--------------------------------------------------------------------------------------------------

Cada que se hace un cambio en el código:

git add . && git commit -m "Modificacion de README"

git push

--------------------------------------------------------------------------------------------------------

Fase 1:

uvicorn app.main:app --reload

curl -X GET http://localhost:8000/generate-jwt

curl -X POST http://localhost:8000/DevOps \

-H "X-Parse-REST-API-Key: 2f5ae96c-b558-4c7b-a590-a501ae1c3f6c" \

-H "X-JWT-KWY: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoidGVzdCIsImV4cCI6MTc0NzcxNDA2M30.pGAJ8x2eqBloEBGh7GlsngZg6VqVo8XGuS\_NQJVjQis" \

-H "Content-Type: application/json" \

-d '{

"message": "Hola",

"to": "Henry",

"from": "Azure",

"timeToLifeSec": 45

}'

--------------------------------------------------------------------------------------------------------

Fase 2:

- Construir imagen:

docker build -t devops-microservice:local .

**Nota**: Si cambiamos el dockerfile hay que reconstruir la imagen y para eso primero hay que borrarla y volverla a crear.

Lo mismo con el contenedor.

Cambios que requieren reconstrucción de la imagen:

Modificas el Dockerfile

Cambias el requirements.txt

Cambias código fuente y usas COPY . . en el Dockerfile

Cambias la estructura de carpetas o rutas copiadas al contenedor

Quieres cambiar la imagen base (FROM python:...)

Actualizas dependencias (pip, apt, etc.)

Cambias la CMD o ENTRYPOINT

- Cosntruir contenedor:

docker rm -f devops-microservice

docker rmi devops-microservice:local

docker build -t devops-microservice:local .

Correr contenedor sin pasar las claves (recomendado):

docker run -d --name devops-microservice \

--env-file .env \

-p 8000:8000 \

devops-microservice:local

Correr contenedor pasando las claves:

docker run --name devops-microservice \

-p 8000:8000 \

--env SECRET\_KEY=clave\_super\_secreta\_segura \

--env API\_KEY=2f5ae96c-b558-4c7b-a590-a501ae1c3f6c \

devops-microservice:local

docker start -ai devops-microservice

Para entrar al contenedor:

docker exec -it devops-microservice /bin/sh

Acceder en el navegador: http://localhost:8000/generate-jwt

y ejecutar el curl ...

------------------------------------------------------------------------------

Empaquetar tu imagen en un archivo .tar para compartirla (siempre y cuando pasamos las claves de .env directamente en

el comando de docker para crear el contenedor):

docker save -o devops\_microservice.tar devops-microservice:local

Ese .tar pasamos a la persona y el ejecutará esto:

docker load -i /ruta/donde/guardo/devops\_microservice.tar

Una ves carga la imagen, el ya puede correr el contenedor si esta en Linux:

docker run --name devops-microservice \

-p 8000:8000 \

--env SECRET\_KEY=clave\_super\_secreta\_segura \

--env API\_KEY=2f5ae96c-b558-4c7b-a590-a501ae1c3f6c \

devops-microservice:local

O si estan en windows en powershell o CMD con:

docker run --name devops-microservice ^

-p 8000:8000 ^

--env SECRET\_KEY=clave\_super\_secreta\_segura ^

--env API\_KEY=2f5ae96c-b558-4c7b-a590-a501ae1c3f6c ^

devops-microservice:local

Y ya puede probar con: http://localhost:8000/generate-jwt

------------------------------------------------------------------------------------

Fase 3:

docker build -t devops-microservice:v7 .

docker tag devops-microservice:v4 devopsregistryhenry.azurecr.io/devops-microservice:v7

docker push devopsregistryhenry.azurecr.io/devops-microservice:v7

curl -X POST https://devops-ms-henry-hhfsbjcubeh5ehg3.eastus-01.azurewebsites.net/DevOps \

-H "X-Parse-REST-API-Key: 2f5ae96c-b558-4c7b-a590-a501ae1c3f6c" \

-H "X-JWT-KWY: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoidGVzdCIsImV4cCI6MTc0MzU2Mjc5MX0.8m2XG-mU61bV2bohMOAMR2x-zdj0sDzd6QzXDjupEzU" \

-H "Content-Type: application/json" \

-d '{

"message": "Hola",

"to": "Henry",

"from": "Azure",

"timeToLifeSec": 45

}'

http://localhost:8000/generate-jwt

Prender el contenedor con:

docker run -d -p 8000:8000 --name devops-microservice devops-microservice:local

---

generar jwt

python -m app.utils.generate\_jwt

---

--------------- AKS

Creación

az aks create \

--resource-group devops \

--name aks-devops-henry \

--node-count 2 \

--generate-ssh-keys

conectar terminal al cluster:

az aks get-credentials --resource-group devops --name aks-devops-henry

Verificación de estar conectado.

