UNIVERSIDAD DE ANTIOQUIA

Implementación de CI/CD en Kubernetes usando Kaniko y Tekton

Autor(es)

John David Gonzalez Alzate

Universidad de Antioquia.

Facultad de ingeniería, Departamento de sistemas.

Medellín, Colombia

2021

Implementación de CI/CD en Kubernetes usando Kaniko y Tekton.

John David Gonzalez Alzate

Tesis o trabajo de investigación presentada(o) como requisito parcial para optar al título de:
Ingeniero de sistemas

Asesores:
Ph. D. Leonardo Augusto Pachón Contreras
Ing. Martín Elías Quintero

Universidad de Antioquia
Facultad de ingeniería, Departamento de sistemas
Medellín, Colombia
2021.

Tabla de Contenidos

Resumen

La automatización de los procesos de construcción y despliegue de aplicaciones facilita la actualización y el soporte de los sistemas que son desarrollados por los equipos de software. Agregar valor a los clientes les permite un crecimiento rápido en sus procesos y continua labor diaria por lo que detectar errores en los sistemas y darles solución rápidamente ayudan a su crecimiento empresarial. Es así que los equipos de desarrollo de software al hacer las correcciones reportadas por los clientes requieren que sean puestas en producción lo más pronto posible, es allí donde sistemas de integración y despliegue continuo (CI/CD) facilitan la labor de los desarrolladores debido a que las tareas de puesta en marcha de lo desarrollado no les resta el tiempo que se gana con las automatizaciones. En ese documento se presenta la ejecución de un pipeline de CI/CD a una aplicación que se encuentra disponible en un repositorio remoto, será desplegada haciendo uso de la herramienta kaniko para la construcción y carga de la imagen, y Tekton como recurso nativo de la nube especialmente de Kubernetes para la ejecución de las tareas e integración remota con el repositorio haciendo uso de los webhooks disponibles en github.

Introducción

Las empresas con base tecnológica implementan metodologías ágiles (Scrum, Extreme Programming, Kanban) para satisfacer las continuas necesidades de los clientes. Estas ayudan a mejorar los tiempos de entrega, flexibilizan las entregas de los productos y mejoran la eficiencia en el ciclo de vida del desarrollo de software [1]. Su implementación en la industria tecnológica permite realizar entregas constantes de los productos desarrollados para dar mayor valor a sus clientes. Para lograr esto se utilizan conceptos que ayudan a la automatización de los procesos de entrega y despliegue continuo. Estos términos fueron presentados inicialmente por Martin Fowler (integración

continua) en el año 2000 y posteriormente J.Humble and D.Farley extendieron el concepto de despliegue continuo [1].

El desarrollo de software se ha transformado gracias a la rápida implantación de nuevas herramientas que mejoran constantemente todo el ecosistema tecnológico. Es bastante amplio nombrar a todos los implicados en el desarrollo de un producto de software, pero en general, la implementación de estos se lleva a cabo por equipos de operaciones y de desarrollo. La ineficacia es evidente cuando no hay comunicación entre los dos equipos, sobre todo porque los objetivos de ambos están en disonancia. Por poner un ejemplo, los equipos de desarrollo tienen como objetivo lanzar nuevas características del producto en un corto período de tiempo, con la principal implicación de la posible inestabilidad del sistema, que es la mayor preocupación de las operaciones. Para cerrar esta brecha, se ha avanzado una cultura llamada DevOps y su implementación SRE(*site reliability engineer*), demostrando una alta tasa de éxito en la construcción de aplicaciones modernas usando metodologías ágiles.

En el desarrollo de este proyecto se utilizarán herramientas enfocadas a la nube nativa utilizando como entorno Kubernetes como Tekton y Kaniko para el funcionamiento conjunto de todas ellas y conseguir la automatización del despliegue y la integración continua para un Cluster enfocado a los asistentes cognitivos.

Objetivos

Implementar procesos de CI/CD en las fases de desarrollo y producción usando kubernetes.

