Taller 1: Construcción de pipelines en Cloud

MANUEL HERRERA
COLLIN GONZALEZ

Arquitectura y Patrones de Diseño Cloud

RETRY PATTERN

El Retry Pattern implementa una estrategia de reintentos automáticos para operaciones que pueden fallar temporalmente. En nuestro Users API, cuando una operación falla, el sistema automáticamente reintenta la operación hasta 3 veces con un intervalo de 2 segundos entre cada intento, mejorando la probabilidad de éxito en operaciones que pueden fallar por problemas temporales de red o recursos.

CIRCUIT BREAKER

El Circuit Breaker actúa como un interruptor en nuestro sistema. Cuando un servicio (como el Auth API) detecta múltiples fallos al comunicarse con otro servicio (como Users API), automáticamente "abre el circuito" y redirige las peticiones a una respuesta de fallback, evitando así que el sistema se sobrecargue y permitiendo que se recupere.

Diagrama

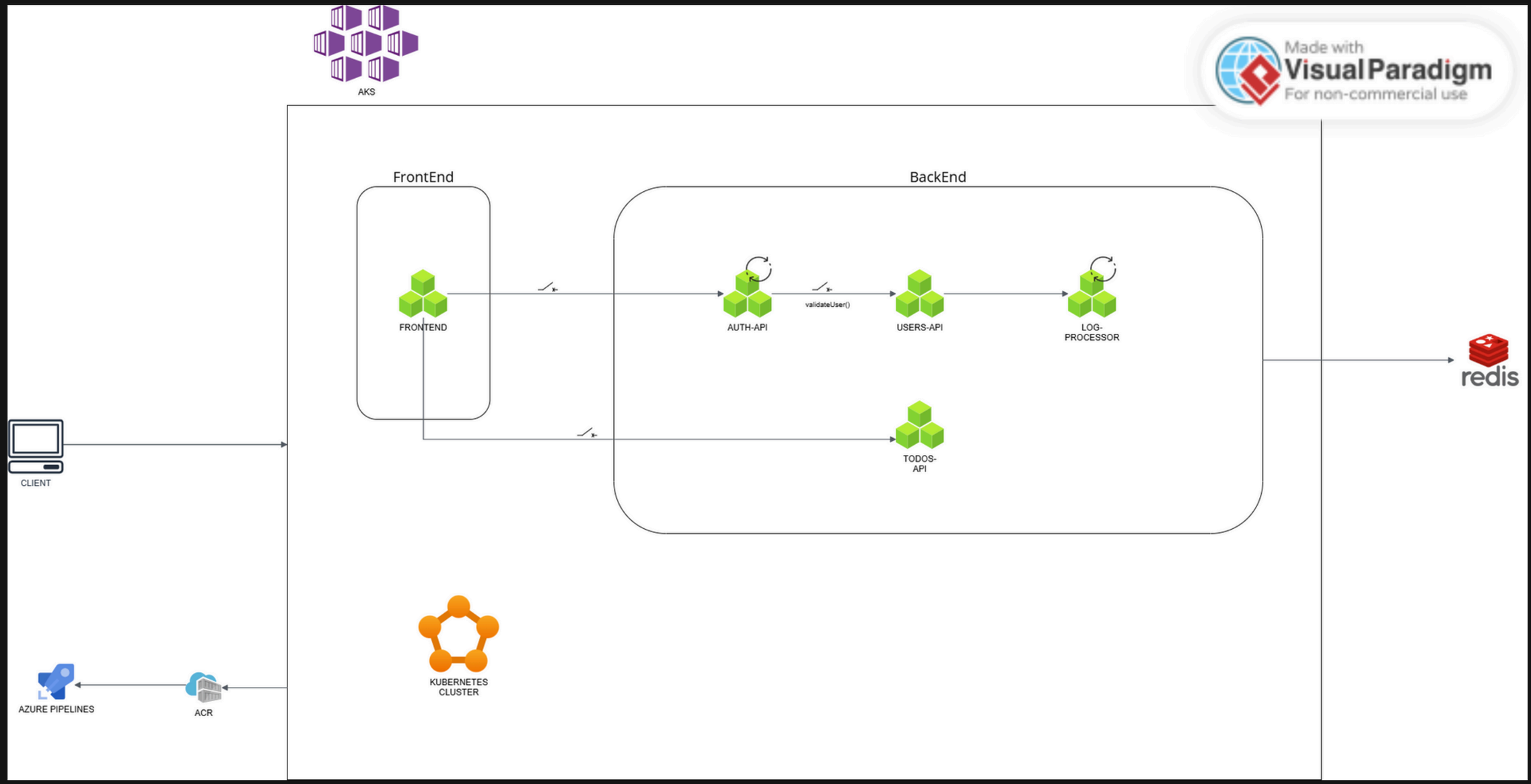
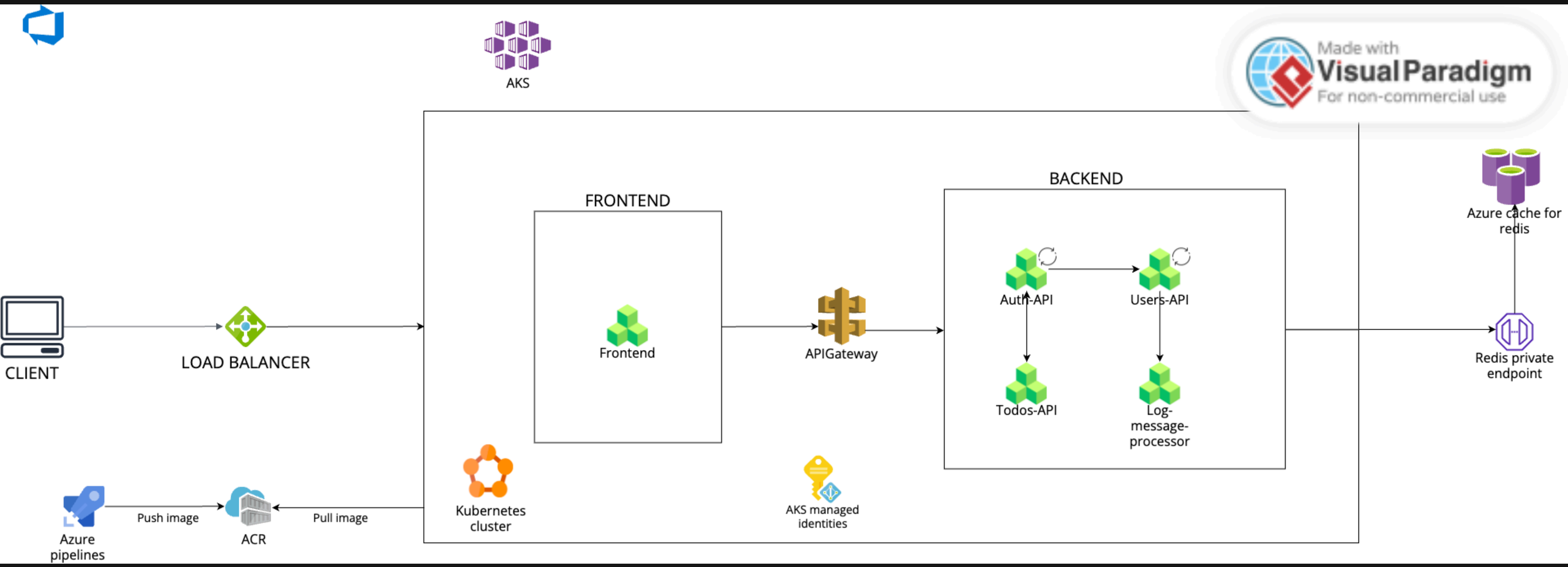
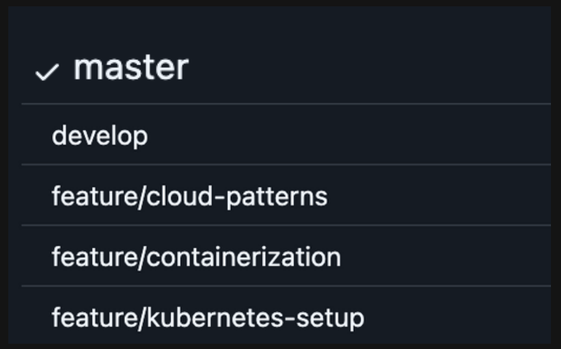

