**Diseño de Pipeline: Demo Devops Java**

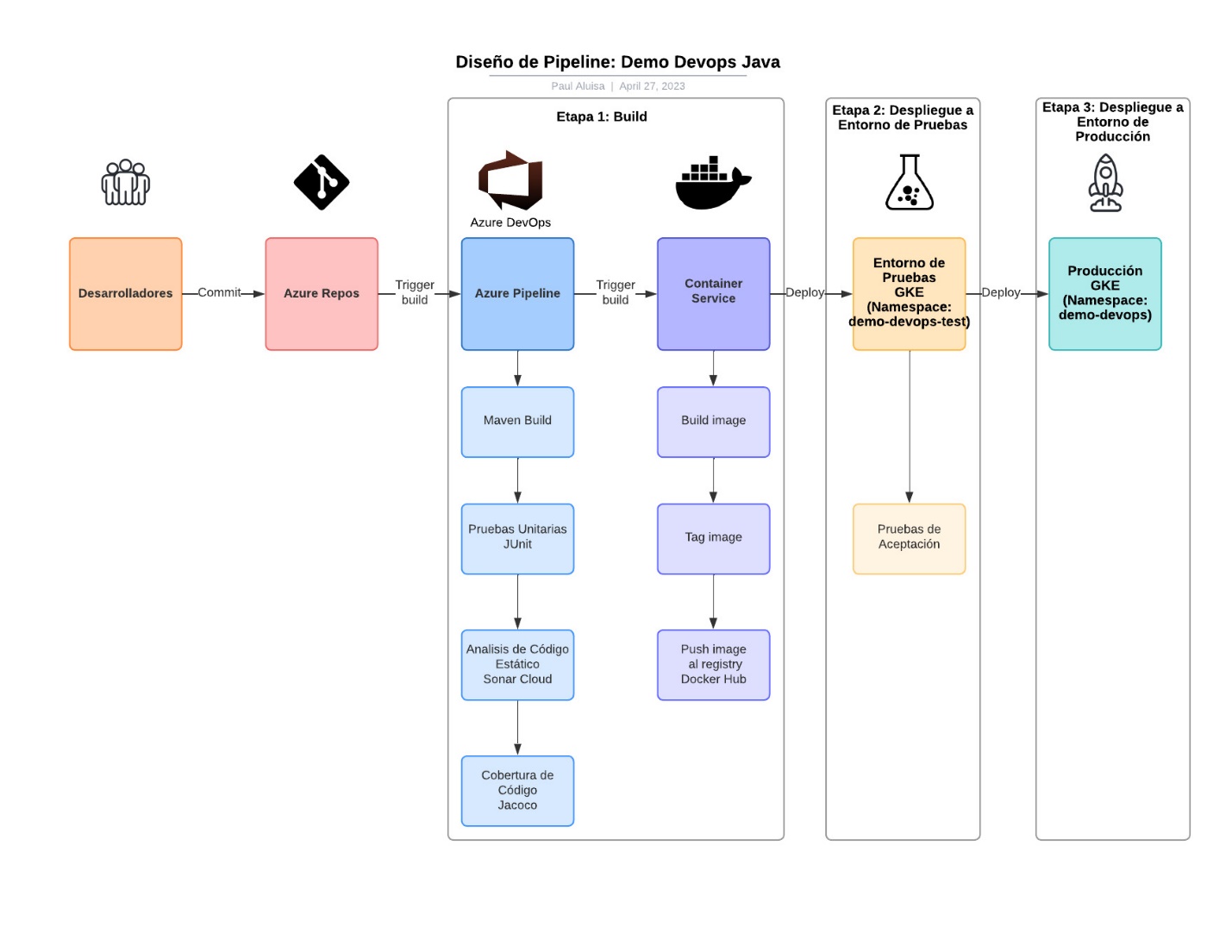
**Descripción general**

Este pipeline tiene como objetivo construir, probar y entregar el servicio Demo Devops Java. El servicio provee endpoints para consulta y creación de usuarios. Se desplegará en un cluster de Google Kubernetes Engine (GKE) a través de un pipeline de Azure DevOps.

El pipeline usará como apoyo un PC Windows con el agente on-premise de Azure DevOps para la ejecución de los task requeridos.

**Estructura del Pipeline**

El pipeline consta de tres etapas: Build (Pruebas Unitarias, Validación Estática de Código, Cobertura de Código, Construcción de Imagen Docker), Despliegue a Entorno de Pruebas (GKE) y Despliegue a Entorno de Producción (GKE).



**Etapa 1: Build**

La etapa de construcción se encarga de compilar y empaquetar la aplicación. Se utiliza el task Maven@3 de azure pipelines que ejecutará la compilación, pruebas unitarias y empaquetado en .jar utilizando el goal ‘test’ con Maven.

Para el análisis estático de código se realiza la configuración del task SonarCloudPrepare@1, en el mismo se indica el proyecto de Azure Devops que se analizará y se lo asocia al servicio de Sonar Cloud previamente configurado.

