*INGENIERIA DEVOPS*

A computer with icons flying out of it

AI-generated content may be incorrect.

Estudiantes:Javiera Aguirre y Elias Delgado

Asignatura: Ing. Devops

Profesor:Daniel Riquelme

Sesión:001V

Creación y Configuración de la Instancia EC2

Esta primera parte consiste en el despliegue de nuestro servidor virtual.

Lanzamiento de la Instancia: Se creó una instancia de Amazon EC2 llamada ***"Práctica de monitoreo".***

1. AMI (Imagen de Máquina de Amazon): Se seleccionó una Amazon Linux 2023 AMI.
2. Tipo de Instancia: Es un servidor pequeño, t2.micro, elegido probablemente por ser elegible para el nivel gratuito de AWS (Free tier eligible).
3. Almacenamiento: Se configuró un volumen raíz de 8 GiB de tipo gp3.

Perfil de Instancia **IAM** (esta parte es necesaria para seguir avanzando): En los detalles avanzados, se prefirió un perfil de instancia IAM llamado ***LabInstanceProfile***. Esto es fundamental, ya que este perfil es el que otorga los permisos necesarios a la instancia para que pueda, por ejemplo, enviar datos y comunicarse con otros servicios de ***AWS,*** *como* ***CloudWatch y SSM***, que usaremos más adelante.

Conectividad y Seguridad

1. Se le asignó un par de claves (Key pair) llamado **22** para el acceso seguro (SSH).
2. Se creó un nuevo grupo de seguridad llamado ***launch-wizard-2***.
3. Se habilitó el acceso SSH desde cualquier dirección IP (0.0.0.0/0).
4. Se habilitó el auto-asignación de una IP pública ***(54.163.193.98).***

Debemos fijarnos si el estado de la instancia fue lanzada y está en estado **Running**, si en el caso no diga que está corriendo la instancia hay que repetir el paso nuevamente, hasta que quede andando.

A screenshot of a computer

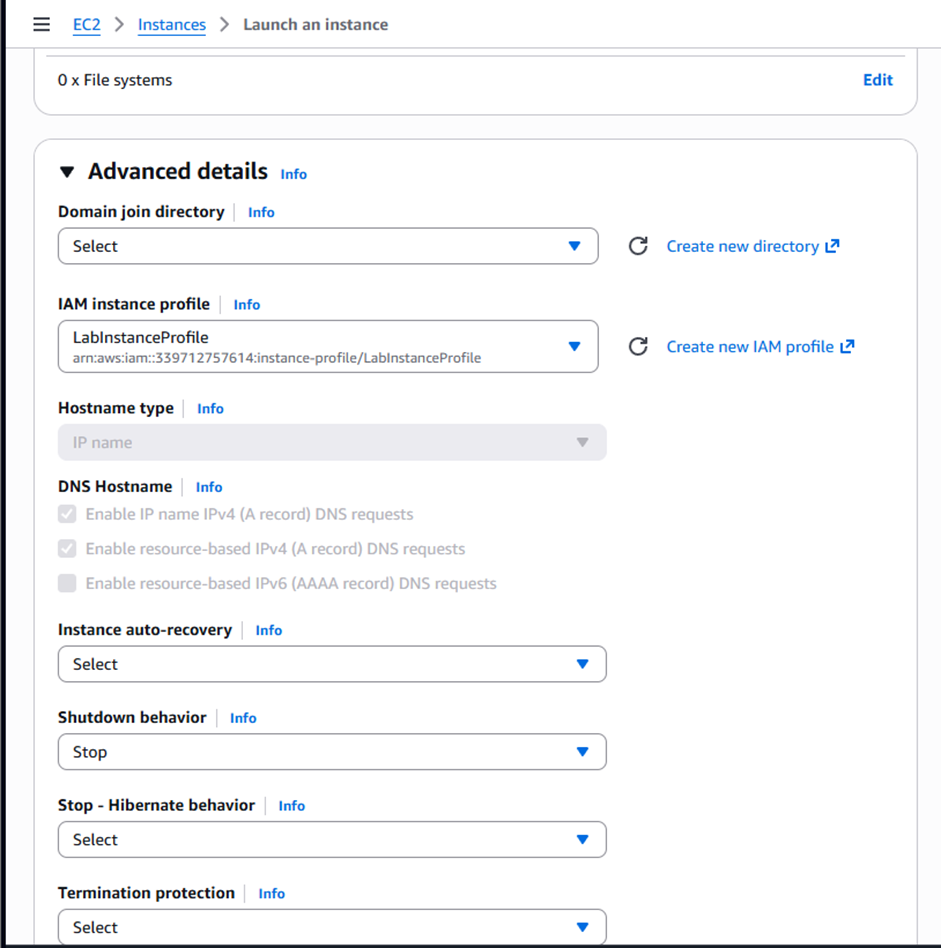
AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer

AI-generated content may be incorrect.

