

Modernización a microservicios nativos en la Nube con Microsoft Azure Kubernetes Service

Nombre: Steven Vallejo Sacoto

Profesión: Tecnólogo Superior en Ciberseguridad

Fecha del informe: 14/10/2025

Descripción:

Este proyecto se realizó utilizando **Fedora (Linux)**.

Se simula la modernización de una aplicación monolítica "on-premise" a una arquitectura de microservicios nativa en la nube utilizando una prueba gratuita de Azure. Se despliega una aplicación de varios niveles (frontend, API de negocio, base de datos) en **Azure Kubernetes Service (AKS)**. La infraestructura completa, incluyendo las redes, balanceadores de carga y el clúster de AKS, se definirá y desplegará usando **Bicep (IaC)**. Se implementará **Azure Monitor for Containers** para obtener visibilidad total del rendimiento y la salud de la aplicación.

Próximamente actualización del índice

Resumen ejecutivo y análisis de costo

Resumen de servicios implementados

Se ha desplegado con éxito una arquitectura de microservicios nativa en la nube en Microsoft Azure, simulando la modernización de una aplicación monolítica. La solución es resiliente, escalable y observable.

Arquitectura y Servicios Clave:

Servicio de Azure	Propósito en el Proyecto	Tipo de Servicio
Azure Kubernetes Service (AKS)	Orquestador de contenedores. Gestiona el ciclo de vida de la aplicación.	PaaS (Contenedores)
Azure Container Registry (ACR)	Registro privado para almacenar y gestionar las imágenes de Docker de forma segura.	PaaS
Azure Virtual Network (VNet)	Proporciona una red privada y aislada para el clúster de AKS, mejorando la seguridad.	IaaS (Redes)
Azure Load Balancer	Creado automáticamente por el Service de Kubernetes para exponer la aplicación a Internet.	PaaS (Redes)

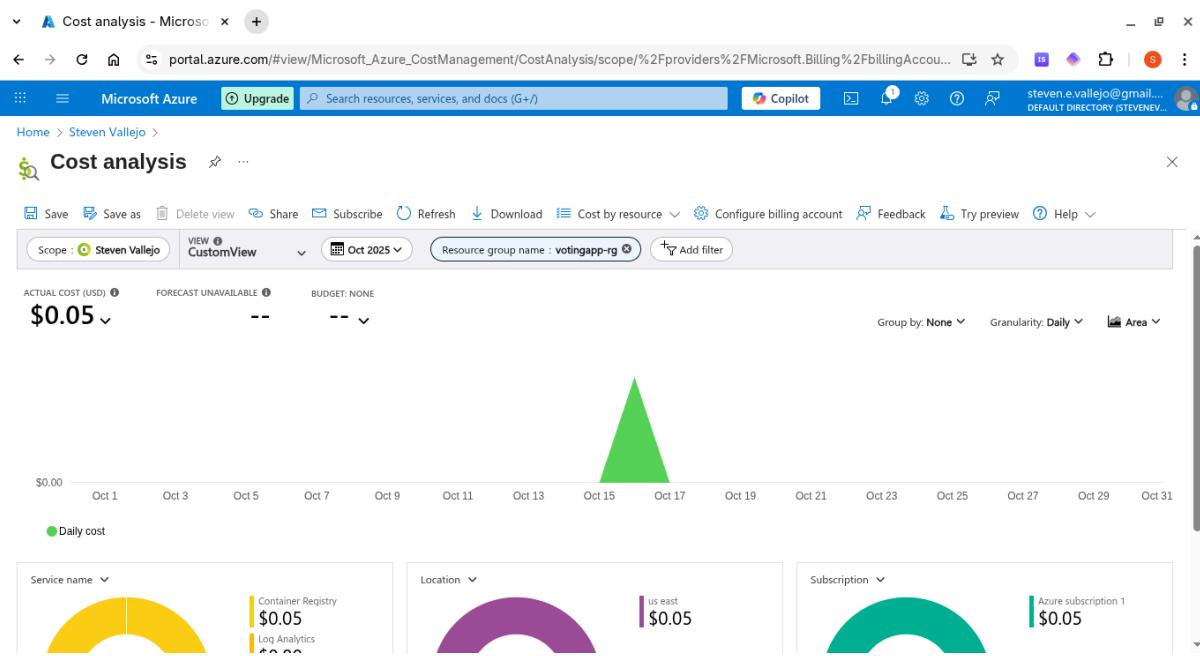
Azure Monitor for Containers	Solución de monitoreo que proporciona visibilidad total sobre el rendimiento y la salud del clúster y los contenedores.	PaaS (Monitoreo)
Log Analytics Workspace	Repositorio centralizado para almacenar logs y métricas de todos los componentes.	PaaS
Identidades Administradas	Utilizadas para que AKS se autentique de forma segura en ACR sin necesidad de contraseñas.	PaaS (Identidad)
Bicep (Infraestructura como Código)	Lenguaje declarativo usado para automatizar el despliegue de toda la infraestructura.	-