Objetivos específicos

- Implementar Tekton para la manipulación de tareas en el pipeline de integración y despliegue continuo.
- Automatizar la construcción de imágenes de contenedores usando Kaniko.
- Integrar GitHub como repositorios de códigos fuente para crear el pipeline integración y despliegue continuo.

Marco Teórico

La construcción de software es una disciplina que ha evolucionado exponencialmente en los últimos años. Hoy existe una noción más clara sobre las mejores prácticas para llevar a cabo proyectos que involucran componentes digitales. Entre estas prácticas se encuentran las llamadas metodologías ágiles que permiten orquestar y sincronizar equipos de trabajo para lograr un objetivo. Por otro lado, están las estrategias técnicas que aseguran desde la tecnología el apoyo a todo el proceso de desarrollo en todas las fases de la construcción. La práctica de poner en producción un producto de software implica intrínsecamente que los artefactos vayan de un lado a otro en función de la solidez del ciclo de vida del software empresarial. Por lo tanto, cambiar uno de estos componentes implica una manipulación que tiene un alto índice de fallos debido al error humano que se puede inyectar en estos procesos tan robustos. Por ello, delegar esta responsabilidad para pasar del desarrollo a la producción está tomando cada vez más protagonismo en la automatización a través del mismo software.

En un entorno de desarrollo, más aún si se aplica una metodología ágil, es bastante común implementar nuevas funcionalidades a diario y que éstas

surtan efecto en el producto de forma casi inmediata dada la naturaleza evolutiva de los artefactos de software. Sin embargo, como se ha mencionado anteriormente, estos procesos realizados por equipos humanos pueden ser bastante complejos a medida que crece el volumen de cambios y características. Para apoyar y delegar estas tareas, se utiliza un software especializado para integrar los artefactos de software y ponerlos en producción, lo que se conoce comúnmente como Integración Continua(***CI***) y Despliegue Continuo(**CD**). Así, la integración continua es la técnica que permite realizar múltiples entregas preservando la estabilidad del software. Para cada integración, se realiza un proceso de compilación automatizado que ayuda a los equipos a detectar los problemas con antelación [3]. Por su parte, el despliegue continuo es un método que utiliza procesos de automatización para validar que los cambios realizados en los proyectos durante la fase de integración continua son adecuados y estables para su implementación en un entorno de producción. [4].

Para llevar a cabo procesos de integración y despliegue, existen varias herramientas que permiten el desarrollo ágil de pasos consecutivos llamados *Pipelines* que se despliegan asertivamente en la fase de desarrollo. En los últimos años y apoyado por el gran impulso de disciplinas como SRE [5], Kubernetes se ha convertido en la plataforma más estable y utilizada en la industria tecnológica. Así, las herramientas de CI/CD se han construido sobre el mismo entorno para soportar un enfoque totalmente basado en la nube. Esto resuelve uno de los grandes problemas del mundo del software a nivel de arquitectura: la escalabilidad y reproducibilidad de las aplicaciones, y lo hace utilizando tecnologías como los contenedores que aíslan los artefactos de software de forma lógica[6].

Una de las herramientas que mejor combina los procesos de integración y despliegue continuo en un entorno totalmente Cloud Native es Tekton. Tekton proporciona al desarrollador un ecosistema completo que va desde la

creación, las pruebas y el despliegue de secuencias de tareas hasta hacer que la integración CI/CD dentro de un entorno empresarial sea óptima, eficiente y relativamente fácil [7]. Este tipo de soluciones basadas íntegramente en Kubernetes orientan su desarrollo sobre contenedores, por tanto, la forma de llevar a cabo un proceso completo de vida del software consiste en construir una configuración que describa los elementos que ejecutan dichos contenedores, de esta forma, a través de estas instancias, la plataforma permite la gestión de los artefactos de software de forma minimalista.

Estas imágenes contienen toda la información necesaria para que la aplicación se ejecute de forma extremadamente eficiente. La creación de un archivo de imagen de contenedor se ha convertido en un arte que los desarrolladores afinan para que sus aplicaciones sean lo más portables posible, es decir, se puede construir adecuadamente de forma manual. Sin embargo, una buena práctica es delegar esta responsabilidad en una herramienta de construcción de imágenes como Kaniko, que con el apoyo de Tekton crea imágenes de contenedores sobre Kubernetes. Esta última herramienta utiliza un archivo de configuración que contiene todos los pasos secuenciales para construir las imágenes sin depender de soluciones comerciales que pueden no estar disponibles en Kubernetes. [8].

