Icesi-Health: Implementación de la Infraestructura Cloud, Pipeline de Integración Continua y Despliegue Continuo

Valentina Arias Parra
José Luis Restrepo Obando
Jhon Alejandro Saldarriaga López

Universidad Icesi
Facultad de Ingeniería
Ingeniería Telemática
Cali
2023

Icesi-Health: Implementación de la Infraestructura Cloud, Pipeline de Integración Continua y Despliegue Continuo

Valentina Arias Parra
José Luis Restrepo Obando
Jhon Alejandro Saldarriaga López

Automatización de la Infraestructura para Ingeniería Continua

Ing. Joan Sebastian Garcia Delgado
Ing. Juan Camilo Diaz

Universidad Icesi
Facultad de Ingeniería
Ingeniería Telemática
Cali
2023

Multinacional MyHealth

Es una empresa del sector salud con presencia en el país durante los últimos 30 años. MyHealth a lo largo de este tiempo, ha logrado consolidar y fidelizar una gran base de pacientes, alcanzando 150.000 aproximadamente. Este crecimiento, obligó a la compañía a evolucionar las soluciones de TI que soportan los procesos críticos de negocio. La empresa viene almacenando los registros de sus pacientes en un DB SQL, que no se ha podido actualizar, conectada a un BackEnd (Java), que ha venido operando de forma estable con una arquitectura monolítica. También sus reglas de negocio están escritas en Java.

Es por esto que se ha desarrollado una aplicación, denominada Icesi-Health que es una interfaz para un sistema de registros de pacientes. La aplicación se encuentra constituida por: un frontend en Node.JS Express, donde encontramos la interfaz de usuario que está programada utilizando librerías de código open source de JavaScript, CSS y HTML5 Canvas. Un backend donde encontramos la API servida por Node.js. Por último, una base de datos NoSQL denominada CouchDB. Por ahora, Icesi-Health ha evolucionado sus viejas reglas de negocio escritas en Java, y bajo una arquitectura monolítica, a una arquitectura microservicio desplegada en Node.Js.

La compañía también ha estado revisando el tema de la computación en la nube, vislumbrando la posibilidad de efectuar una posible migración de Icesi-Health. En la actualidad, todo su código se ejecuta en servidores on-premise. Sin embargo, algunos de sus arquitectos de software consideran importante, evolutivo y complementario acelerar el desarrollo de nuevas funciones de la aplicación y explorar las posibilidades que podría ofrecer el machine learning para aprovechar la base de información con la cuenta y que han acumulado por años.

Entrega #1: Implementación de la infraestructura Cloud, Pipeline de Integración Continua y Despliegue Continuo.

Proceso:

Herramientas usadas:

Terraform y Azure: Hashicorp Terraform es una herramienta de Infraestructura como Código (IaC) de código abierto para aprovisionar y gestionar la infraestructura en la nube. Esta herramienta codifica la infraestructura en archivos de configuración que describen el estado deseado de tu topología. Terraform permite la gestión de cualquier tipo de infraestructura, como nubes públicas, nubes privadas y servicios SaaS, utilizando proveedores de Terraform.

Ilustración 1. Terraform y Azure (IaC)

Nuestro código Terraform describe la infraestructura provisionada por Azure. Explicaremos los recursos y su funcionalidad de manera resumida:

I. Se establece el proveedor de Azure y se activan todas las características por defecto.
II. Se crea un grupo de recursos llamado "icesihealth-resource-group" en la ubicación "East US". Este grupo de recursos servirá como contenedor lógico para los recursos relacionados con la aplicación.
III. Se crea una red virtual llamada "icesihealth-vnet" con un espacio de direcciones IP de "10.0.0.0/16" en la ubicación del grupo de recursos.
IV. Se crea una subred llamada "icesihealt_subnet" con un rango de direcciones IP de "10.0.1.0/24" dentro de la red virtual. Esta subred estará asociada a la red virtual y al grupo de recursos.
V. Se crea una dirección IP pública llamada "couchIP" en la ubicación del grupo de recursos. La dirección IP se asigna de forma estática y utiliza el SKU "Standard".
VI. Se crea un espacio de trabajo de log analytics llamado "acctest-01" en la ubicación del grupo de recursos. Este espacio de trabajo se utilizará para recopilar y analizar registros de las aplicaciones.
VII. Se crea un entorno de aplicación de contenedor llamado "Containers-Environment" en la ubicación del grupo de recursos. Este entorno de aplicación estará asociado al espacio de trabajo de log analytics creado anteriormente.
VIII. Se crea una aplicación de contenedor llamada "icesihealth-front-app" que representa el front-end de la aplicación. Esta aplicación utiliza una plantilla de contenedor con la imagen "alejova2023/icesi-health-front:latest". Se configuran recursos como CPU y memoria para la aplicación.
IX. Se crea una aplicación de contenedor llamada "icesihealth-back-app" que representa el back-end de la aplicación. Al igual que la aplicación front-end, utiliza una plantilla de contenedor con la imagen "alejova2023/icesi-health-back:latest".
X. Se crea una aplicación de contenedor llamada "icesihealth-couch-db" que representa la base de datos CouchDB utilizada por la aplicación. Utiliza la

imagen "couchdb:3".

XI. Se crea una puerta de enlace de aplicación llamada "icesihealth-AppGateway" en la ubicación del grupo de recursos. Esta puerta de enlace sirve como punto de entrada para el tráfico y se configura con un SKU "Standard_v2" y una capacidad de 3 instancias.

XII. Se configura la puerta de enlace de aplicación para utilizar la subred y la dirección IP pública creadas anteriormente. También se especifica la configuración del puerto frontal y la configuración HTTP de destino para enrutar el tráfico a la aplicación front-end.

Además, contamos con dos pipelines:

El primero automatiza el proceso de construcción, prueba y despliegue de una aplicación en contenedor, y también realiza análisis estáticos del código y verificaciones de calidad utilizando SonarQube. Este pipeline cuenta con las siguientes etapas:

- La función qualityGateValidation(qg) se define para verificar si el estado de un "quality gate" (puerta de calidad) es exitoso. Si el estado no es "SUCCESS", se devuelve true, lo que indica que el gate no ha pasado. De lo contrario, se devuelve false.
- Se define el bloque pipeline con el agente como any, lo que significa que se puede ejecutar en cualquier agente disponible en Jenkins.
- Se define la sección tools para especificar la herramienta Node.js que se utilizará en el pipeline.
- Se establece la sección environment para definir variables de entorno que se utilizarán en el pipeline. Estas variables incluyen la ruta del proyecto, la URL del registro de Docker, las credenciales del registro, el grupo de recursos de Azure, la URL del servidor SonarQube, el puerto, el usuario y la contraseña de la base de datos, el directorio de CouchDB, la URL del repositorio y las variables de entorno de Docker y CouchDB.
- Se definen las etapas del pipeline dentro del bloque stages.
- La primera etapa es "Build" y utiliza el paso checkout para obtener el código fuente del repositorio de GitHub.
- La siguiente etapa es "Install dependencies" y utiliza el comando npm install para instalar las dependencias del proyecto.
- La etapa "Generate coverage report" ejecuta el comando npm run coverage para generar un informe de cobertura.
- La etapa "scan" utiliza el escáner de SonarQube para analizar el código fuente y los informes de pruebas. Se configuran diversas propiedades para el análisis, incluyendo la clave del proyecto, el nombre, la versión, la URL

del servidor SonarQube, las rutas de los archivos fuente y de prueba, y el archivo de informe de cobertura.

