GitHub



De esta manera, desde su fecha de implementación, obtener un tiempo de despliegue promedio de 30 minutos, como se puede ver en la Figura 46.

Figura 46

Promedio del tiempo de creación de la AMI para los agentes de Teamcity



En adición a esto, gracias a tener el proceso de creación y provisionamiento automatizado, se pudo reducir considerablemente el tiempo de disponer de un nuevo agente, así como también se redujo el involucramiento de otros equipos al mínimo. Se obtuvo un tiempo de espera de aproximadamente 2 minutos cuando no existe un agente encendido como se puede apreciar en la Figura 47 y de aproximadamente 15 segundos cuando el agente está en espera, como se muestra en la Figura 48.

Figura 47

Tiempo de espera de creación y asignación de un agente por primera vez

Figura 48

Tiempo de espera de asignación de un agente existente

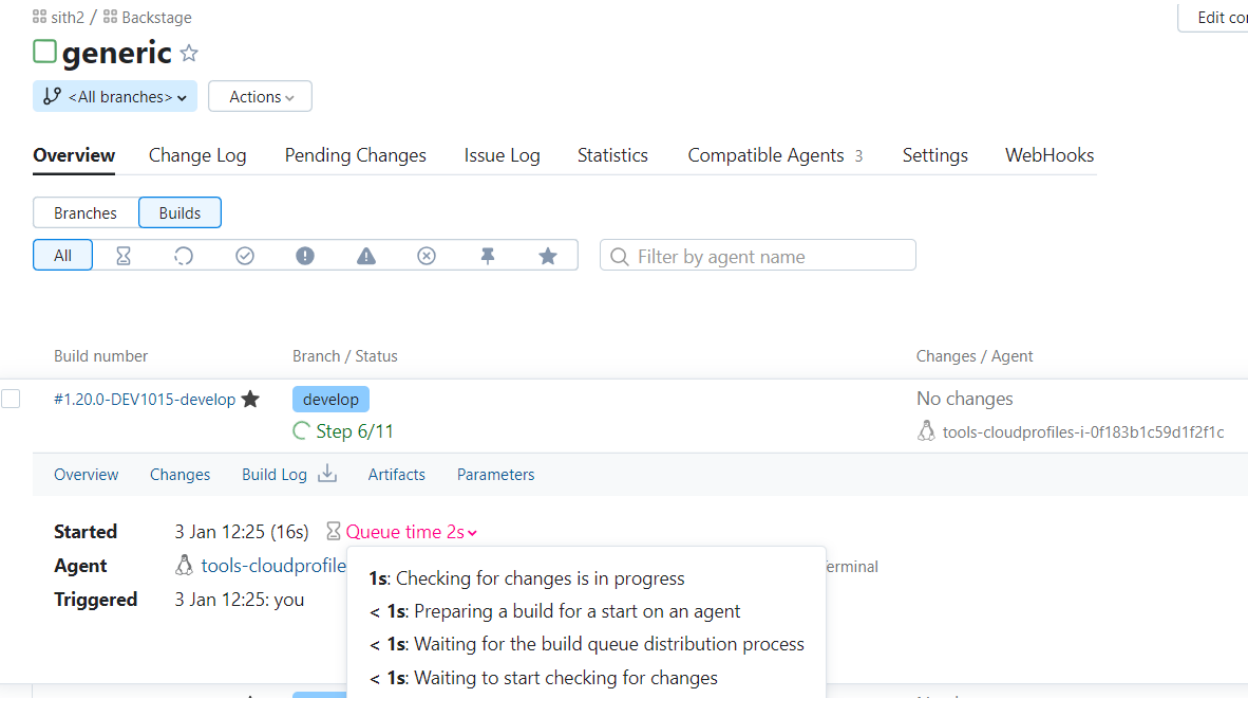


Tabla 4

Comparativa entre implementaciones

Variables/Etapas	Anteriormente	Actualidad
Plataforma	Azure	AWS
Provisionamiento de herramientas en los agentes	Método manual	Usando IAC
Tiempo de habilitación de los agentes (creación más instalación de herramientas)	1 día aproximado por agente	30 min aproximado para la imagen base, luego la creación de los agentes demora 2 min promedio por agente.