Metodología

La figura 1 muestra el diagrama de flujo que recoge las etapas desarrolladas en la práctica empresarial.

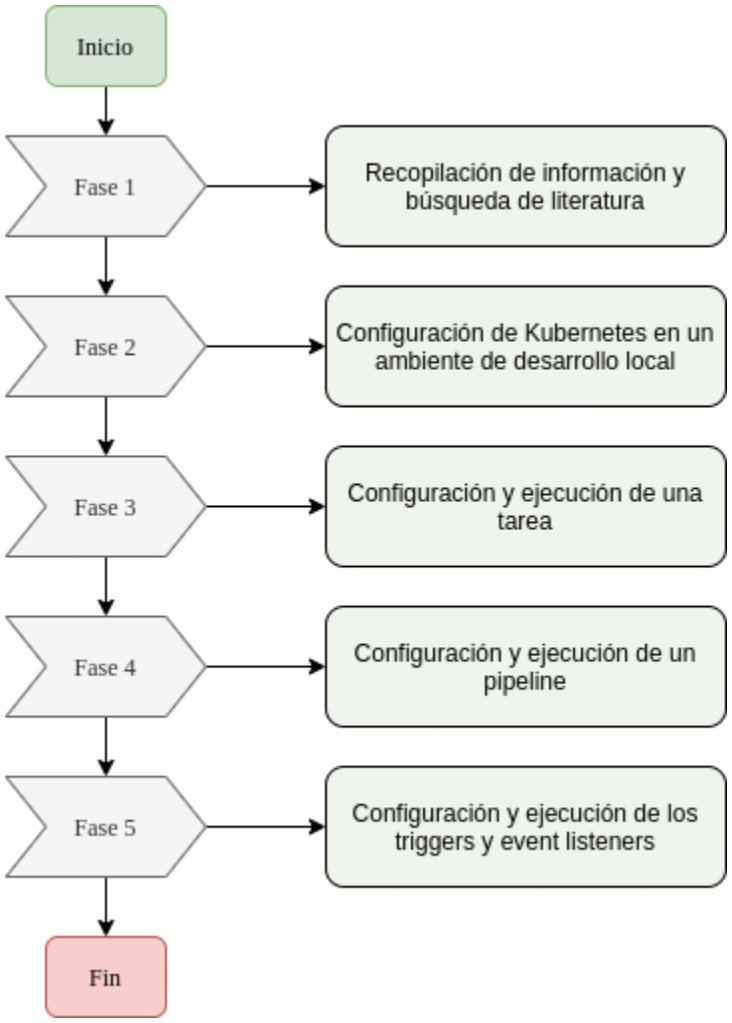


Figura 1. Diagrama de flujo de la metodología aplicada

Fase 1. Recopilación de información y búsqueda de bibliografía

La primera etapa de este proyecto se centró en incorporar diferentes fuentes de información para obtener la documentación suficiente para implementar

los procesos de CI/CD. Previamente se realizó un estudio exhaustivo sobre Kubernetes, por lo que tener un conocimiento inicial sobre las unidades fundamentales de esta plataforma como son los pods, los servicios, los configmaps y los namespaces, facilitó la creación de la fase de configuración adecuada para la implementación de una aplicación básica basada en contenedores y la gestión de la misma. A continuación, se comenzó a explorar la documentación oficial de Tekton[7], concretamente las secciones de los recursos de Kubernetes que facilitan la instalación y uso de los componentes para realizar un correcto proceso de CI/CD. Paralelamente a las lecturas relacionadas con la documentación de Tekton, investigamos Kaniko, un sistema destinado a crear imágenes Docker ayudado por la definición de un archivo de configuración secuencial llamado Dockerfile.

