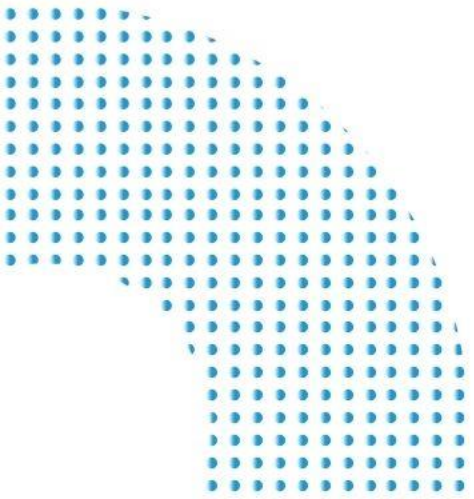

entrenamientocertero
Microsoft
Learning Partner
aws
PARTNER
SCRUMstudy
Targeting success
VMEdu
Authorized Training Partner
EXIN
Partner
FORTINET
CertiProf | Partner

Laboratorio: Deploying Apps to Google Kubernetes Engine

Aldo Trucios

Agenda

- Objetivo
- Task 01: Descargar una aplicación de muestra desde GitHub
- Task 02: Implementar en Kubernetes con Cloud Build y Artifact Registry



Objetivos

En este laboratorio, aprenderá a realizar las siguientes tareas:

- Descargue una aplicación de muestra desde GitHub
- Implementar en Kubernetes Engine

Task 01: Descargar una aplicación de muestra desde GitHub

Descargue una aplicación de muestra de GitHub y obtenga una vista previa en Cloud Shell.

1. En Cloud Console, haga clic en **Activar Cloud Shell**.
2. Si se le solicita, haga clic en **Continuar** .
3. Para crear una nueva carpeta, ejecute el siguiente comando:

```
mkdir gcp-course
```

4. Cambia a la carpeta que acabas de crear:

```
cd gcp-course
```

5. Clonar una aplicación Python Flask simple desde GitHub:

```
git clone https://GitHub.com/GoogleCloudPlatform/training-data-analyst.git
```

6. Cambiar a la carpeta deploying-apps-to-gcp:

```
cd training-data-analyst/courses/design-process/deploying-apps-to-gcp
```

Task 01: Descargar una aplicación de muestra desde GitHub

7. Para probar el programa, escriba el siguiente comando para crear un contenedor Docker de la imagen:

```
docker build -t test-python .
```

8. Para ejecutar la imagen de Docker, escriba el siguiente comando:

```
docker run --rm -p 8080:8080 test-python
```

9. Para ver el programa en ejecución, haga clic en Vista previa web (Icono de vista previa web) en la barra de herramientas de Google Cloud Shell. Luego, seleccione Vista previa en el puerto 8080.

Task 02: Implementar en Kubernetes Engine con Cloud Build y Artifact Registry

Kubernetes Engine le permite crear un clúster de máquinas e implementar cualquier cantidad de aplicaciones en él. Kubernetes abstrae los detalles de la administración de las máquinas y le permite automatizar la implementación de sus aplicaciones con comandos CLI simples.

Para implementar una aplicación en Kubernetes, primero debe crear el clúster. Luego, debe agregar un archivo de configuración para cada aplicación que implementará en el clúster.

1. En el menú de navegación, haga clic en Kubernetes Engine. Si aparece un mensaje que indica que se está inicializando la API de Kubernetes, espere a que finalice.
2. Haga clic en crear cluster.
3. Acepte todos los valores predeterminados, seleccione la región haga clic en **Crear**. La creación del clúster de Kubernetes Engine demorará un par de minutos. Cuando el clúster esté listo, aparecerá una marca de verificación verde.
4. Haga clic en los tres puntos a la derecha del clúster y luego haga clic en Conectar.
5. En la pantalla Conectarse al clúster , haga clic en Ejecutar en Cloud Shell . Esto abre Cloud Shell y el comando de conexión se ingresa automáticamente.

Task 02: Implementar en Kubernetes Engine con Cloud Build y Artifact Registry

Descargue una aplicación de muestra de GitHub y obtenga una vista previa en Cloud Shell.