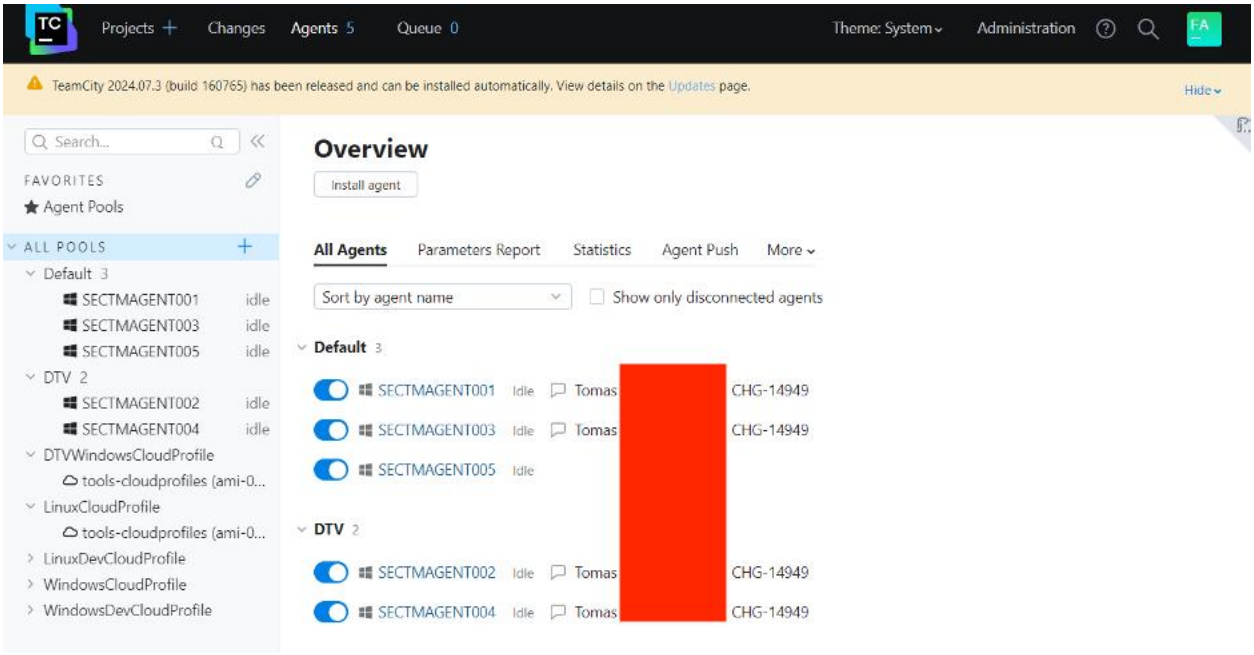
Cantidad de agentes	3	6
Tiempo encendido	24/7	15 min máximo si no hay trabajos en ejecución

Resultado 3: Reducir las vulnerabilidades de seguridad para optimizar el proceso de provisionamiento de agentes de construcción en el área de TI implementando AWS y CI/CD en la empresa.

Anteriormente en Azure, se creaban las máquinas virtuales con un sistema operativo base, y se tenía que actualizar manualmente tanto el sistema operativo como las herramientas instaladas, usualmente mediante change cards, como se muestra en la Figura 49. Siendo esto considerado como un proceso lento y tedioso para el equipo encargado de realizarlo, además de agregarles la responsabilidad de estar al tanto de los últimos parches de seguridad.

Figura 49

Cambios manuales por maquina mediante change card, indicada como comentario



Amazon nos brinda Amazon Machine Image (AMI) con los últimos parches de seguridad, liberándonos de la responsabilidad de estar actualizando el sistema operativo. Actualmente, mediante el código de Terraform mostrado en la Figura 50, obtenemos la última versión de la distribución disponible del sistema operativo en AWS. Por otra parte, podemos actualizar las versiones de las herramientas de una manera sencilla utilizando IAC y el script de bash mostrado en la Figura 51.