Para el análisis de cobertura de código se especifica en el tast Maven@3 que la herramienta para realizar será Jacoco.

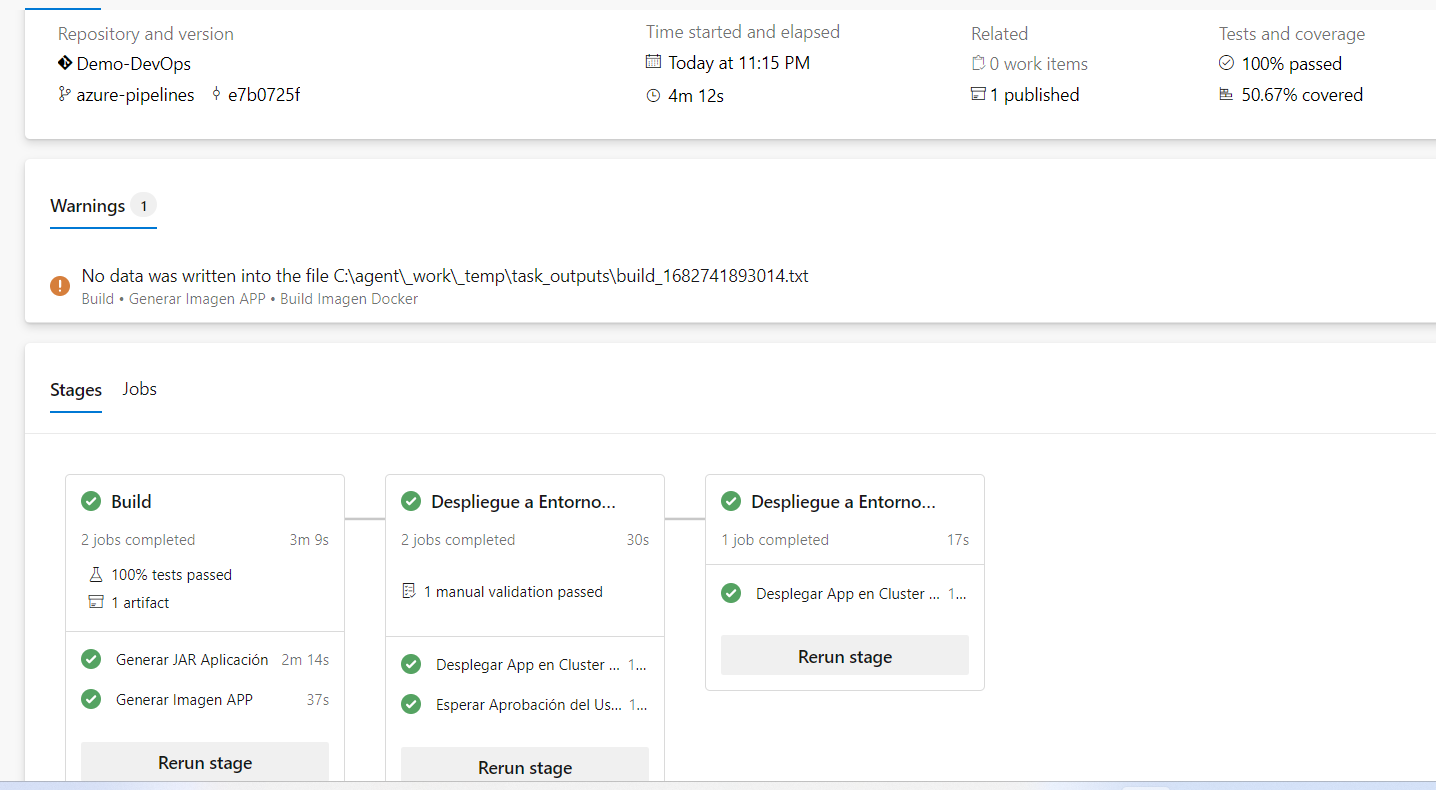
Con la ayuda del task Docker@2 se realiza el build de la imagen de Docker de acuerdo con el Dockerfile del repositorio. El push de la imagen se lo realiza a un repositorio de Docker Hub público previamente conectado, a través del pipeline. Dejando la imagen lista para el despliegue. Luego de la ejecución exitosa de la generación de la imagen se utiliza el task PowerShell@2 para modificar el tag de la imagen en el yaml que se utiliza para la creación del deployment en kubernetes.