Fase 2. Configuración de Kubernetes en un ambiente de desarrollo local

Tras la etapa de consulta, recopilación de información y búsqueda bibliográfica, se continuó con la implementación de un entorno de desarrollo local que ayudó a reforzar los conocimientos adquiridos sobre las herramientas utilizadas, es decir, se puso en práctica lo aprendido durante la recopilación de información. Inicialmente se utilizó minikube[9], que permitió la construcción de un cluster de Kubernetes donde se instalaron las dependencias necesarias para la ejecución de Tekton: Interceptores, Trigger y un dashboard.

El desarrollo de la práctica continuó añadiendo en el clúster la respectiva configuración de permisos para determinados roles dentro de la plataforma que facilitaban la creación, edición y borrado de recursos a las dependencias instaladas. Para vincular el proceso con un control de versiones como GitHub, se creó un archivo encriptado dentro de Kubernetes que tiene la función de,

como se muestra en la Figura 2, ocultar información sensible dentro del clúster como el nombre de usuario y la contraseña para la autenticación en la aplicación. El uso de este mecanismo se debe a que se requieren credenciales dentro del proceso de Tekton para acceder a los repositorios privados de donde se obtiene la información del proyecto.

De forma similar a la creación del secreto para el control de versiones, se configuró uno adicional para la autenticación del registro Docker Hub para la subida y descarga de las imágenes que se desplegaba en el clúster.

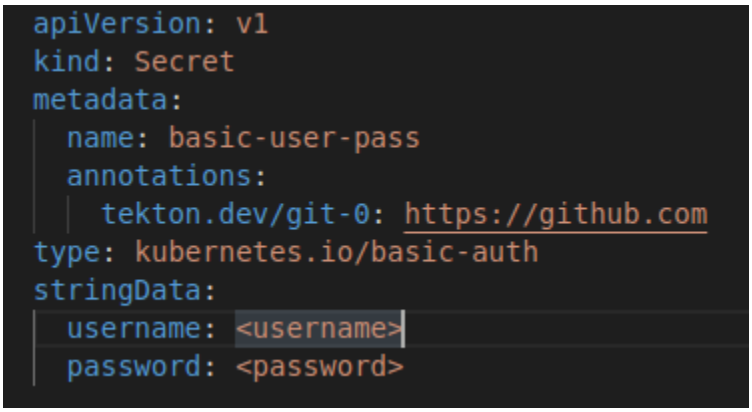


Figura 2. Definición de Secret para la autenticación básica en GitHub.

Fase 3. Configuración y ejecución de una tarea

Una vez configurado el clúster, se procedió a crear las tareas necesarias para la ejecución de las tareas gestionadas por el motor de CI/CD. Para esta fase, se procedió a crear los ficheros de configuración donde se encuentran las definiciones de los pasos de las tareas. En el archivo de definición mostrado en la Figura 3, se pueden ver los pasos definidos para construir la imagen y subirla a un registro de pruebas en DockerHub usando Kaniko. Antes de realizar esta

tarea, se descarga el repositorio sobre el que se va a trabajar y se define en los recursos del pipeline (Figura 4).

```yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: task-test
  namespace: test
spec:
  resources:
    inputs:
      - name: git-repo
        type: git
  steps:
    - name: build-image
      image: gcr.io/kaniko-project/executor:v1.3.0
      env:
        - name: "DOCKER_CONFIG"
          value: "/tekton/home/.docker"
      args:
        - "--dockerfile=/workspace/git-repo/dockerfile"
        - "--context=/workspace/git-repo"
        - "--destination=jhndagon11/test:latest"
    - name: delete-kubectl
      image: lachlanevenson/k8s-kubectl
      command: ["kubectl"]
      args:
        - "delete"
        - "-f"
        - "/workspace/git-repo/k8s"
    - name: run-kubectl
      image: lachlanevenson/k8s-kubectl
      command: ["kubectl"]
      args:
        - "apply"
        - "-f"
        - "/workspace/git-repo/k8s"

```

Figura 3. Definición del Task resource como archivo de configuración.