Figura 50

Código de Terraform con la última versión de la AMI en base al parámetro obtenido

```
data "aws_ami" "ubuntu" {
  most_recent = true

  filter {
    name   = "name"
    values = var.aws_ami_filter_name_values
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }

  owners = var.aws_ami_owners
}

resource "aws_instance" "ec2_cloud_profiles" {
  ami = data.aws_ami.ubuntu.id

  cpu_options {
    core_count       = try(var.cpu_options_core_count, null)
    threads_per_core = try(var.cpu_options_threads_per_core, null)
  }
```

Figura 51

Extracto del script de bash utilizado para instalar las herramientas

```
############## PODMAN ##############
##### Install Podman ####################
execute_command . /etc/os-release
execute_command echo "deb https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/xUbuntu_${VERSION_ID}/ /" | sudo tee /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list
execute_command curl -L "https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/xUbuntu_${VERSION_ID}/Release.key" | sudo apt-key add -
execute_command sudo apt-get -y update
execute_command sudo apt-get -y upgrade
execute_command sudo apt-get -y install podman || handle_error "Error installing Podman"
execute_command cd /home/devops
#####################################

########## podman as docker ############
#execute_command sudo mv /usr/bin/docker /usr/bin/docker-original
#execute_command sudo ln -s /usr/bin/podman /usr/bin/docker
#####################################

### Setup Podman with Docker Compose  #####
#execute_command systemctl start podman.socket
#execute_command export DOCKER_HOST=unix:///run/user/$UID/podman/podman.sock
#########################################
############## PODMAN ##############

######## Install powershell 7.2.1
execute_command sudo wget https://github.com/PowerShell/PowerShell/releases/download/v7.2.1/powershell-lts_7.2.1-1.deb_amd64.deb || handle_error "Error download package powershell 7.2.1"
execute_command sudo dpkg -i powershell-lts_7.2.1-1.deb_amd64.deb || handle_error "Error installing powershell version 7.2.1"
execute_command sudo rm -rf powershell-lts_7.2.1-1.deb_amd64.deb || handle_error "Error deleting package powershell 7.2.1"
########

####### Install terraform 1.6.6
execute_command wget https://releases.hashicorp.com/terraform/1.6.6/terraform_1.6.6_linux_amd64.zip || handle_error "Error downloading terraform 1.6.6 zip file"
execute_command unzip terraform_1.6.6_linux_amd64.zip
execute_command sudo mv terraform /usr/local/bin/  || handle_error "Unable to move terraform binary to /usr/local/bin/"
execute_command rm -rf terraform_1.6.6_linux_amd64.zip
############################
```

A su vez, AWS nos permite conectarnos a las instancias mediante Session Manager, reduciendo así la necesidad de habilitar puertos SSH y manejar llaves privadas. Solo los usuarios con los permisos adecuados pueden conectarse a las instancias, señalado en la Figura 52.

Figura 52

Conexión mediante AWS Session Manager

CAPÍTULO V. CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

En base al trabajo elaborado, se concluye que:

Se logró una optimización significativa en los costos operativos de los agentes de TeamCity al implementar Cloud Profiles y utilizar instancias spot de AWS. Este enfoque permitió un ahorro aproximado de $1246.98 USD, manteniendo un mayor número de agentes disponibles sin incurrir en gastos excesivos. Las instancias spot demostraron ser una solución eficiente, ya que su naturaleza bajo demanda permite que TeamCity las reemplace automáticamente en caso de ser reclamadas por AWS, asegurando la continuidad del servicio sin interrupciones. A lo largo del tiempo, incluso operando con la máxima capacidad de agentes, los costos mensuales se mantuvieron por debajo de los que se generarían con instancias permanentes, evidenciando la efectividad del modelo adoptado.

La adopción de infraestructura como código permitió reducir drásticamente el tiempo de configuración y disponibilidad de los agentes de TeamCity, pasando de casi un día a aproximadamente 30 minutos. Además, el versionado del código proporciona un historial detallado de cambios, lo que facilita la identificación y reversión de errores. Esta metodología ha optimizado la detección, configuración y despliegue de los agentes, incrementando la eficiencia del proceso.

