TIC

TALENTO TECH

Misión 3

ARQUITECTURA EN LA NUBE

Innovador

Tema 2: Automatización de despliegues en la nube

iTraining | UNIVERSIDAD DE ANTIOQUIA | Universidad de Caldas | ubicua Technology

Campista, llegó el momento de retar tus conocimientos y que los pongas a prueba a través de los diferentes recursos que encontrarás en este espacio como son: conceptos, ejemplos, herramientas, actividades prácticas y retos, los cuales te ayudarán a alcanzar los objetivos trazados en el nivel Innovador.

Introducción a DevOps

DevOps es una metodología que combina el desarrollo de software (Dev) y las operaciones de TI (Ops) con el objetivo de mejorar la colaboración y la eficiencia entre estos dos equipos. Al unificar el desarrollo y las operaciones, DevOps permite un ciclo de desarrollo más rápido y de mayor calidad, con menos errores y menos tiempo de inactividad.

Beneficios de DevOps

- **Reducción del tiempo de entrega:** DevOps permite desplegar software de manera más rápida y frecuente.

- **Mejora la colaboración:** DevOps rompe las barreras entre los equipos de desarrollo y operaciones.
- **Automatización:** Automatiza procesos repetitivos, lo que mejora la eficiencia y reduce los errores humanos.
- **Mayor calidad:** Con CI/CD, cada cambio en el código se prueba y despliega automáticamente, asegurando una mayor estabilidad.

Problemas antes de DevOps

- **Despliegues lentos y propensos a errores:** La falta de integración entre desarrollo y operaciones generaba problemas en los despliegues.
- **Dificultad para escalar:** La falta de automatización dificultaba la escalabilidad de las aplicaciones.
- **Falta de responsabilidad compartida:** Cada equipo (desarrollo y operaciones) tenía sus propios objetivos, lo que generaba conflictos y problemas de comunicación.

Ciclo de vida de DevOps

El ciclo de vida de DevOps abarca varias fases que aseguran la integración continua, entrega continua y monitoreo constante de las aplicaciones:

- **Desarrollo:** Los desarrolladores crean y prueban el código.
- **Integración Continua (CI):** Los cambios en el código se integran continuamente en un repositorio compartido y se prueban automáticamente.
- **Entrega Continua (CD):** El software se despliega automáticamente en diferentes entornos, desde pruebas hasta producción.
- **Monitoreo:** Se supervisa el rendimiento y el estado del software en producción para detectar y corregir problemas rápidamente.
- **Retroalimentación:** Se recopilan datos del monitoreo para mejorar continuamente el ciclo de desarrollo y despliegue.

Concepto de Pipeline en DevOps

Un pipeline en DevOps es una serie de pasos automatizados que el software sigue desde la codificación hasta la producción. Un pipeline típico incluye:

1. Construcción:

Compilación del código fuente para crear ejecutables.

2. Pruebas Unitarias:

Validación del código a nivel de función.

3. Pruebas de Interación:

Verificación del funcionamiento conjunto de componentes.

4. Despliegue:

Implementación del software en entornos de staging o producción.

5. Monitoreo:

Supervisión del software en producción para garantizar su funcionamiento correcto.

Herramientas Clave en DevOps

En DevOps, se utilizan diversas herramientas para automatizar y optimizar el ciclo de vida del desarrollo de software:

- **Jenkins:** Herramienta de automatización de CI/CD que permite gestionar y ejecutar pipelines.
- **GitLab CI:** Herramienta integrada con GitLab para la integración y entrega continua.
- **AWS CodePipeline:** Servicio de AWS para automatizar el flujo de trabajo de despliegue.
- **Terraform:** Herramienta de Infrastructure as Code (IaC) que permite gestionar y provisionar infraestructura de manera automatizada.
- **Prometheus y Grafana:** Herramientas para la monitorización y visualización del rendimiento de aplicaciones.