Tras la creación y carga de la imagen y esperando que no hubiera errores de construcción, se utilizó la imagen "lachlanevenson/k8s-kubectl" para ayudar a la ejecución de los comandos *kubectl* para desplegar, actualizar o eliminar un

registro dentro del clúster. Esto se consiguió gracias a la configuración creada previamente en la carpeta de Kubernetes del proyecto-repositorio que se descargó en uno de los pasos anteriores.

```yaml
apiVersion: tekton.dev/v1alpha1
kind: PipelineResource
metadata:
  name: repository
  namespace: test
spec:
  type: git
  params:
    - name: url
      value: <githuburlrepository>
    - name: revision
      value: main

```

Figura 4. Definición del pipeline resource como archivo yml.

Para la ejecución de la tarea previamente configurada, fue necesario definir un recurso de tipo Task Run (Figura 5), que permitió visualizar y probar que la tarea definida se estaba ejecutando satisfactoriamente y sin ningún tipo de error de codificación.

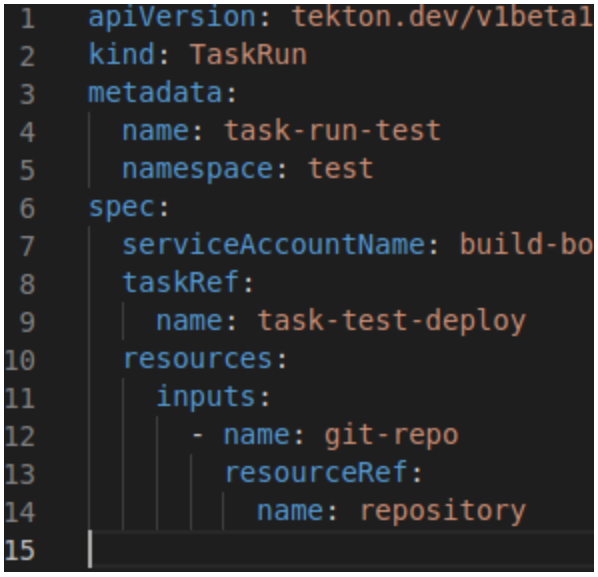
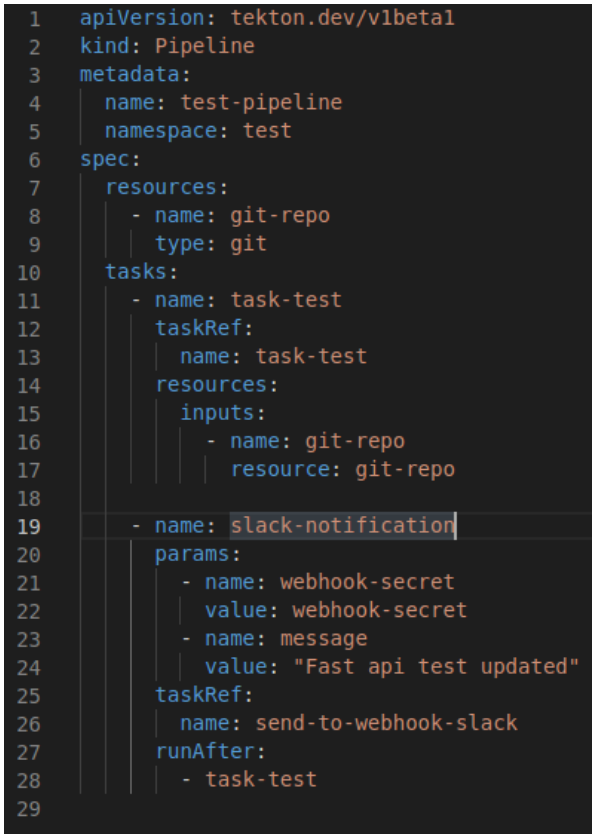


Figura 5. Definición del task run como archivo yml.