Análisis de Costos del Proyecto

La gestión de costos fue un pilar fundamental del proyecto. La estrategia se centró en utilizar recursos económicos y aplicar prácticas operativas para minimizar el gasto. El costo del proyecto fue de **\$0.05**

Conclusiones Clave de Costos:

- **El Clúster de AKS es el principal motor de costos.** Al usar VMs de la serie B (B2s) y detener el clúster cuando no se usa es una medida efectiva.
- Los servicios PaaS como ACR y Log Analytics son muy rentables a pequeña escala.
- La automatización con Bicep no tiene costo directo, pero ahorra costos indirectos al evitar errores humanos y permitir despliegues y limpiezas rápidas.



Proyecto: Modernización a microservicios nativos en la nube con Microsoft Azure Kubernetes Service

Objetivo: Demostrar una migración exitosa de un monolito a una arquitectura de microservicios resiliente, escalable y observable en Azure, utilizando prácticas de Infraestructura como Código (IaC) para garantizar la repetibilidad y la gobernanza.

Fase 0: Arquitectura y planificación

1. La aplicación a modernizar (Simulada):

- Utilizaré una aplicación de ejemplo de **Aplicación de Votación** estándar que simula perfectamente un entorno de microservicios para no perder el tiempo en dependencias.
- **Microservicio 1 (Frontend):** Una aplicación web simple (Python/Flask o Node.js/React) que permite a los usuarios votar.
- **Microservicio 2 (API de Backend):** Una API REST (Python/FastAPI o Node.js/Express) que recibe los votos del frontend y los persiste.
- **Componente de datos (Base de datos):** Una base de datos simple en memoria como **Redis** para almacenar los votos. Se desplegará también como un contenedor dentro de Kubernetes para simplificar la gestión en este proyecto.

2. Arquitectura de la solución en Azure:

- **Orquestación:** **Azure Kubernetes Service (AKS)** será el centro de la operación, gestionando el ciclo de vida de nuestros contenedores.
- **Registro de imágenes:** **Azure Container Registry (ACR)** será nuestro repositorio privado y seguro para almacenar las imágenes de Docker de cada microservicio.
- **Redes:** Crearemos una **Red Virtual (VNet)** dedicada con subredes específicas para el clúster de AKS, siguiendo buenas prácticas de aislamiento.
- **Automatización:** Utilizaremos **Bicep** como único lenguaje para definir y desplegar toda la infraestructura. El objetivo es un despliegue manos libres.
- **Observabilidad:** **Azure Monitor for Containers** (habilitado a través de un Log Analytics Workspace) nos ayudará a monitorear la salud y el rendimiento.

3. Estrategia de gestión de presupuesto y tiempo:

- Control de costos:

- **SKU de VMs:** Para los nodos de AKS, utilizaremos la serie **B (B-series)**, que es la más económica y puede "acumular" CPU cuando está inactiva y usarla en ráfagas.
- **Apagado:** El clúster de AKS es el principal consumidor de crédito. **Lo detendremos al final de cada día de trabajo en el caso que se requiera** con el comando az aks stop.
- **Limpieza:** Todos los recursos se crearán en un único **grupo de recursos**. Al finalizar el proyecto, eliminaremos este grupo para detener instantáneamente todo el consumo de crédito.