AKS
Made with
Visual Paradigm
For non-commercial use
FrontEnd
BackEnd
FRONTEND
AUTH-API
validateUser()
USERS-API
LOG-
PROCESSOR
redis
TODOS-
API
CLIENT
KUBERNETES
CLUSTER
AZURE PIPELINES
ACR

Diagrama ideal

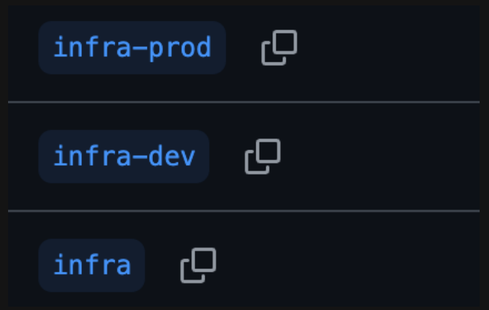


Estrategia de branching



DESARROLLADORES

Organizamos nuestro desarrollo con ramas feature/ para nuevas funcionalidades, release/ para versiones estables y hotfix/ para correcciones urgentes. Todo fluye desde develop hasta master, asegurando un desarrollo controlado y ordenado.



INFRAESTRUCTURA

Mantenemos ramas separadas para infraestructura: infra para cambios generales, infra/dev para pruebas y infra/prod para producción. Esta estructura nos permite gestionar de forma segura los cambios en Kubernetes y pipelines.

Pipelines de desarrollo

1. PIPELINE FRONTEND

Propósito: Construcción y despliegue de la interfaz de usuario
Tecnología: Node.js, React

2. PIPELINE AUTH API

Propósito: Servicio de autenticación y autorización
Tecnología: Go

3. PIPELINE USERS API

Propósito: Gestión de usuarios y perfiles
Tecnología: Java/Spring Boot

4. PIPELINE TODOS API

Propósito: Gestión de tareas y lista de TODOs
Tecnología: Node.js

Pipelines de desarrollo

```yaml
trigger:
  branches:
    include:
      - master
  paths:
    include:
      - users-api/*

pool:
  vmImage: 'ubuntu-latest'

variables:
  imageRepository: 'users-api'
  containerRegistry: 'microservicesacr2025.azurecr.io'
  tag: '$(Build.BuildId)'

stages:
- stage: Build
  displayName: Build and Push
  jobs:
  - job: Build
    steps:
    - task: Docker@2
      displayName: Build and Push Image
      inputs:
        containerRegistry: 'acr-microservices-connection'
        repository: '$(imageRepository)'
        command: 'buildAndPush'
        Dockerfile: 'users-api/Dockerfile'
        tags: |
          $(tag)
          latest
```

Implementamos un pipeline de CI/CD (Integración Continua/Entrega Continua) utilizando Azure DevOps. Este pipeline automatiza completamente el proceso de desarrollo y despliegue, y se divide en dos etapas principales:

ETAPA DE CONSTRUCCIÓN (BUILD STAGE):

- Se activa automáticamente cuando hay cambios en la rama master
- Utiliza una imagen de Ubuntu como entorno de construcción
- Construye una imagen Docker del microservicio
- Etiqueta la imagen con un número de build único y 'latest'
- Sube la imagen al Azure Container Registry

Pipelines de desarrollo

```yaml
- stage: Deploy
  displayName: Deploy to AKS
  dependsOn: Build
  condition: succeeded()
  jobs:
  - job: Deploy
    steps:
    - task: Kubernetes@1
      displayName: Check Cluster Status
      inputs:
        connectionType: 'Kubernetes Service Connection'
        kubernetesServiceEndpoint: 'aks-microservices-connection'
        command: 'get'
        arguments: 'nodes'

    - task: Kubernetes@1
      displayName: Create Namespace
      inputs:
        connectionType: 'Kubernetes Service Connection'
        kubernetesServiceEndpoint: 'aks-microservices-connection'
        command: 'apply'
        useConfigurationFile: true
        inline: |
          apiVersion: v1
          kind: Namespace
          metadata:
            name: microservices
```

ETAPA DE DESPLIEGUE (DEPLOY STAGE):