Configuración del Agente CloudWatch

Para monitorear el uso interno de la memoria y el sistema operativo, se necesita un agente especial.

Validación del Agente SSM: Primero, se verificó que el agente SSM (Systems Manager) estuviera Online para poder gestionar la instancia de forma remota.

Validación del Agente CloudWatch: El sistema intentó validar el agente CloudWatch. Inicialmente, el estado fue "Did not respond", esto pasa porque la cuenta estudiantil no tiene todos los permisos pero igual se puede llevar a cabo la instalación exitosa.

Selección de Métricas

Se configuró el agente para que recoja métricas detalladas:

1. ***CPU: Se seleccionó Usage active y Usage system.***
2. ***Memoria: Se seleccionó Used percent, Total, y Active.***

Estado Final

La configuración del agente CloudWatch se envió exitosamente a la instancia.

Configuración del Servicio de Notificación (Amazon SNS)

Para recibir alertas por correo, se configuró el sistema de notificación.

Creación del Tópico SNS: Se creó un Tópico Standard de Amazon Simple Notification Service (SNS) llamado ***NotificacionPorCorreo.***

Creación y Confirmación de Suscripción

1. Se creó una suscripción para el tópico, usando el protocolo Email y la dirección ***javi.aguirre@duocuc.cl*** como ***Endpoint*.**
2. El sistema envió un correo de confirmación (que terminó en la carpeta de ***Spam***).
3. Se confirmó la suscripción mediante el enlace del correo, y el estado de la suscripción cambió a ***Confirmed.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

Creación de la Alarma de Monitoreo (CloudWatch Alarm)

Finalmente, se enlaza el monitoreo con la notificación para crear la alerta.

Creación de la Alarma: Se optó por Create an alarm para la instancia *i-0d8e8b48d04371e4f (práctica de monitoreo)*

Configuración de la Notificación: Se asoció la alarma al tópico SNS que creamos: “*NotificacionPorCorreo”.*

Definición de Umbrales: Se especificaron las condiciones para que la alarma se active:

1. *Métrica: CPU utilization.*
2. *Agrupación de Muestras: Maximum.*
3. *Condición (Umbral): Cuando el porcentaje de utilización de la CPU sea mayor o igual a (> =) 80%.*
4. *Período: Durante 1 período consecutivo de 5 minutos.*

Nombre de la Alarma: Se nombró la alarma como *awsec2-i-0d8e8b48d04371e4f-GreaterThanOrEqualToThreshold-CPUUtilization.*

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a web page

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

***A screenshot of a computer

AI-generated content may be incorrect.***

Resultado Final

Al final del proceso, la alarma fue creada exitosamente, y ahora el sistema está listo para enviar una alerta por correo electrónico si el uso de la CPU del servidor ***Práctica de monitoreo*** se dispara por encima del ***80%*** durante al menos 5 minutos.

Este es un ejemplo completo de cómo usar la infraestructura de monitoreo y notificación de AWS para asegurar la disponibilidad y rendimiento de un servidor.

Configuración de un Entorno de Microservicios con Istio

Este proyecto consistió en configurar una herramienta de gestión de tráfico llamada Istio en un entorno de Kubernetes (el orquestador de contenedores), que estaba corriendo localmente en Docker Desktop. Luego se desplegaron dos servicios (A y B) para demostrar cómo Istio puede controlar el tráfico entre ellos.

1. Preparación del Entorno Base:

Kubernetes en Docker Desktop: Se activó la función de Kubernetes dentro de Docker Desktop, que esencialmente crea un pequeño "clúster" o entorno de prueba en la propia máquina del usuario.

1. Resultado: Se verificó que el clúster (docker-desktop) estaba listo y corriendo con la versión v1.32.2 de Kubernetes.
2. Instalación de Istio: Istio se instaló usando el comando istioctl install con un perfil de demostración (--set profile=demo).

Resultado: Se confirmó que los componentes principales de Istio (istiod, *gateways*, etc.) se instalaron correctamente y sus *pods* estaban en estado Running en el *namespace* istio-system.