- Después de ejecutar el escáner de SonarQube, se utiliza la función waitForQualityGate() para esperar la finalización del análisis y la evaluación del "quality gate".
- Si el "quality gate" no pasa, se aborta el pipeline utilizando la función abortPipeline.
- La siguiente etapa es "Build docker-image" y se utiliza el comando docker.build para construir una imagen Docker utilizando el archivo Dockerfile y las variables de entorno definidas anteriormente.
- En la etapa "Deploy docker-image", se utiliza el comando dockerImage.push() para subir la imagen Docker al registro especificado en las credenciales del registro.
- Finalmente, en la etapa "Update Azure Container", se utiliza el comando az containerapp update para actualizar el contenedor de Azure con la nueva imagen del registro.

El segundo automatiza el proceso de construcción, prueba y despliegue de una aplicación de frontend en contenedor. También realiza análisis estático del código y verificaciones de calidad utilizando SonarQube. Este pipeline cuenta con las siguientes etapas:

- La función qualityGateValidation(qg) se define nuevamente para verificar si el estado de un "quality gate" (puerta de calidad) es exitoso. Si el estado no es "SUCCESS", se devuelve true, indicando que el gate no ha pasado. De lo contrario, se devuelve false.
- Se define el bloque pipeline con el agente como any, lo que significa que se puede ejecutar en cualquier agente disponible en Jenkins.
- Se establece la sección tools para especificar la herramienta Node.js que se utilizará en el pipeline.
- Se define la sección environment para definir variables de entorno que se utilizarán en el pipeline. Estas variables incluyen la URL del servidor SonarQube, el nombre del proyecto, el registro de Docker, las credenciales del registro, el grupo de recursos de Azure, la URL de la API, el puerto, la URL del repositorio y las variables de entorno de Docker.
- Se definen las etapas del pipeline dentro del bloque stages.
- La primera etapa es "Build" y utiliza el paso checkout para obtener el código fuente del repositorio de GitHub.
- La siguiente etapa es "Install dependencies" y utiliza el comando npm install para instalar las dependencias del proyecto.
- La etapa "Generate coverage report" ejecuta el comando npm run coverage

para generar un informe de cobertura.

- La etapa "scan" utiliza el escáner de SonarQube para analizar el código fuente y los informes de pruebas. Se configuran diversas propiedades para el análisis, incluyendo la clave del proyecto, el nombre, la versión, la URL del servidor SonarQube, las rutas de los archivos fuente y de prueba, y el archivo de informe de cobertura.
- Después de ejecutar el escáner de SonarQube, se utiliza la función waitForQualityGate() para esperar la finalización del análisis y la evaluación del "quality gate".
- Si el "quality gate" no pasa, se aborta el pipeline utilizando la función abortPipeline.
- La siguiente etapa es "Build docker-image" y se utiliza el comando docker.build para construir una imagen Docker utilizando el archivo Dockerfile y las variables de entorno definidas anteriormente.
- En la etapa "Deploy docker-image", se utiliza el comando dockerImage.push() para subir la imagen Docker al registro especificado en las credenciales del registro.
- Finalmente, en la etapa "Update Azure Container", se utiliza el comando az containerapp update para actualizar el contenedor de Azure con la nueva imagen del registro.

Retrospectiva del proyecto

¿Que salió bien?	¿Qué salió mal?	¿Cómo mejorar?
El despliegue de la infraestructura se logró correctamente y se puede evidenciar en el uso del frontend y backend de la aplicación, a excepción del application gateway, sin embargo se pudo solventar usando un frontdoor.	Al inicio se usó container groups pero tuvimos que desistir de la idea ya que no se podían actualizar. El Application Gateway dejó de funcionar al momento de actualizar los contenedores. Verificamos esto muy tarde y no se pudo actuar a tiempo.	Mejorar la búsqueda de recursos o alternativas. Entrenar o hacer diferentes ejercicios que nos permitan obtener experiencia y habilidad.

Tabla 1. Retrospectiva del proyecto