Fase 4. Configuración y ejecución de una secuencia de tareas (*pipeline*)

Once the task execution was configured and tested, we proceeded to create a task sequence (pipeline) that was responsible for executing a group of tasks, which are defined in the file shown in Figure 6. In the task sequence file we defined one specifically to build the application inside a container and deploy the image in Kubernetes, this task was built in phase 3. An additional task was added to help send notifications to a Slack channel after the initial task executed all its steps successfully. The configuration of notifications in the Slack channel was based on the task repositories implemented by the Tekton community and can be found on TektonHub [10].



```yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: test-pipeline
  namespace: test
spec:
  resources:
    - name: git-repo
      type: git
  tasks:
    - name: task-test
      taskRef:
        name: task-test
      resources:
        inputs:
          - name: git-repo
            resource: git-repo

    - name: slack-notification
      params:
        - name: webhook-secret
          value: webhook-secret
        - name: message
          value: "Fast api test updated"
      taskRef:
        name: send-to-webhook-slack
      runAfter:
        - task-test

```

Figura 6. Definición del pipeline como archivo yml.

De forma similar a la creación realizada en la fase 3 de la Ejecución de Tareas, en esta fase se ha definido un recurso de tipo Pipeline Run (Figura 7) que facilita la ejecución de las secuencias de tareas previamente definidas conteniendo el orden en el que deben ser aplicadas y además asegura que se ejecuten sin errores. Finalmente, se envía una notificación al canal Slack.

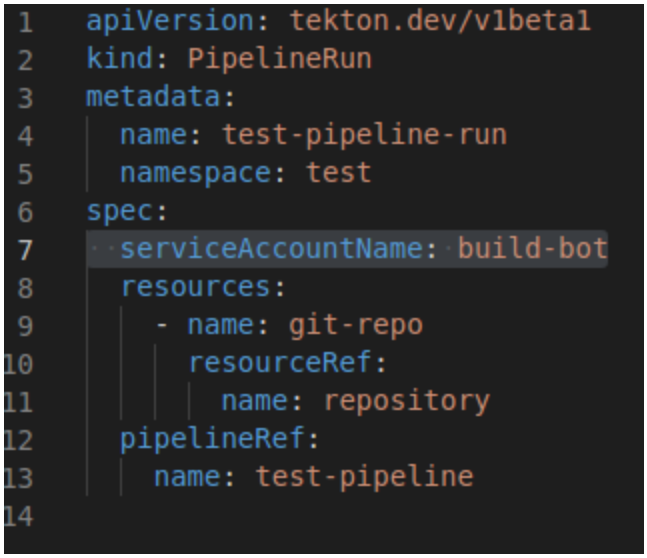


Figura 7. Definición del pipeline run como archivo yml.

Fase 5. Configuración y ejecución de los triggers y event listeners

En esta última fase de configuración se definieron los elementos necesarios para la creación de Triggers y Listener de eventos. Es necesario tener dentro del clúster las dependencias recomendadas por la documentación de Tekton para su uso. Además de la instalación, fue preciso estructurar los permisos dados en el clúster a las dependencias instaladas para que pudieran realizar las respectivas creaciones, actualizaciones o eliminaciones de los recursos propios de Tekton. Tras esta instalación, se configuraron los manifiestos necesarios para la ejecución de los Triggers.

El primero de los recursos configurados para la ejecución fue una plantilla de activación (Figura 8) que permitía la configuración para la ejecución automática del Pipeline Run definido en la fase 4. Este recurso requería una cuenta de servicio que contenía la respectiva composición de las credenciales

utilizadas para cargar la imagen generada desde el contenedor al registro definido en la secuencia de tareas.

```yaml
apiVersion: triggers.tekton.dev/v1alpha1
kind: TriggerTemplate
metadata:
  name: test-pipeline-template
  namespace: test
spec:
  resourcetemplates:
    - apiVersion: tekton.dev/v1beta1
      kind: PipelineRun
      metadata:
        generateName: test-pipeline-run-
      spec:
        serviceAccountName: build-bot
        pipelineRef:
          name: test-pipeline
        resources:
          - name: git-repo
            resourceRef:
              name: repository

```

Figura 8. Configuración del trigger template como archivo yml.