kubectl get nodes

Verificar imagen en ACR

az acr repository list --name devopsregistryhenry --output table

tener IP publica

kubectl get service devops-service

get:

http://134.33.251.5/generate-jwt

post:

curl -X POST http://134.33.251.5/DevOps \

-H "Content-Type: application/json" \

-H "X-Parse-REST-API-Key: 2f5ae96c-b558-4c7b-a590-a501ae1c3f6c" \

-H "X-JWT-KWY: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyIjoidGVzdCIsImV4cCI6MTc0MzkxNDcyNX0.grcmx9gUWXbx1qT4X7NKQrOci9x0IIgRT0IIY6UghiQ" \

-d '{

"message": "This is a test",

"to": "Juan Perez",

"from": "Rita Asturia",

"timeToLifeSec": 45

}'

\*\*\*\*\*\*

Paso 1. Crear la Infra:

terraform init

terraform plan

terraform apply -auto-approve

Paso 2.

az login

az account show

az account set --subscription "Nombre-de-tu-suscripcion"

Autenticarse en el ACR

az acr login --name devopsregistryhenry2025

az acr login -n devopsregistryhenry2025 --expose-token

Puedes validar con:

az acr list --resource-group DevOpsRG --query "[].{acrLoginServer:loginServer}" --output table

az acr credential show --name devopsregistryhenry2025

{

"passwords": [

{

"name": "password",

"value": "EQnnhodIP+fNsM8hriCJ46L6bs4phA5oocTq03ezMY+ACRCxGKq6"

},

{

"name": "password2",

"value": "SjbVM6omhmLSwmWAOBJSJJQ/ULvgjahtiHL37r7/E3+ACRCh7igi"

}

],

"username": "devopsregistryhenry2025"

}

docker login devopsregistryhenry2025.azurecr.io

Construir la imagen local Desde la raíz de tu proyecto:

docker build -t devops-microservice:latest .

Etiquetar la imagen con la URL del ACR

Necesitamos “taggear” la imagen local para empujarla al ACR:

docker tag devops-microservice:latest devopsregistryhenry2025.azurecr.io/devops-microservice:latest

Hacer push al ACR para subir la imagen:

docker push devopsregistryhenry2025.azurecr.io/devops-microservice:latest

Verificar en ACR

Puedes comprobar que está en tu registro:

az acr repository list --name devopsregistryhenry2025 --output table

az acr repository show-tags --name devopsregistryhenry2025 --repository devops-microservice:latest --output table

Paso 3.

Asegurarnos de tener acceso al AKS

Primero verifica que puedes conectarte a tu cluster AKS:

az aks get-credentials --resource-group DevOpsRG --name devops-aks --overwrite-existing

kubectl get nodes

Si ves tus nodos, ya estás conectado.

Paso 4

¡Súper! Vamos a dejarte **los manifests completos + el paso a paso** para que el evaluador pueda hacer su curl https://${HOST}/DevOps sin comprar dominio ni nada raro.

La vía más simple y realista en Azure es:

* **Ingress NGINX** (Service LoadBalancer propio) → obtiene **IP pública** de Azure.
* Le ponemos un **DNS label** a esa IP pública para tener un **FQDN gratuito**: <label>.<region>.cloudapp.azure.com.
* Generamos **certificado TLS válido** con **Let’s Encrypt** (vía cert-manager) para ese FQDN.
* Tu **Service** del microservicio queda como **ClusterIP** y es el Ingress el que expone HTTPS.

**0) Preparación (una vez por cluster)**

**0.1 Instala NGINX Ingress Controller**

# Namespace para el Ingress

kubectl create namespace ingress-nginx

# Instalar Ingress NGINX por Helm (recomendado)

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm install ingress-nginx ingress-nginx/ingress-nginx \

--namespace ingress-nginx

Comprueba la IP pública:

kubectl get svc -n ingress-nginx ingress-nginx-controller

**0.2 Coloca un DNS label a esa IP pública**

Obtenemos la IP:

PIP=$(kubectl get svc -n ingress-nginx ingress-nginx-controller \

-o jsonpath='{.status.loadBalancer.ingress[0].ip}')

echo $PIP

Identificamos el **resource group** “MC\_…” donde Azure crea los recursos del cluster y el nombre del Public IP:

# Obtén el RG de nodos (node resource group)

az aks show -g DevOpsRG -n devops-aks --query nodeResourceGroup -o tsv

# Usa ese RG para encontrar el Public IP por su dirección

MC\_RG=$(az aks show -g DevOpsRG -n devops-aks --query nodeResourceGroup -o tsv)

az network public-ip list -g $MC\_RG --query "[?ipAddress=='$PIP'].[name,resourceGroup,location]" -o tsv