6. Presione Enter para conectarse al clúster.
7. Para probar su conexión, escriba el siguiente comando; este comando simplemente muestra las máquinas de tu clúster. Si funciona, estás conectado.

```
Kubectl get nodes
```

8. En Cloud Shell, haga clic en **Abrir editor.**
9. Expande la carpeta "training-data-analyst/courses/design-process/deploying-apps-to-gcp" en el panel de navegación de la izquierda. Luego, haz clic en main.py para abrirla.
10. En la función “main()” cambia el titulo por “Hello Kubernetes Engine” como se muestra a continuación.

```
@app.route("/")
def main():
            model = {"title" "Hello Kubernetes Engine"}
            return render_template('index.html', model=model)
```

Task 02: Implementar en Kubernetes Engine con Cloud Build y Artifact Registry

11. Guarda tus cambios.
12. Agrega un archivo llamado kubernetes-config.yaml en el folder “training-data-analyst/courses/design-process/deploying-apps-to-gcp”.
13. Pega el siguiente código en el archivo para configurar la aplicación.

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: devops-deployment
  labels:
    app: devops
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: devops
      tier: frontend
  template:
    metadata:
      labels:
        app: devops
        tier: frontend
    spec:
      containers:
      - name: devops-demo
        image: <YOUR IMAGE PATH HERE>
        ports:
        - containerPort: 8080

---
apiVersion: v1
kind: Service
metadata:
  name: devops-deployment-lb
  labels:
    app: devops
    tier: frontend-lb
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: devops
    tier: frontend
```

Task 02: Implementar en Kubernetes Engine con Cloud Build y Artifact Registry

14. En **Cloud Shell,** escriba el siguiente comando para crear un repositorio de Artifact Registry llamado devops-demo:

```
gcloud artifacts repositories create devops-demo \
--repository-format=docker \
--location="REGION"
```

15. Para configurar Docker para que se autentique en el repositorio Docker de Artifact Registry, escriba el siguiente comando:

```
gcloud auth configure-docker "REGION"-docker.pkg.dev
```

16. Para utilizar Kubernetes Engine, debe crear una imagen de Docker. Escriba los siguientes comandos para utilizar Cloud Build para crear la imagen y almacenarla en Artifact Registry:

```
cd ~/gcp-course/training-data-analyst/courses/design-process/deploying-apps-to-gcp
gcloud builds submit --tag "REGION"-docker.pkg.dev/$DEVSHELL_PROJECT_ID/devops-demo/devops-image:v0.2
```

Task 02: Implementar en Kubernetes Engine con Cloud Build y Artifact Registry

17. Cuando se complete el comando anterior, el nombre de la imagen aparecerá en la salida. El nombre de la imagen tiene el formato REGION-docker.pkg.dev/PROJECT_ID/devops-demo/devops-image:v0.2.
18. Resalte el nombre de la imagen y cópielo en el portapapeles. Pegue ese valor en el archivo kubernetes-config.yaml y sobrescriba la cadena <YOUR IMAGE PATH HERE>.
19. Deberías ver algo similar a lo siguiente:

```
spec:
  containers:
  - name: devops-demo
  image: "REGION"-docker.pkg.dev/PROJECT_ID/devops-demo/devops-image:v0.2
  ports:
```

Task 02: Implementar en Kubernetes Engine con Cloud Build y Artifact Registry

20. Escriba el siguiente comando de Kubernetes para implementar su aplicación:

```
kubectl apply -f kubernetes-config.yaml
```

21. En el archivo de configuración se especificaron tres réplicas de la aplicación. Escriba el siguiente comando para ver si se crearon tres instancias:

```
kubectl get pods
```

22. Asegúrate de que todas las cápsulas estén listas. Si no lo están, espera unos segundos y vuelve a intentarlo.
23. También se agregó un balanceador de carga en el archivo de configuración. Escriba el siguiente comando para ver si se creó:

```
kubectl get services
```

Task 02: Implementar en Kubernetes Engine con Cloud Build y Artifact Registry

24. Deberías ver algo similar a lo siguiente:

```
NAME                  TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
devops-deployment-lb  LoadBalancer   10.12.6.226   34.69.232.18   80:30406/TCP   3m11s
kubernetes            ClusterIP      10.12.0.1     <none>         443/TCP        41m
```

25. Si la dirección IP externa del balanceador de carga dice "pendiente", espere unos segundos y vuelva a intentarlo.

26. Cuando tengas una IP externa, abre una pestaña del navegador y hazle una solicitud. Debería devolver Hello Kubernetes Engine. Puede que tarde unos segundos en estar lista.

¡Gracias!

Contáctanos!

www.ec-elearning.com

+51 973 899 009

www.facebook.com/EntrenamientoCertero

entrenamiento.certero

www.linkedin.com/company/entrenamiento-certero