2. Instalación de Herramientas de Monitoreo

Para poder ver qué estaba pasando con el tráfico, se instalaron cuatro herramientas accesorias (los *addons*):

1. Prometheus: Para la recolección de métricas (números y estadísticas del sistema).
2. Kiali: Es el panel visual que nos permite ver un gráfico de tráfico (el *Service Mesh*) entre los servicios.
3. Jaeger y Zipkin: Para el *tracing* (rastreo), que permite seguir la ruta completa de una solicitud a medida que pasa por múltiples servicios.
4. Grafana: Para visualizar los datos recopilados por Prometheus a través de *dashboards*.
5. Resultado: Todos estos *addons* se desplegaron y sus respectivos *pods* también aparecieron en estado Running.

3. Despliegue de los Microservicios (Servicio A y Servicio B)

Se prepararon y desplegaron los dos servicios de prueba que interactuarán dentro de la malla de servicios:

1. Construcción de Imágenes: Se crearon las imágenes de Docker para los servicios service-a y service-b.
2. Etiquetado del *Namespace*: Se activó la "inyección automática" de Istio en el *namespace* default con el comando kubectl label namespace default istio-injection=enabled. Esto asegura que cada *pod* que se despliegue allí recibirá automáticamente un *sidecar* de Istio (una especie de "agente de tráfico").
3. Creación del *Namespace* demo-mesh: Aunque se etiquetó default, los servicios se desplegaron en un *namespace* nuevo llamado demo-mesh y se aplicaron sus archivos de configuración (service-a.yaml, service-b.yaml).
4. Configuración de Tráfico (Gateway y VirtualService): Se crearon las reglas de Istio para permitir que el tráfico externo (entrante) llegara a estos servicios.

4. Prueba y Visualización del Control de Tráfico

Prueba Inicial de Tráfico: Se usó el comando curl para enviar solicitudes al servicio.

Comportamiento de Istio: Por defecto, Istio aplica una estrategia de Round Robin (o balanceo de carga equitativo), lo que significa que la solicitud se alternaba entre el Servicio A y el Servicio B.

Visualización en Kiali: El panel de Kiali mostró el gráfico de tráfico:

Se observó un flujo de tráfico donde el ingressgateway (la puerta de entrada de Istio) dirigía las solicitudes al Servicio B, y este, a su vez, reenviaba la solicitud al Servicio A.

Prueba de Resiliencia (Balanceo de Carga): Para demostrar el balanceo de carga:

Se eliminó manualmente una de las instancias (pods) del Servicio B (kubectl delete pod).

Resultado: Kubernetes/Istio automáticamente iniciaron un nuevo pod para reemplazar el eliminado, manteniendo el servicio en ejecución y demostrando que el sistema es tolerante a fallos (resiliente)

En resumen

Se logró establecer un entorno de microservicios, se instaló la malla de servicios Istio para controlar el tráfico entre ellos, se visualizaron las interacciones en el panel Kiali, y se demostró que el sistema puede equilibrar la carga y recuperarse de la pérdida de una instancia de servicio.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

Extensión y Cumplimiento del Pipeline CI/CD

Esta fase se centra en asegurar que los microservicios, además de ser operacionales (monitoreados con CloudWatch/Istio), cumplan con los estándares de calidad y seguridad definidos, integrando estas validaciones directamente en el flujo de entrega continua (CI/CD).

Aplicación de Políticas de Cumplimiento con SonarQube

Para garantizar la calidad y seguridad del código antes del despliegue, se integró la plataforma SonarQube en la fase de análisis del pipeline CI.

1. Propósito: Analizar estáticamente el código fuente de service-a y service-b para detectar errores, *bugs*, código duplicado y vulnerabilidades de seguridad.
2. Quality Gate: Se configuró un "Quality Gate" estricto que exige que el código no contenga *vulnerabilidades críticas* ni *errores mayores*.

Evidencia de Cumplimiento:

1. Regla de Branch Protection (GitHub/GitLab): Se configuraron reglas en el repositorio de código fuente para la rama principal (*main/master*) que obligan a que el estado de la revisión de SonarQube sea exitoso antes de permitir una fusión (*merge*), asegurando una revisión mínima antes de la integración.
2. Análisis Exitoso: Al pasar el Quality Gate, se valida el cumplimiento normativo básico.