DevOps es fundamental para modernizar y mejorar el ciclo de vida del desarrollo de software. A través de la integración continua (CI), entrega continua (CD), y la monitorización constante, DevOps no solo acelera el proceso de desarrollo, sino que también mejora la calidad del software, reduce los errores y optimiza los recursos. Las herramientas como Jenkins, Terraform, y Prometheus son esenciales para implementar un enfoque DevOps eficaz. La adopción de DevOps y IaC no solo requiere un cambio en las herramientas, sino también un cambio cultural hacia la colaboración y la automatización.

Introducción a DevOps en Azure DevOps

DevOps es una combinación de prácticas, herramientas y filosofías culturales que mejora la capacidad de una organización para entregar aplicaciones y servicios a alta velocidad. Esta metodología permite a las empresas evolucionar y mejorar productos más rápidamente que las organizaciones que utilizan procesos tradicionales de desarrollo y gestión de infraestructura.

Objetivo de DevOps:

- Unificar desarrollo (Dev) y operaciones (Ops) para mejorar la colaboración y eficiencia.
- Fomentar la integración continua (CI) y la entrega continua (CD), asegurando que el software esté siempre en un estado de entrega.

Beneficios de DevOps:

- Aceleración del ciclo de desarrollo.
- Mejora en la colaboración entre equipos de desarrollo y operaciones.
- Mayor eficiencia en la entrega y despliegue de software.
- Reducción de fallos y tiempos de recuperación.

Introducción a Azure DevOps

¿Qué es Azure DevOps?

Azure DevOps es una plataforma de Microsoft que ofrece un conjunto completo de herramientas para gestionar todo el ciclo de vida del desarrollo de aplicaciones. Permite la planificación, desarrollo, pruebas, entrega, y monitoreo de aplicaciones a través de un solo entorno integrado.

Componentes principales de Azure DevOps:

- **Boards:** Herramientas para la gestión ágil de proyectos utilizando Kanban, Scrum y Sprints.
- **Repos:** Repositorios Git para la gestión del código fuente.
- **Pipelines:** Automatización de la integración continua (CI) y la entrega continua (CD) para pruebas y despliegues.
- **Test Plans:** Herramientas para la planificación y ejecución de pruebas automatizadas y manuales.
- **Artifacts:** Gestión y distribución de paquetes como NuGet, npm, entre otros.

Creación de Pipelines en Azure DevOps

¿Qué es un Pipeline?

Un pipeline en Azure DevOps es una serie de pasos automatizados que permiten construir, probar y desplegar una aplicación. Estos pipelines se configuran para ejecutar tareas en secuencia, como la compilación del código, la ejecución de pruebas y la implementación en diversos entornos.

Secciones de un Pipeline:

Un pipeline en Azure DevOps se compone de varias secciones clave:

- **Stages (Etapas):** Divisiones lógicas del pipeline que representan una fase del ciclo de vida de la aplicación, como build, test, y deploy.
- **Jobs:** Grupos de pasos que se ejecutan en conjunto dentro de una etapa. Un job puede contener múltiples pasos que se ejecutan en secuencia.

- **Steps:** Instrucciones individuales dentro de un job. Estos pueden ser scripts, comandos, o tareas predefinidas.
- **Triggers:** Definen cuándo y cómo se ejecuta el pipeline, como por ejemplo, en cada push al repositorio o de manera programada.
- **Variables:** Parametrizan el pipeline, permitiendo reutilizar configuraciones y hacer ajustes dinámicos.

Pools de Agentes:

- **¿Qué son los Agentes?** Los agentes son máquinas que ejecutan los trabajos definidos en un pipeline. Pueden ser máquinas locales, máquinas virtuales, o contenedores en la nube.
- **Pools de Agentes:** Un pool de agentes es un grupo de agentes disponibles para ejecutar trabajos en los pipelines. Azure DevOps ofrece pools de agentes hospedados (proporcionados por Microsoft) y pools de agentes auto- hospedados (gestionados por la propia organización).