Plan de Ejecución

- **Fase 1 - Preparación y containerización.**
 - **Punto 1:** Configurar el entorno de desarrollo local (Azure CLI, Docker Desktop, VS Code, kubectl).
 - **Punto 2:** Escribir los Dockerfiles para el microservicio frontend y la API de backend.
 - **Punto 3:** Construir las imágenes localmente y verificar que funcionen (docker build, docker run).
- **Fase 2 - Infraestructura como Código con Bicep.**
 - **Punto 1:** Desarrollar el archivo Bicep principal (main.bicep).
 - **Punto 2:** Crear módulos de Bicep para la red (VNet), el registro (ACR) y el clúster (AKS) para mantener el código limpio.
 - **Punto 3:** Desplegar la infraestructura en Azure usando **az deployment group create**. Depurar y validar que todos los recursos se creen correctamente.
- **Fase 3 - Despliegue de la Aplicación en AKS.**
 - **Punto 1:** Subir las imágenes de Docker a Azure Container Registry (docker push).
 - **Punto 2:** Escribir los manifiestos de Kubernetes (.yaml):
 - Deployment para el frontend, la API y Redis.
 - Service para exponer la API internamente y el frontend externamente (usando un tipo LoadBalancer).
 - **Punto 3:** Conectarse al clúster (az aks get-credentials) y desplegar la aplicación (kubectl apply).

- **Fase 4 - Validación, Monitoreo y Pruebas.**
 - **Punto 1:** Verificar que la aplicación esté funcionando accediendo a la IP pública del Service del frontend.
 - **Punto 2:** Simular una falla (ej. eliminando un pod de la API) y ver cómo Kubernetes lo recrea automáticamente, demostrando la alta disponibilidad. Se crearán alertas de que pondremos a prueba (Ej: Error HTML)
 - **Punto 3:** Demostrar la escalabilidad (Escalado horizontal)
 - **Punto 4:** Explorar Azure Monitor for Containers (Observabilidad)
- **Fase 5 - Limpieza.**
 - **Punto 1: Eliminar el grupo de Recursos**

Fase 1: Preparación del entorno (Fedora) y Containerización

Objetivo: Tener las imágenes de Docker de los microservicios construidas, probadas localmente en Fedora y listas para ser subidas a un registro. Costo de Azure en esta fase: **\$0.**

Punto 1.1: Configuración del entorno de desarrollo en Fedora

1. **Instalar Azure CLI:** Utilizaremos el gestor de paquetes dnf y el repositorio oficial de Microsoft.
 - Abre una terminal en Fedora.

Importamos la clave de Microsoft GPG:

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

Añadimos el repositorio de Azure CLI:

```
sudo dnf install -y dnf-plugins-core
```

```
sudo dnf config-manager --add-repo  
https://packages.microsoft.com/yumrepos/azure-cli
```