Crea un **DNS label** (elige uno único, ej. devops-henry-123):

PIP\_NAME=$(az network public-ip list -g $MC\_RG --query "[?ipAddress=='$PIP'].name" -o tsv)

REGION=$(az network public-ip list -g $MC\_RG --query "[?ipAddress=='$PIP'].location" -o tsv)

az network public-ip create \

-g $MC\_RG \

-n devopsIngressIP \

--sku Standard \

--allocation-method static \

--dns-name devops-henry-2025

Resultado:

[Coming breaking change] In the coming release, the default behavior will be changed as follows when sku is Standard and zone is not provided: For zonal regions, you will get a zone-redundant IP indicated by zones:["1","2","3"]; For non-zonal regions, you will get a non zone-redundant IP indicated by zones:null. { "publicIp": { "ddosSettings": { "protectionMode": "VirtualNetworkInherited" }, "dnsSettings": { "domainNameLabel": "devops-henry-2025", "fqdn": "devops-henry-2025.eastus.cloudapp.azure.com" }, "etag": "W/\"b8b7dbf7-a0c1-4398-9605-af783c245187\"", "id": "/subscriptions/673cc71e-b603-4e80-8ceb-7a980df54561/resourceGroups/MC\_DevOpsRG\_devops-aks\_eastus/providers/Microsoft.Network/publicIPAddresses/devopsIngressIP", "idleTimeoutInMinutes": 4, "ipAddress": "20.168.209.54", "ipTags": [], "location": "eastus", "name": "devopsIngressIP", "provisioningState": "Succeeded", "publicIPAddressVersion": "IPv4", "publicIPAllocationMethod": "Static", "resourceGroup": "MC\_DevOpsRG\_devops-aks\_eastus", "resourceGuid": "9a124540-ddef-4af3-a38e-ebf70d91c75e", "sku": { "name": "Standard", "tier": "Regional" }, "type": "Microsoft.Network/publicIPAddresses" } }

**Instalar (o reconfigurar) NGINX Ingress Controller con Helm**

Primero asegurémonos de que tienes Helm inicializado en tu cluster:

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

**Seguir usando el namespace original (ingress-nginx)**

Ya tienes un Ingress Controller instalado ahí, entonces solo lo **actualizamos** en el mismo namespace para que use la IP estática que creaste:

helm upgrade --install ingress-nginx ingress-nginx/ingress-nginx \

--namespace ingress-nginx \

--set controller.replicaCount=2 \

--set controller.nodeSelector."kubernetes\.io/os"=linux \

--set defaultBackend.nodeSelector."kubernetes\.io/os"=linux \

--set controller.service.loadBalancerIP="20.168.209.54" \

--set controller.service.annotations."service\.beta\.kubernetes\.io/azure-dns-label-name"="devops-henry-2025"

Después verificas:

kubectl get svc -n ingress-nginx

y deberías ver el **LoadBalancer** en 20.168.209.54.

**Re-enganchar el Ingress Controller a la IP estática (20.168.209.54)**

Esto recrea solo el Service del controller para que tome la IP pública que ya creamos:

# 1. Borramos el Service (Helm lo recrea sin romper el release)

kubectl delete svc ingress-nginx-controller -n ingress-nginx

# 2. Upgrade con la IP y el label DNS

helm upgrade --install ingress-nginx ingress-nginx/ingress-nginx \

--namespace ingress-nginx \

--set controller.replicaCount=2 \

--set controller.nodeSelector."kubernetes\.io/os"=linux \

--set defaultBackend.nodeSelector."kubernetes\.io/os"=linux \

--set controller.service.loadBalancerIP="20.168.209.54" \

--set controller.service.annotations."service\.beta\.kubernetes\.io/azure-dns-label-name"="devops-henry-2025"

Verifica que ahora sí tome **tu** IP:

kubectl get svc -n ingress-nginx ingress-nginx-controller

# Debe mostrar EXTERNAL-IP = 20.168.209.54

Y que el FQDN resuelva a esa IP:

nslookup devops-henry-2025.eastus.cloudapp.azure.com

**Instalar cert-manager y ClusterIssuer (Let’s Encrypt)**

Si aún no lo instalaste:

kubectl create namespace cert-manager

helm repo add jetstack https://charts.jetstack.io

helm repo update

helm install cert-manager jetstack/cert-manager \

--namespace cert-manager \

--create-namespace \

--set crds.enabled=true

Para borrar:

kubectl delete deployment devops-microservice

kubectl delete service devops-service

kubectl delete ingress devops-ingress

kubectl delete secret devops-secrets

kubectl delete secret devops-tls-secret

az network public-ip delete --resource-group DevOpsRG --name devops-ingress-ip

kubectl apply -f k8s/

kubectl delete -f k8s/

terraform destroy -auto-approve