Continuamos con la estructuración del Trigger Binding (Figura 9), que es un recurso que especifica los campos que llegan en el cuerpo del mensaje enviado por el evento disparado y del que se extrae la información necesaria para completar el resto de pases correspondientes a la ejecución de las tareas correspondiente. Es decir, este recurso se encarga de obtener la información básica para enviarla como parámetros necesarios en la ejecución de las tareas. En este caso, solo se requería la dirección del repositorio desde el que se construía y ejecutaba la imagen que se desplegaba en el clúster sobre el que se trabajaba.

```yaml
apiVersion: triggers.tekton.dev/v1alpha1
kind: TriggerBinding
metadata:
  name: pipeline-binding
  namespace: test
spec:
  params:
    - name: gitrepositoryurl
      value: $(body.repository.url)

```

Figura 9. Configuración del trigger binding como archivo yml.

Para finalizar la fase de configuración y ejecución de los Triggers, se procedió a formular los últimos tres elementos necesarios. El primero de ellos fue crear un manifiesto que estuviera atento a los eventos que entraban en el sistema, este recurso se conoce como EventLister y su declaración se muestra en la Figura 10. Cada vez que se disparaba un evento, este mecanismo se encargaba de validar la autenticidad de la información entrante, es decir, que la notificación se recibiera desde el repositorio remoto desde el que se configuraba, también se confirmaba que el Token fuese consistente con la clave secreta creada para este recurso, y se aprobaba si el evento disparador era de tipo Push. Cuando se ejecutaba este recurso, automáticamente un servicio de tipo HTTP dentro del espacio de nombres correspondiente se disparaba.

Con el servicio ya en funcionamiento, se diseñó el segundo paso de esta etapa final, que consistió en configurar un recurso de ingreso(ver Figura 11) que expusiera a la red los instrumentos creados en el primer paso de esta fase. En el cluster utilizado como entorno de desarrollo, fue necesario añadir un complemento que facilitara la creación de los elementos de tipo Ingress. Como el entorno estaba en un ambiente virtual la dirección IP que se creó era local, por lo que fue necesario utilizar una herramienta para exponerla a internet, en este caso se usó Ngrok.

Para la última configuración fue necesario entrar en la interfaz gráfica del repositorio en GitHub desde donde se creó un Webhook que despacha los eventos del repositorio remoto cada vez que una persona con los permisos correspondientes hacía un Push para modificar o crear contenido dentro del proyecto.

```yaml
apiVersion: triggers.tekton.dev/v1alpha1
kind: EventListener
metadata:
  name: github-listener-interceptor
  namespace: test
spec:
  serviceAccountName: tekton-triggers-example-sa
  triggers:
    - name: github-listener
      interceptors:
        - github:
            secretRef:
              secretName: github-secret
              secretKey: secretToken
            eventTypes:
              - push
      bindings:
        - ref: pipeline-binding
      template:
        ref: test-pipeline-template