Instalamos Azure CLI:

```
sudo dnf install -y azure-cli
```

- **Verificación:** Cierra y vuelve a abrir tu terminal, luego ejecuta `az --version`.
- **Inicio de sesión:** Ejecuta `az login` para autenticarte en tu cuenta de Azure.

```
stevenvallejo@fedora:/$ az --version
azure-cli          2.68.0 *
core                2.68.0 *
telemetry           1.1.0
Dependencies:
msal                1.32.3
azure-mgmt-resource 23.1.1

Python location '/usr/bin/python3'
Extensions directory '/home/stevenvallejo/.azure/cliextensions'

Python (Linux) 3.13.7 (main, Aug 14 2025, 00:00:00) [GCC 15.2.1 20250808 (Red Hat 15.2.1-1)]
Legal docs and information: aka.ms/AzureCliLegal

You have 2 update(s) available. Consider updating your CLI installation with 'az upgrade'
stevenvallejo@fedora:/$ az login
/usr/lib/python3.13/site-packages/azure/cli/core/aaz/_command.py:132: FutureWarning: functools.partial will be a method descriptor in future Python versions; wrap it in staticmethod() if you want to preserve the old behavior
    if self.AZ_PREVIEW_INFO:
/usr/lib/python3.13/site-packages/azure/cli/core/aaz/_command.py:133: FutureWarning: functools.partial will be a method descriptor in future Python versions; wrap it in staticmethod() if you want to preserve the old behavior
    self.preview_info = self.AZ_PREVIEW_INFO(cli_ctx=self.cli_ctx)
/usr/lib/python3.13/site-packages/azure/cli/core/aaz/_command.py:50: FutureWarning: functools.partial will be a method descriptor in future Python versions; wrap it in staticmethod() if you want to preserve the old behavior
    self.group_kwargs['preview_info'] = self.AZ_PREVIEW_INFO(cli_ctx=self.cli_ctx)
A web browser has been opened at https://login.microsoftonline.com/organizations/oauth2/v2.0/authorize. Please continue the login in the web browser.
If no web browser is available or if the web browser fails to open, use device code flow with 'az login --use-device-code'.
```

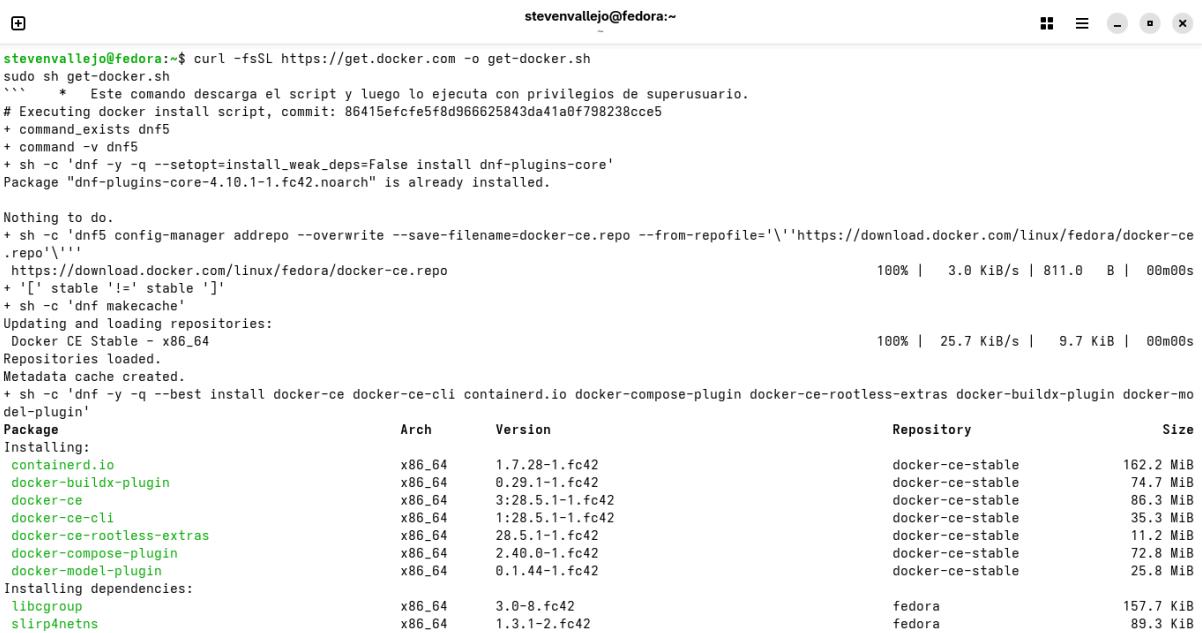
2. Limpieza inicial (Si es que se lo requiere)

```
sudo dnf remove docker docker-client docker-client-latest docker-common  
docker-latest docker-latest-logrotate docker-logrotate docker-engine
```

Instalamos Docker usando script oficial:

```
expand_less curl -fsSL https://get.docker.com -o get-docker.sh  
sudo sh get-docker.sh
```

- * Este comando descarga el script y luego lo ejecuta con privilegios de superusuario.