- Se ejecuta solo si la etapa de construcción fue exitosa
- Despliega la aplicación en nuestro cluster de Kubernetes (AKS)
- Aplica la configuración de despliegue usando archivos YAML
- Verifica el estado del despliegue y los pods

```yaml
    - task: Kubernetes@1
      displayName: Deploy Application
      inputs:
        connectionType: 'Kubernetes Service Connection'
        kubernetesServiceEndpoint: 'aks-microservices-connection'
        command: 'apply'
        arguments: '-f users-api/k8s/deployment.yml'

    - task: Kubernetes@1
      displayName: Update Image
      inputs:
        connectionType: 'Kubernetes Service Connection'
        kubernetesServiceEndpoint: 'aks-microservices-connection'
        command: 'set'
        arguments: 'image deployment/users-api users-api=$(containerRegistry)/$(imageRepository):$(tag) -n micr

    - task: Kubernetes@1
      displayName: Get Pod Status
      inputs:
        connectionType: 'Kubernetes Service Connection'
        kubernetesServiceEndpoint: 'aks-microservices-connection'
        command: 'get'
        arguments: 'pods -n microservices -l app=users-api -o wide'

    - task: Kubernetes@1
      displayName: Get Pod Logs
      condition: succeededOrFailed()
      inputs:
        connectionType: 'Kubernetes Service Connection'
        kubernetesServiceEndpoint: 'aks-microservices-connection'
        command: 'logs'
        arguments: '-l app=users-api -n microservices --tail=100'
```

Pipelines de infrestructura

1. PIPELINE DE SUBIDA

Ubicación: infra/azure-pipeline.yml
Propósito: Despliega la infraestructura base, configura namespace, redis, zipkin e ingress. Escala los servicios a 1.

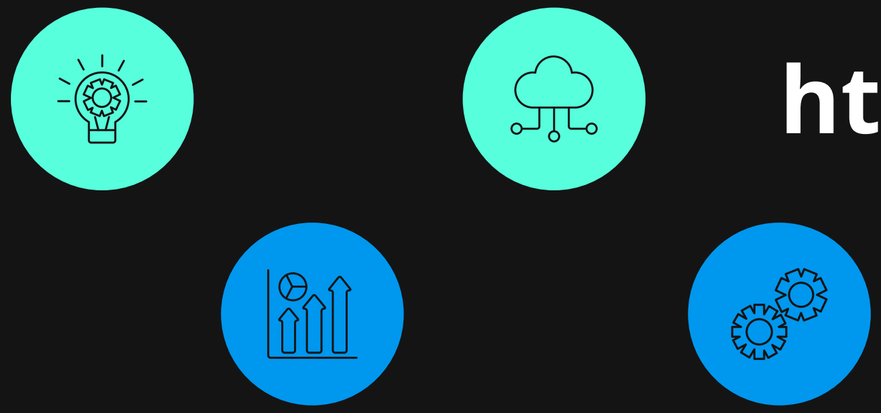
2. PIPELINE DE BAJADA

Ubicación: infra/stop.yml
Propósito: Escala todos los servicios a 0, se ejecuta cuando necesitamos detener la infraestructura.

Pipelines de infraestructura

```yaml
trigger:
  branches:
    include:
      - master
  paths:
    include:
      - infra/*

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: KubernetesManifest@0
  inputs:
    action: 'deploy'
    kubernetesServiceConnection: 'aks-microservices-connection'
    namespace: 'microservices'
    manifests: |
      infra/k8s/namespace.yml
      infra/k8s/redis.yml
      infra/k8s/zipkin.yml
      infra/k8s/ingress.yml

- task: Kubernetes@1
  inputs:
    connectionType: 'Kubernetes Service Connection'
    kubernetesServiceEndpoint: 'aks-microservices-connection'
    command: 'get'
    arguments: 'pods,services,ingress -n microservices'

# Escalar cada deployment a 1 réplica
- task: Kubernetes@1
  inputs:
    connectionType: 'Kubernetes Service Connection'
    kubernetesServiceEndpoint: 'aks-microservices-connection'
    command: 'scale'
    arguments: 'deployment -n microservices frontend auth-api todos-api users-api log-message-processor zipkin redis --replicas=1'
```

Demostracion



http://74.179.236.206/

Gracias