Service Connections:

- **¿Qué son las Service Connections?** Las service connections permiten que Azure Pipelines se conecte de manera segura a otros servicios externos, como AWS, Kubernetes, Azure, entre otros. Estas conexiones se utilizan para autenticar y acceder a la infraestructura y servicios durante la ejecución del pipeline.
- **Configuración de Service Connections:** Para configurar una service connection, se debe proporcionar la información de autenticación y permisos necesarios para que Azure DevOps pueda interactuar con el servicio externo. Por ejemplo, para conectarse a AWS, se necesita configurar una conexión con las credenciales de AWS (Access Key y Secret Key) y definir los roles y permisos.

Pasos Detallados para Crear un Pipeline:

1. Configurar el Repositorio:

- Vincula el repositorio de código fuente (por ejemplo, GitHub, Azure Repos) a Azure DevOps.

2. Definir Tareas en los Jobs:

- **Build (Construcción):** Define tareas para compilar el código y ejecutar pruebas unitarias.
- **Test (Pruebas):** Incluye tareas para pruebas de integración y otras validaciones.

- **Deploy (Despliegue):** Configura tareas para desplegar la aplicación en los entornos definidos.

3. Configurar Entornos:

- **Entorno de Desarrollo:** Implementa cambios y pruebas iniciales.
- **Entorno de Pruebas:** Valida las funcionalidades en un entorno que simula producción.
- **Entorno de Producción:** Despliega la aplicación final para los usuarios.

4. Utilizar Variables y Condiciones:

- Define variables globales que pueden ser reutilizadas en múltiples etapas.
- Configura condiciones para ejecutar trabajos solo si las etapas anteriores han tenido éxito.

Ejecutar el Pipeline:

- Corre el pipeline y revisa los resultados en tiempo real en cada sección.
- Ajusta las configuraciones basadas en los resultados obtenidos.

Ejemplo de Configuración de un Pipeline:

```
trigger:
- main

pool:
  vmImage: ubuntu-latest'

stages:
- stage: Build
  jobs:
  - job: BuildJob
    steps:
    - script: echo Building the project...
    - script: dotnet build MyApp.sln


- stage: Deploy
  jobs:
  - deployment: DeployJob
    environment: &#39;Production&#39;
    strategy:
      runOnce:
        deploy:
          steps:
          - script: echo Deploying to Production...
          - script: dotnet publish -o /publish
          - script: az webapp deploy --resource-group MyResourceGroup --name
MyApp --src-path /publish
```

Este es un ejemplo básico que muestra cómo estructurar un pipeline con etapas de construcción y despliegue.

Integración y Despliegue de Aplicaciones con Azure DevOps

Azure DevOps facilita la integración continua (CI) y la entrega continua (CD) mediante la automatización de todo el proceso de construcción, prueba y despliegue de aplicaciones. Esta automatización permite:

- Pruebas automatizadas: Asegurar la calidad del código mediante pruebas continuas durante el proceso de desarrollo.
- Despliegue en múltiples entornos: Implementar aplicaciones en diferentes entornos sin intervención manual.

- Notificaciones: Recibir alertas automáticas sobre el estado del pipeline, fallos, y despliegues exitosos.

Conexión de Azure Pipelines con AWS y Kubernetes en EKS:

1. Crear una Service Connection a AWS:

- Navega a "Project Settings" en Azure DevOps.
- Selecciona "Service Connections" y añade una nueva conexión tipo AWS.
- Introduce las credenciales de AWS (Access Key y Secret Key) y define los permisos necesarios.
- Guarda y valida la conexión.

2. Configurar el Acceso a Kubernetes en EKS:

- En AWS, asegura que el cluster de Kubernetes (EKS) esté correctamente configurado y accesible.
- Crea un archivo kubeconfig en el pipeline de Azure para interactuar con EKS.
- Usa kubectl en las tareas del pipeline para desplegar y gestionar aplicaciones en Kubernetes.