The screenshot shows a terminal window titled 'stevenvallejo@fedora:~'. The command entered was 'curl -fsSL https://get.docker.com -o get-docker.sh' followed by 'sudo sh get-docker.sh'. The output of the script shows the download of the Docker CE repository configuration file, the update of the package database, and the installation of various Docker components. A table at the bottom lists the installed packages along with their architectures, versions, repositories, and sizes.

Package	Arch	Version	Repository	Size
Installing:				
containerd.io	x86_64	1.7.28-1.fc42	docker-ce-stable	162.2 MiB
docker-buildx-plugin	x86_64	0.29.1-1.fc42	docker-ce-stable	74.7 MiB
docker-ce	x86_64	3:28.5.1-1.fc42	docker-ce-stable	86.3 MiB
docker-ce-cli	x86_64	1:28.5.1-1.fc42	docker-ce-stable	35.3 MiB
docker-ce-rootless-extras	x86_64	28.5.1-1.fc42	docker-ce-stable	11.2 MiB
docker-compose-plugin	x86_64	2.40.0-1.fc42	docker-ce-stable	72.8 MiB
docker-model-plugin	x86_64	0.1.44-1.fc42	docker-ce-stable	25.8 MiB
Installing dependencies:				
libcgroup	x86_64	3.0-8.fc42	fedora	157.7 KiB
slirp4netns	x86_64	1.3.1-2.fc42	fedora	89.3 KiB

3. Iniciar, habilitar y configurar Docker:

Iniciamos el servicio de Docker:

```
sudo systemctl start docker
```

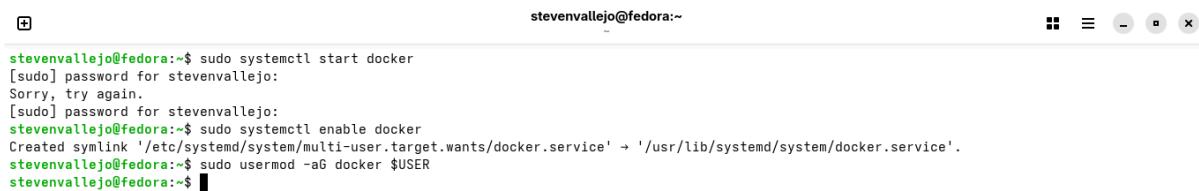
Habilitamos para que inicie con el sistema:

```
sudo systemctl enable docker
```

Añadimos un usuario al grupo docker: Esto te permitirá ejecutar comandos de Docker sin sudo.

```
sudo usermod -aG docker $USER
```

Este cambio requiere que **cierres tu sesión actual y vuelvas a iniciarla** para que tu membresía al grupo se actualice.



The screenshot shows a terminal window titled 'stevenvallejo@fedora:~'. It displays the following command history:

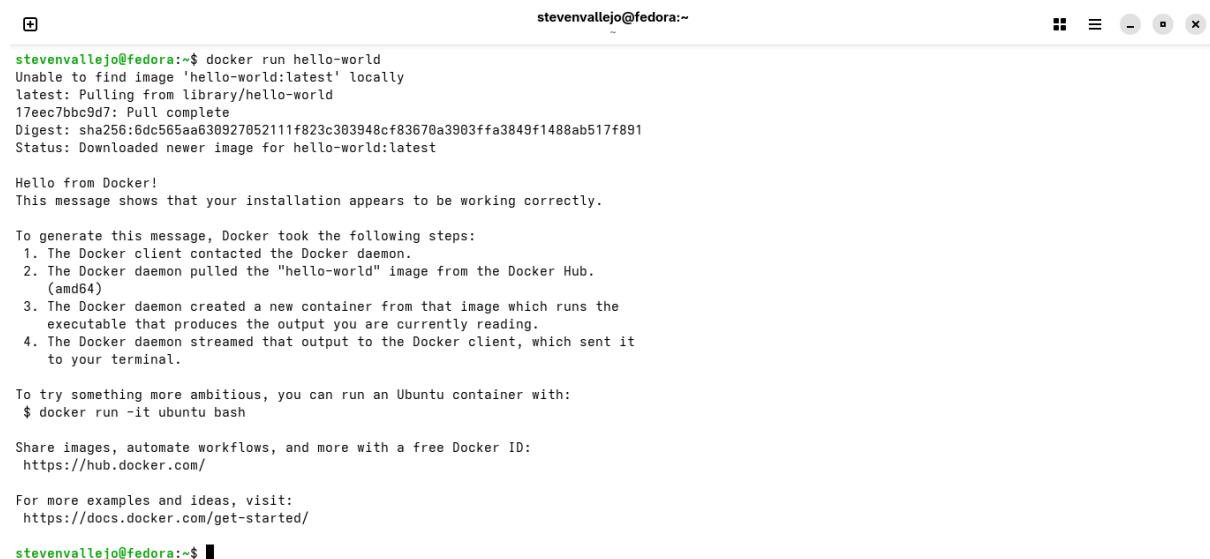
```
stevenvallejo@fedora:~$ sudo systemctl start docker
[sudo] password for stevenvallejo:
Sorry, try again.
[sudo] password for stevenvallejo:
stevenvallejo@fedora:~$ sudo systemctl enable docker
Created symlink '/etc/systemd/system/multi-user.target.wants/docker.service' → '/usr/lib/systemd/system/docker.service'.
stevenvallejo@fedora:~$ sudo usermod -aG docker $USER
stevenvallejo@fedora:~$
```

Verificación:

Después de volver a iniciar sesión, abre una nueva terminal.

Ejecuta el siguiente comando **sin sudo** para comprobar los cambios:

```
docker run hello-world
```



The screenshot shows a terminal window with the following content:

```
stevenvallejo@fedora:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
17ec7bbc9d7: Pull complete
Digest: sha256:6dc565aa63092705211f823c303948cf83670a3903ffa3849f1488ab517f891
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
 executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
 to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
stevenvallejo@fedora:~$
```