Se implementaron controles de seguridad que eliminaron la necesidad de exponer el puerto SSH de las instancias gracias a AWS Systems Manager Session Manager. Además, se fortaleció la administración de accesos mediante IAM, asegurando que solo usuarios autorizados

puedan operar sobre las instancias. La infraestructura como código permitió centralizar el control del software instalado en los agentes, facilitando la actualización de versiones y parches de seguridad. Asimismo, herramientas de análisis estático como SonarQube y Terrascan permitieron detectar y corregir vulnerabilidades de manera temprana, mejorando la seguridad general del entorno.

5.2 Recomendaciones

En base a las lecciones aprendidas, se recomienda:

Monitorear el uso de las instancias en relación con los procesos de construcción que ejecutan, con el fin de estimar las capacidades necesarias y elegir el tamaño adecuado. Además, se sugiere evaluar la integración con Microsoft Azure utilizando máquinas virtuales spot, similares a las de AWS, para comparar costos y determinar cuál de los dos proveedores ofrece un mayor ahorro en la implementación.

Evaluar un cambio en el método de aprovisionamiento en instancias Linux, considerando herramientas como Puppet, Ansible o cloud-init en lugar de scripts de Bash, lo que permitiría agilizar el proceso. Asimismo, se sugiere analizar la viabilidad de implementar agentes con sistema operativo Windows con el objetivo de unificar la gestión de la infraestructura mediante Terraform y optimizar su administración.

Revisar periódicamente los reportes de vulnerabilidades generados por herramientas de análisis estático con el fin de prevenir brechas de seguridad en el código. Además, se sugiere modificar la configuración de los grupos de seguridad en AWS para restringir la conectividad

únicamente a las subredes dentro de la VPC, reduciendo la exposición innecesaria. Finalmente, se recomienda monitorear y actualizar las versiones de las herramientas instaladas en los agentes para evitar la ejecución de software con vulnerabilidades y garantizar un entorno más seguro.

REFERENCIAS

Abad, J. (2024, 28 octubre). Qué es un script - Definición, significado y para qué sirve. Arimetrics. Recuperado el 20 de noviembre del 2024, de https://www.arimetrics.com/glosario-digital/script

Amazon Web Services (s.f.) ¿Qué es Amazon EC2? - Amazon Elastic Compute Cloud. Recuperado del 12 de diciembre del 2024, de https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html

Amazon Web Services. (s. f.). ¿Qué es AWS? Amazon. Recuperado el 12 de diciembre de 2024, de https://aws.amazon.com/es/what-is-aws/?nc1=f_cc

Aranda, P. (2023, febrero 24). ¿Qué es un build? ITDO Blog. Recuperado el 12 de noviembre de 2024, de https://www.itdo.com/blog/que-es-un-build/

Cîrlan, A. (2024, julio 12) Mining for Cost Awareness in Cloud Computing: A Study of AWS CloudFormation and Developer Practices [tesis para bachiller, Universidad de Groninga]. Repositorio institucional de la Universidad de Groninga. https://fse.studenttheses.ub.rug.nl/id/eprint/33994

Gitlab (s.f.) What is CI/CD? Recuperado el 16 de noviembre del 2024, de https://about.gitlab.com/topics/ci-cd/#what-is-continuous-delivery-cd

Gitlab (s.f.) What is a CI/CD pipeline? Recuperado el 16 de noviembre del 2024, de https://about.gitlab.com/topics/ci-cd/cicd-pipeline/#ci-cd-pipelines-defined

Gracia, K. J. (2022, 30 septiembre). Propuesta de implementación de un proceso de despliegue automatizado, previo pruebas y análisis de código automatizado [tesis para título profesional, Pontificia Universidad Católica del Ecuador].