Mecanismo de Interrupción Automática (Pipeline Fail-Fast)

El principio Fail-Fast se implementó utilizando el resultado del análisis de SonarQube. El objetivo es detener el pipeline automáticamente si se detecta una falla crítica de calidad o seguridad, protegiendo así el entorno de producción.

1. Prueba de Falla Crítica: se introdujo intencionalmente un fallo de seguridad de nivel "Crítico" en el código del service-a.
2. Comportamiento Esperado: Cuando el pipeline ejecutó el paso de SonarQube, el Quality Gate falló inmediatamente.
3. Resultado: El pipeline fue interrumpido automáticamente, impidiendo que el paso de Despliegue a Kubernetes/Istio se ejecutara.

Conclusión

Este mecanismo asegura que un código que no cumpla con los estándares de seguridad no sea desplegado en el entorno orquestado, garantizando la protección del entorno productivo y el cumplimiento normativo.

Documentación de la Integración para la Toma de Decisiones

La integración de las herramientas de Monitoreo, Métricas y Seguridad es la base del ciclo de retroalimentación continua que permite a los equipos DevOps tomar decisiones técnicas informadas y mejorar la calidad del producto.

| Fase del Pipeline | Herramientas Integradas | Aporte a la Decisión Técnica |
| --- | --- | --- |
| Commit/Build (CI) | SonarQube | Decisión Preventiva: Bloquea el *merge* de código con vulnerabilidades críticas antes de la integración. |
| Deploy (CD) | Istio | Decisión de Despliegue: Permite estrategias de despliegue de bajo riesgo (ej., *canary releases*) controlando el tráfico a la nueva versión para evaluar su estabilidad. |
| Operate (Continuous Monitoring) | CloudWatch / Prometheus / Grafana / Kiali | Decisión Reactiva: Alerta sobre umbrales críticos (ej, CPU > 80%) o aumento de errores. Los *dashboards* de Grafana/Kiali permiten decidir rápidamente si se requiere escalar el recurso o ejecutar un rollback. |

Resultado Final

Al final del proceso, se logró

1. Monitoreo Transparente: La instancia EC2 está monitoreada por CloudWatch con un sistema de alertas por correo SNS.
2. Observabilidad Completa: El entorno orquestado de microservicios con Istio en Kubernetes está dotado de observabilidad a través de Prometheus (Métricas), Jaeger/Zipkin (Trazabilidad) y Kiali/Grafana (Visualización)
3. Calidad Controlada: La integración de SonarQube y el mecanismo de interrupción automática aseguran que la calidad y seguridad sean validadas antes de la entrega.

Este es un ejemplo completo de cómo integrar la Observabilidad y el Cumplimiento en un proceso de entrega, asegurando tanto el rendimiento como la confiabilidad y seguridad de la infraestructura.

Indicaciones para el Uso de Inteligencia Artificial (IA)

Declaramos que la Inteligencia Artificial fue utilizada únicamente como apoyo para mejorar la redacción del informe y sugerir la estructura de la documentación, de acuerdo con la política institucional. Todas las ideas, justificaciones técnicas y análisis son propios del equipo. La herramienta utilizada fue geminis.

Reflexiones Individuales Obligatorias (Sin Uso de IA)

Reflexión de Elias Delgado

Mi contribución principal al proyecto fue la orquestación de los microservicios con Istio en Kubernetes y la configuración de AWS CloudWatch]. El mayor aprendizaje que obtuve fue la comprensión de que la Observabilidad no es solo monitorear CPU, sino establecer la Trazabilidad (*Tracing*) para entender cómo interactúan los servicios y diagnosticar fallas rápidamente utilizando herramientas como Jaeger y Kiali. También comprendí la importancia del principio "Shift Left", al ver cómo la integración de herramientas como SonarQube nos obliga a pensar en la seguridad y calidad del código desde el inicio del ciclo de vida del software, previniendo problemas en producción.

Reflexión de Javiera Aguirre

Mi enfoque principal fue en la integración de SonarQube y la creación del mecanismo de interrupción automática y el diseño de los dashboards en Grafana]. El concepto más valioso que internalicé fue el del Cumplimiento Normativo automatizado. Me di cuenta de que un *pipeline* DevOps debe actuar como un guardián de calidad. Ver cómo el Quality Gate de SonarQube detiene automáticamente el despliegue de código inseguro, me demostró, de forma práctica, cómo la automatización protege el entorno productivo de riesgos que un proceso manual no detectaría. Esto refuerza la idea de que DevOps busca no solo velocidad, sino también fiabilidad y seguridad continua.