3. Ejemplo de Pipeline para Desplegar en EKS:

```
trigger:

- main

pool:

vmImage: 'ubuntu-latest'

stages:

- stage: Build

jobs:

- job: BuildJob

steps:

- script: echo Building the project...
```

- script: dotnet build MyApp.sln

- stage: DeployToEKS

jobs:

- deployment: DeployJob

environment: 'Production'

strategy:

runOnce:

deploy:

steps:

- task: UseKubernetes@1

inputs:

connectionType: 'Azure Resource Manager'

azureSubscription: 'Azure_Subscription_ID'

resourceGroupName: 'Resource_Group_Name'

kubernetesCluster: 'EKS_Cluster_Name'

- script: |

kubectl apply -f k8s/deployment.yaml

kubectl apply -f k8s/service.yaml

displayName: 'Deploy to Kubernetes'

Este pipeline muestra cómo conectar Azure DevOps con AWS y desplegar aplicaciones en un clúster de Kubernetes (EKS) utilizando las credenciales y configuraciones definidas.

Comparativa con AWS CodeBuild y CodePipeline

Azure DevOps vs AWS CodeBuild/CodePipeline:

Azure Devops:

- Mayor integración con el ecosistema de Microsoft.
- Herramientas integradas que cubren todo el ciclo de vida del desarrollo.
- Ideal para desarrolladores que utilizan herramientas y tecnologías de Microsoft.

AWS CodeBuild:

- Proporciona mayor personalización en la compilación de aplicaciones.
- Integración más estrecha con otros servicios de AWS.

AWS CodePipeline:

- Enfocado en la integración continua y entrega continua dentro del ecosistema AWS.
- Flexibilidad en la integración con herramientas de terceros.

Introducción a DevOps en Azure DevOps

Puntos clave:

- DevOps es esencial para mejorar la colaboración entre desarrollo y operaciones.
- Azure DevOps ofrece una plataforma completa para gestionar el ciclo de vida del software.
- Los pipelines en Azure DevOps automatizan CI/CD, mejorando la eficiencia del desarrollo.
- La configuración detallada de pipelines en Azure DevOps, incluyendo pools de agentes y service connections, es fundamental para automatizar y escalar despliegues en infraestructuras multi-nube.
- Azure Pipelines permite la integración fluida con AWS y Kubernetes, habilitando despliegues híbridos y la gestión de aplicaciones en entornos distribuidos.

Conversatorio de estrategia de despliegues en la nube

1. Introducción al Proyecto Final

En esta lección, vamos a discutir las mejores estrategias de despliegue para el proyecto final. El proyecto consta de los siguientes componentes:

- **Frontend**: Aplicación web Angular.
- **Backend**: Dos microservicios que proporcionan la lógica de negocio.

El objetivo es identificar la mejor estrategia para desplegar cada uno de estos componentes, considerando opciones como:

- Infraestructura en EC2.
- Despliegue en contenedores (Kubernetes, EKS).
- Soluciones Serverless.

También discutiremos cómo aplicar Infraestructura como Código (IaC) y DevOps para automatizar y optimizar los procesos de despliegue.

2. Preguntas Iniciales

Antes de sumergirnos en las estrategias, reflexiona sobre las siguientes preguntas:

- **¿Qué estrategias de despliegue conoces?**
  - Pueden incluir Blue/Green, Canary, A/B Testing, entre otras.
- **¿Qué diferencias hay entre EC2, contenedores y serverless?**
  - EC2 ofrece control total sobre la infraestructura, contenedores proporcionan portabilidad y eficiencia, mientras que serverless escala automáticamente pero con menos control.

- **¿Cómo aplicarías IaC y DevOps en este proyecto?**
  - Considera el uso de herramientas como CloudFormation o Terraform para IaC, y Jenkins o Azure DevOps para automatización.