Repositorio institucional de la Pontificia Universidad Católica del Ecuador. https://repositorio.puce.edu.ec/handle/123456789/38149

Jara, T. (2021, julio). Implementación de una plataforma de virtualización usando Amazon Web Services, para soportar las aplicaciones de la empresa Programate S.A.C. [tesis para título profesional, Universidad Autónoma del Perú]. Repositorio de la Universidad Autónoma del Perú. https://repositorio.autonoma.edu.pe/bitstream/handle/20.500.13067/1511/Jara%20Condori%2C%20Terry%20Alexander.pdf?sequence=1&isAllowed=y

JetBrains (s.f.) Build Agent | TeamCity On-Premises. Recuperado el 20 de noviembre del 2024, de https://www.jetbrains.com/help/teamcity/build-agent.html#Build+Agent+Status

JetBrains (s.f.) Install and Start TeamCity Agents | TeamCity On-Premises. Recuperado el 20 de noviembre del 2024, de https://www.jetbrains.com/help/teamcity/install-and-start-teamcity-agents.html

Juconsa, M. (2022). Platform for deploying a highly available, secure and scalable web hosting architecture to the AWS cloud with Terraform [Tesis de grado, Universidad Politécnica de Cataluña]. Repositorio de la Universidad Politécnica de Cataluña. https://upcommons.upc.edu/handle/2117/371366

Kavas, E. (2023). Architecting AWS with Terraform: Design resilient and secure Cloud Infrastructures with Terraform on Amazon Web Services. https://www.oreilly.com/library/view/architecting-aws-with/9781803248561/

Krief, M. (2022). Learning DevOps Second Edition. https://www.packtpub.com/en-us/product/learning-devops-9781801818964

Kruger, J. (2024) Embracing DevOps Release Management. https://www.packtpub.com/en-us/product/embracing-devops-release-management-9781835461853

Landa, R. y Lujan, O. (2024, 17 de junio) Integración de servicios de AWS enfocado al uso de DevSecOps para mejorar el rendimiento y la seguridad en proyectos de Data & Analytics [Trabajo de suficiencia profesional, Universidad Peruana de Ciencias Aplicadas]. Repositorio académico UPC. https://repositorioacademico.upc.edu.pe/handle/10757/674691

Llauce, M. (2020, febrero) Implementación de una arquitectura de computación en la nube (cloud computing) diseñada para escalabilidad automática y alta disponibilidad basado en la plataforma de amazon web services (AWS) en la Universidad de Lambayeque [tesis para título profesional, Universidad Nacional Pedro Ruiz Gallo]. Repositorio institucional Universidad Nacional Pedro Ruiz Gallo. https://repositorio.unprg.edu.pe/handle/20.500.12893/10132

Lucas, J. (2024, 2 febrero). Qué es Azure: Introducción a la nube de Microsoft. OpenWebinars.net. Recuperado el 12 de noviembre del 2024, de https://openwebinars.net/blog/que-es-azure/#qué-es-azure

Microsoft Azure (s. f.) Qué es Azure: Servicios en la nube de Microsoft. Recuperado el 12 de diciembre del 2024, de https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-azure#Popular-questions

Moreno, J. (2022). CI/CD en Infraestructura como código (IaC). Caso real en Amazon Web Services (AWS) [Tesis de grado, Universidad Oberta de Cataluña]. Repositorio institucional de la Universidad Oberta de Cataluña.

https://openaccess.uoc.edu/bitstream/10609/145707/8/moreno_juanTFG0622memoria.pdf

Morris, K. (2021) Infrastructure as Code: Dynamic Systems for the Cloud Age. https://dl.ebooksworld.ir/books/Infrastructure.as.Code.2nd.Edition.Kief.Morris.OReilly.9781098114671.EBooksWorld.ir.pdf

Ofracio, E., Rey, A. (2024, 05 de junio) Diseño de infraestructura como servicio (IaaS) como solución para la entrega de servicios de ti para una empresa de seguros del mercado peruano [Tesis de grado, Universidad Peruana de Ciencias Aplicadas]. Repositorio académico UPC. https://repositorioacademico.upc.edu.pe/handle/10757/674804

Puolitaival, V. (2024). Enhancing development efficiency with DevSecOps template: a design science approach for IaC and CI/CD implementation for AWS. LUT University. https://lutpub.lut.fi/handle/10024/168498

Puppet (s.f.) What is Puppet? Recuperado el 16 de noviembre del 2024, de https://www.puppet.com/docs/puppet/7/what_is_puppet.html

Rojas, W. (2020) Proyecto de implementación de buenas prácticas de administración de recursos alojados en Amazon Web Services para controlar el consumo mensual [tesis para título profesional, Universidad San Ignacio de Loyola]. Repositorio institucional de Universidad San Ignacio de Loyola. https://hdl.handle.net/20.500.14005/10637