```

Figura 10. Configuración del Event Listener como archivo yml.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-resource
  namespace: test
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
    - http:
        paths:
          - path: /test-event-listener
            pathType: Prefix
            backend:
              service:
                name: el-github-listener-interceptor
                port:
                  number: 8080
```

Figura 11. Configuración del recurso tipo ingress.

Resultados y análisis

Una vez completado todo el proceso de instalación y configuración de las dependencias necesarias en el Cluster, y como se puede observar en la arquitectura de la Figura 12, se implementaron una serie de tareas que completaron todo el proceso de una aplicación, facilitando la recepción de notificaciones dentro del cluster de los eventos que se producen tras el envío y actualización al repositorio de los cambios generados por un desarrollador dentro de la aplicación. Este despliegue permitió reducir el tiempo del proceso de construcción y puesta en marcha de los proyectos, además de evitar errores de compilación durante la etapa de construcción de la imagen.

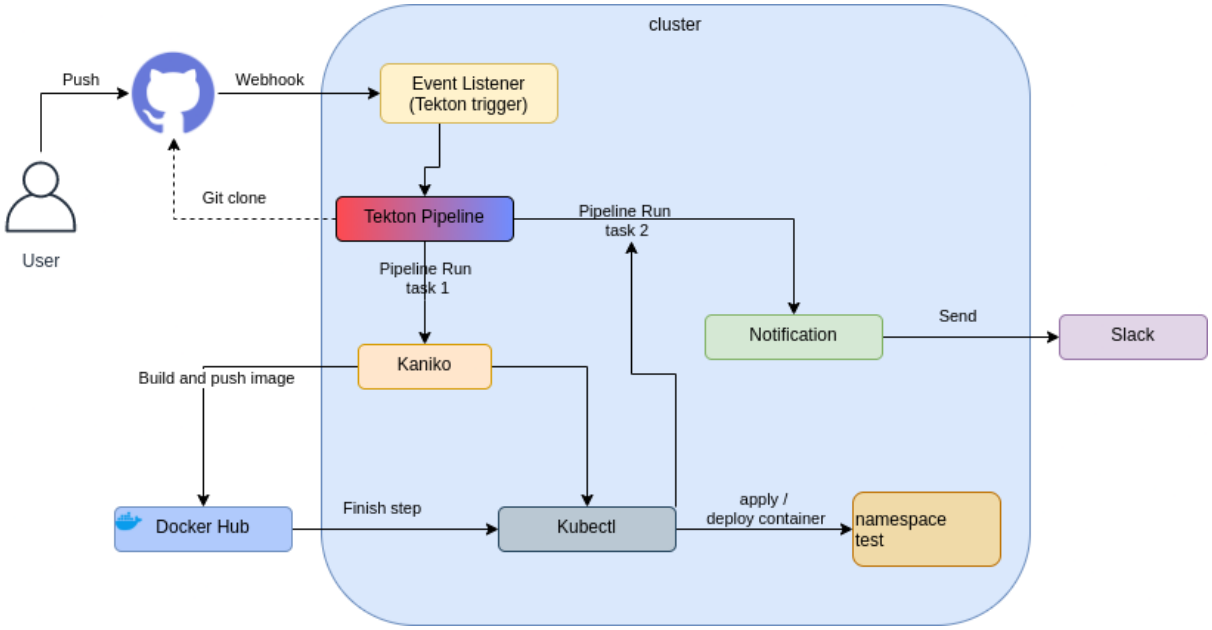


Figura 12. Arquitectura del proceso de integración y despliegue continuo.

Conclusiones

- El uso de herramientas para automatizar las tareas manuales de los desarrolladores facilita la mejora de los tiempos de respuesta a las nuevas funcionalidades y el soporte que requieren los clientes.

- La aplicación de herramientas de integración y despliegue continuo como Tekton ayuda a reducir los errores que podría cometer una persona.

- La integración de las notificaciones de los repositorios se hizo de forma limpia gracias a la facilidad de la interfaz gráfica de GitHub para enviar notificaciones sobre los eventos que ocurren dentro del código del proyecto.

Referencias bibliográficas

[1] Arachchi, S. A. I. B. S., and Indika Perera. "Continuous integration and continuous delivery pipeline automation for agile software project management." 2018 Moratuwa Engineering Research Conference (MERCon). IEEE, 2018.

[2] Mysari, S., & Bejgam, V. (2020, February). Continuous Integration and Continuous Deployment Pipeline Automation Using Jenkins Ansible. In 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE) (pp. 1-4). IEEE.

[3] Naik, V., & Bagmar, A. (2021). Continuous integration | ThoughtWorks. Thought Works. https://www.thoughtworks.com/continuous-integration

[4] Atlassian. (2021). What Is Continuous Deployment? https://www.atlassian.com/continuous-delivery/continuous-deployment

[5] Treynor, B. (2017). *Google - Site Reliability Engineering*. Google's Site Reliability Engineering (SRE) Book. https://sre.google/sre-book/introduction/

[6] Production-Grade Container Orchestration. (2021). Kubernetes. https://kubernetes.io/

[7] Tekton Documentation. (2021). Tekton. https://tekton.dev/docs/

[8] G. (2021). GoogleContainerTools/kaniko. GitHub. https://github.com/GoogleContainerTools/kaniko

[9] *Welcome!* (2021). Minikube. https://minikube.sigs.k8s.io/docs/

[10] *Tekton Hub*. (2021). Tekton Hub. https://hub.tekton.dev/