3. Estrategia de despliegue:

Vamos a analizar las mejores opciones de despliegue para cada componente del proyecto:

3.1 Despliegue del Frontend Angular

Opciones a considerar:

- **S3 + CloudFront:** Ideal para aplicaciones estáticas, proporcionando alta disponibilidad y entrega rápida de contenido.
- **EC2:** Proporciona control total, pero requiere mayor gestión.
- **Contenedores:** Facilita la portabilidad, pero puede ser más complejo de gestionar.
- **Serverless:** Ofrece escalabilidad automática con menor gestión, pero con limitaciones en personalización.

3.2 Despliegue de los Microservicios Backend

Opciones a considerar:

- **Contenedores en EKS**: Excelente para microservicios debido a la portabilidad y facilidad de escalado.
- **EC2:** Ofrece flexibilidad y control, pero requiere mayor gestión y escalado manual.
- **Serverless:** Ideal para funciones pequeñas que necesitan escalar automáticamente, pero con menos control y flexibilidad.

4. Comparativa de Tecnologías

Aquí te ofrecemos una comparación de las tecnologías disponibles para el despliegue:

- **EC2:**
  - **Ventajas:** Gran control sobre la infraestructura, flexibilidad en la configuración.
  - **Desventajas:** Requiere gestión manual, escalabilidad no automática.
- **Contenedores (Kubernetes, EKS):**
  - **Ventajas:** Alta portabilidad, optimización de recursos, fácil escalado horizontal.
  - **Desventajas:** Puede ser complejo de gestionar, requiere conocimiento especializado.

- **Serverless:**
  - **Ventajas:** Escalabilidad automática, menor sobrecarga en gestión.
  - **Desventajas:** Menor control y flexibilidad, limitaciones en algunas funcionalidades avanzadas.

5. Análisis de Escenarios Prácticos

Analizaremos diferentes escenarios donde deberás decidir la mejor estrategia de despliegue:

- Escenario 1: Alta disponibilidad para el frontend Angular.
- Escenario 2: Escalabilidad dinámica para los microservicios backend.
- Escenario 3: Despliegue seguro y eficiente utilizando IaC y DevOps.

Conclusiones y Recomendaciones:

- Identificar y justificar la mejor estrategia de despliegue para cada componente de tu proyecto final.
- Entender las ventajas y desventajas de EC2, contenedores y soluciones serverless.
- Aplicar conceptos de IaC y DevOps para automatizar y optimizar el despliegue de tus aplicaciones en la nube.

Aprendizaje práctico

Estos ejercicios permitirán a los campistas aplicar los conceptos aprendidos en situaciones prácticas y específicas.

1. Despliegue Azure DevOps

Despliegue de Aplicaciones con Azure DevOps

En esta práctica, configurarás y gestionarás pipelines de despliegue en Azure DevOps. Aprenderás a configurar un pipeline completo para una aplicación web, integrando Azure DevOps con AWS para el despliegue en la nube. Además, te enfocarás en la ejecución de pruebas unitarias para asegurar la calidad del código antes del despliegue.

Campista: recuerda revisar el PDF "Despliegue Azure DevOps"

2. Despliegue K8s DevOps

Despliegue de Aplicaciones Contenerizadas desde Pipelines

El objetivo de esta práctica es que implementes un pipeline de CI/CD en Azure DevOps para compilar una imagen Docker de una aplicación Node.js y desplegarla en un clúster de Kubernetes gestionado en AWS EKS. Utilizarás un pipeline de build para la compilación de la imagen y un pipeline de release para el despliegue.

*Campista: recuerda revisar el PDF "**Despliegue K8s DevOps".***

3. Despliegue Serverless DevOps

Despliegue de Aplicaciones Contenerizadas desde Pipelines

El objetivo de esta práctica es que implementes un pipeline de CI/CD en Azure DevOps para compilar una imagen Docker de una aplicación Node.js y desplegarla en un clúster de Kubernetes gestionado en AWS EKS. Utilizarás un pipeline de build para la compilación de la imagen y un pipeline de release para el despliegue.

*Campista: recuerda revisar el PDF "**Despliegue Serverless DevOps**"*