**Etapa 2: Despliegue a Entorno de Pruebas**

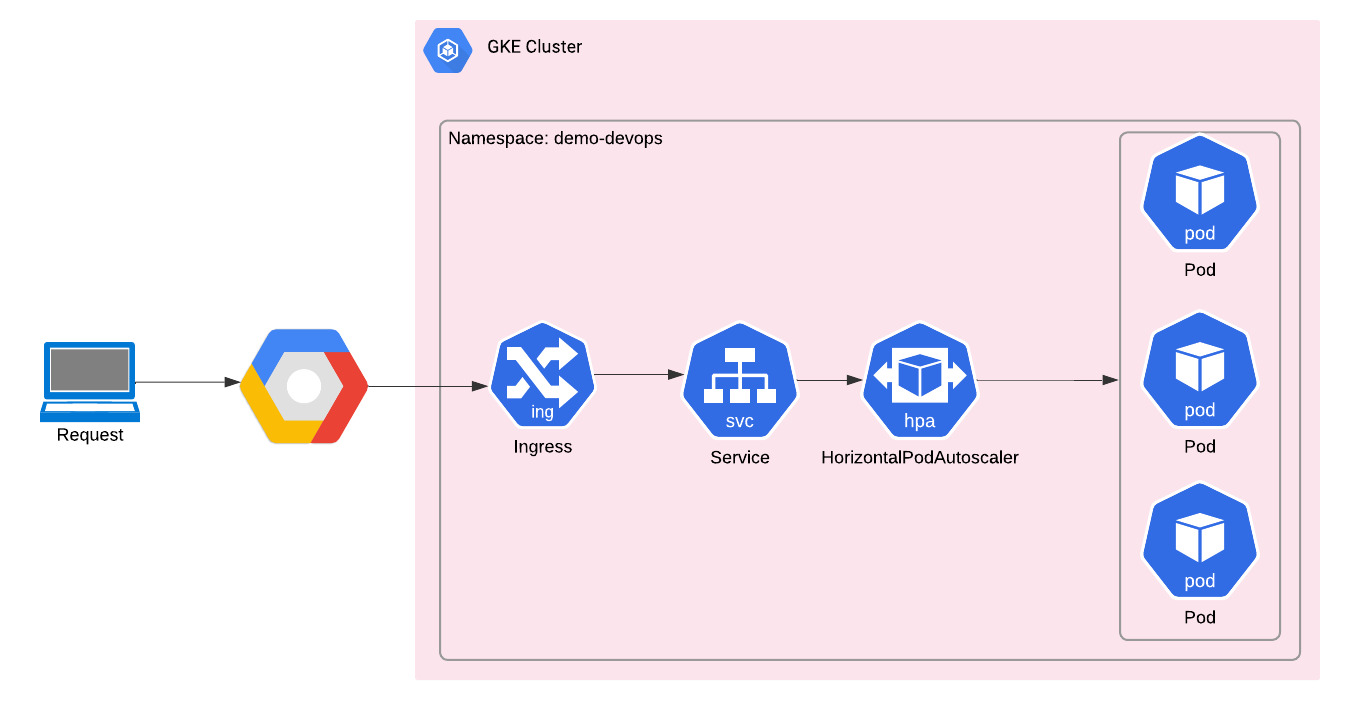
En la etapa de despliegue a entorno de pruebas, se implementa la aplicación en un cluster de kubernetes (GKE) con un namespace separado para el entorno de pruebas. En el despliegue se realiza la creación de los secrets, deployment, horizontal auto scaler, service, ingress y backend config necesarios para el levantamiento del servicio en el cluster GKE en el respectivo namespace para el entorno de pruebas.

**Etapa 3: Despliegue a Entorno de Producción**

En la etapa de despliegue a entorno de producción, se implementa la aplicación en un cluster de kubernetes (GKE) con un namespace separado para el entorno de producción. En el despliegue se realiza la creación de los secrets, deployment, horizontal auto scaler, service, ingress y backend config necesarios para el levantamiento del servicio en el cluster GKE en el respectivo namespace para el entorno de producción.



**Arquitectura de la aplicación**

****

La aplicación se compone de un clúster de Google Kubernetes Engine (GKE) en el que se despliegan los siguientes recursos para el funcionamiento de la aplicación:

**Secret:** Crea el secret en el que se guarda el usuario y contraseña de la conexión a base de datos del servicio.

**Deployment:** Crea el deployment para la creación de los pods, en el mismo se especifica la imagen generada en el pipeline y el respectivo tag, además de asociar el secret para tomar el usuario y contraseña de la base de datos y colocarlos como variables de entorno en los contenedores. Asocia el puerto por el que se expondrá la aplicación en los pods.

**HorizontalPodAutoscaler:** Se asocia al deployment para darle la característica de auto escalado horizontal según el consumo de cpu de los pods. Se establece el umbral para auto escalad y el número mínimo y máximo de replicas.

**Service:** Agrupa los pods identificados con el label app: demo-devops y expone una ip y puerto para acceder a los mismos.

**Ingress:** Se asocia al servicio y crea una interfaz de red para acceder al mismo de forma externa.

**BackendConfig:** Se asocia al servicio y se utiliza para configurar la revisión de la salud de un pod (healthcheck) y si se puede habilitar o no para que se acceda a través del Ingress. Se usa para configurar adecuadamente la verificación de estado.

**Dependencias del Pipeline**

El pipeline depende de las siguientes herramientas y servicios:

* Azure DevOps
* Recurso de cómputo on-Premise con las siguientes características:
  + Java JDK 17
  + Maven
  + Docker
  + Google Cloud SDK Shell
* Cluster Google Kubernetes Engine (GKE)
* Cuenta Sonar Cloud
* Cuenta Docker Hub