Schwaber, K., Sutherland, J. (2020). La Guía de Scrum. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-South-American.pdf

Shahin, M., Ali Babar, M. y L. Zhu (2017) Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices, IEEE Access, 5, 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629

Shore, J., Larsen, D. Klitgaard, G., Warden, S. (2021). The Art of Agile Development Second Edition. https://www.amazon.com/Art-Agile-Development-James-Shore/dp/1492080691

StarAgile. (2024, octubre 15). What is TeamCity and how does it work? StarAgile. Recuperado el 10 de diciembre de 2024, de https://staragile.com/blog/what-is-teamcity#What_is_TeamCity

ANEXOS

Anexo N°1. Carta de autorización del Comité Institucional de Ética en Investigación

ANEXO N° 2. Código de Terraform para crear una instancia EC2

```
data "aws_ami" "ubuntu" {
  most_recent = true

  filter {
    name   = "name"
    values = var.aws_ami_filter_name_values
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }

  owners = var.aws_ami_owners
}

resource "aws_instance" "ec2_cloud_profiles" {
  ami = data.aws_ami.ubuntu.id

  cpu_options {
    core_count       = try(var.cpu_options_core_count, null)
    threads_per_core = try(var.cpu_options_threads_per_core, null)
  }
  get_password_data = length(regexall("windows", var.name)) > 0 ? true : false
  key_name          = var.key_name
  root_block_device {
    delete_on_termination = true
    encrypted             = true
    iops                  = try(var.root_block_device_iops, null)
    kms_key_id            = var.root_block_device_kms_key_id
    volume_size           = var.root_block_device_volume_size
    volume_type           = var.root_block_device_volume_type
  }
  iam_instance_profile = var.iam_instance_profile

  private_dns_name_options {
    enable_resource_name_dns_a_record    = var.private_dns_name_options_enable_resource_name_dns_a_record
    enable_resource_name_dns_aaaa_record = var.private_dns_name_options_enable_resource_name_dns_aaaa_record
  }

  availability_zone                    = try(var.availability_zone, null)
  monitoring                           = try(var.monitoring, null)
  ebs_optimized                        = try(var.ebs_optimized, null)
```

```
  instance_initiated_shutdown_behavior = "stop"
  instance_type                        = var.instance_type
  subnet_id                            = var.subnet_id
  vpc_security_group_ids               = var.vpc_security_group_ids

  provisioner "remote-exec" {
    connection {
      type            = "ssh"
      user            = var.provisioner_user
      password        = aws_instance.ec2_cloud_profiles.get_password_data ?
rsadecrypt(aws_instance.ec2_cloud_profiles.password_data, var.private_key) : null
      host            = aws_instance.ec2_cloud_profiles.private_ip
      insecure        = true
      private_key     = length(regexall("windows", var.name)) > 0 ? null : var.private_key
      target_platform = length(regexall("windows", var.name)) > 0 ? "windows" : "unix"
      timeout         = "10m"
    }
    ## IF computer name contains windows
    inline = length(regexall("windows", var.name)) > 0 ? [
      "powershell.exe while (-not (Test-Path \"C:\\Users\\Administrator\\user-data-status.txt\")) { Start-Sleep -Seconds 6;};",
      "C:\\\"Program Files\"\\Git\\mingw64\\bin\\git config --global
url.\"https://oauth2:${var.github_token}@github.com\".insteadOf https://github.com"
      ] : [
      # else
      "until [ -f /var/lib/cloud/instance/boot-finished ]; do sleep 5; done;",
      "sudo -u devops sh -c 'git config --global url.\"https://oauth2:${var.github_token}@github.com\".insteadOf
https://github.com'"
    ]
  }
  user_data                   = filebase64(var.user_data_file)
  user_data_replace_on_change = true
  tags = merge({
    Name = var.name
    },
    var.tags
  )
  volume_tags = var.tags
}

resource "aws_ec2_instance_state" "stop_instance" {
  instance_id = aws_instance.ec2_cloud_profiles.id
  state       = "stopped"
}
```