4. Instalar Visual Studio Code: Importar la clave y añadir el repositorio de Microsoft:

```
sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

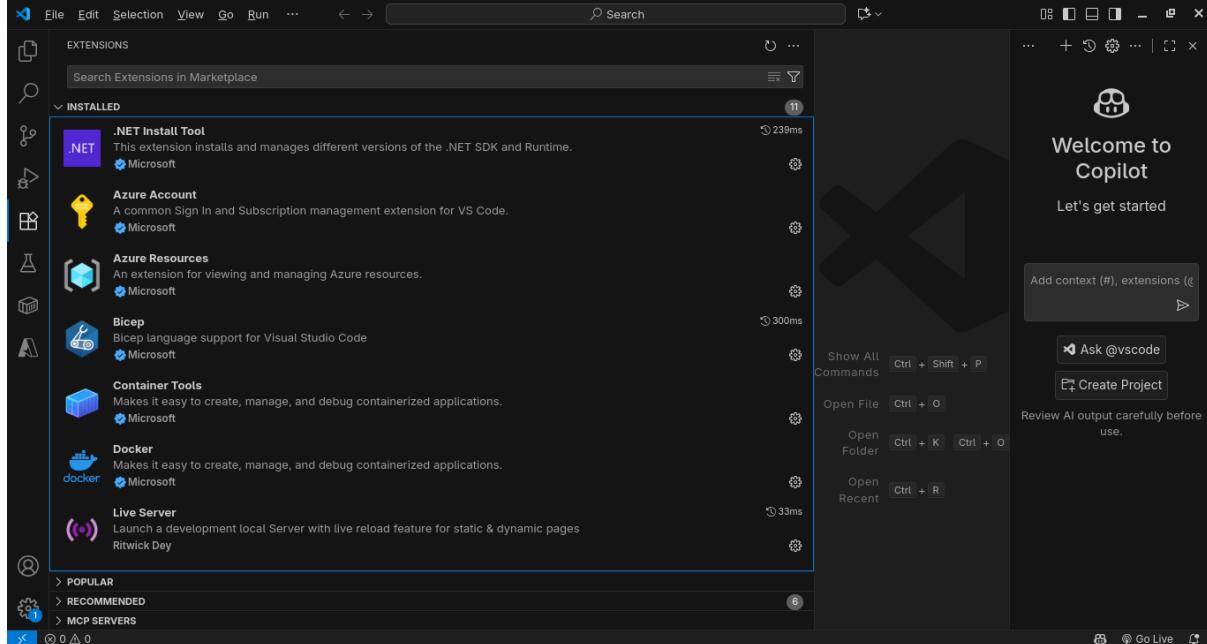
```
printf "[vscode]\nname=Visual Studio\nCode\nbaseurl=https://packages.microsoft.com/yumrepos/vscode\nenabled=1\nngpgch\neck=1\nngpgkey=https://packages.microsoft.com/keys/microsoft.asc" | sudo tee\n/etc/yum.repos.d/vscode.repo > /dev/null
```

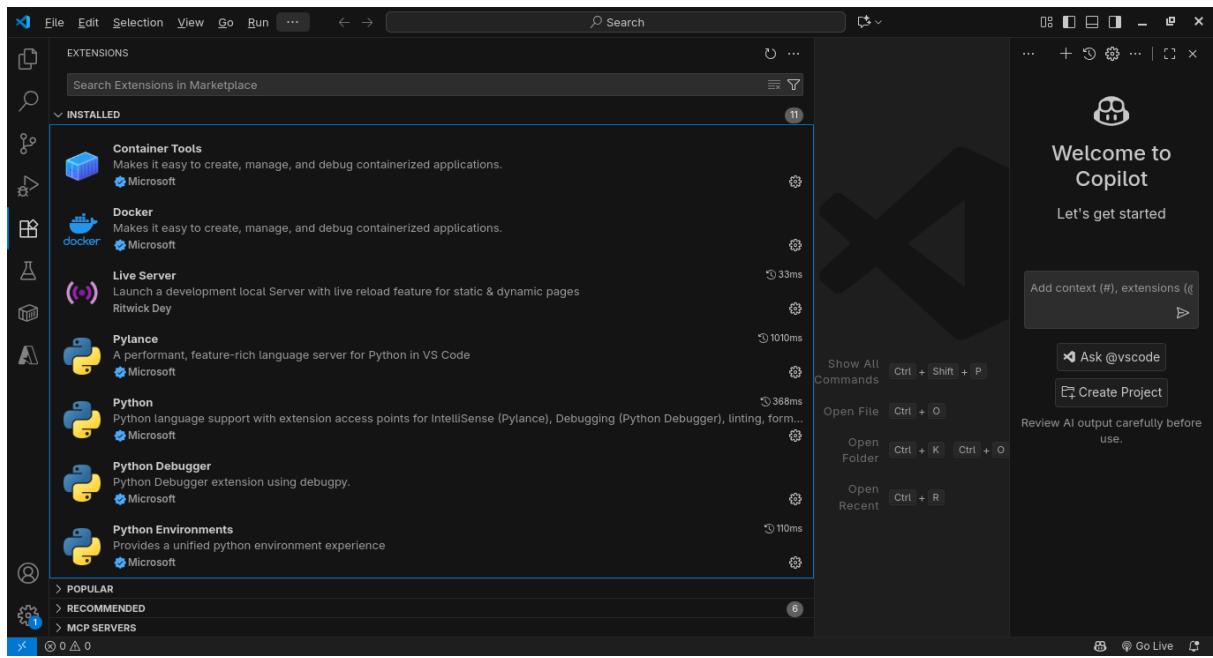
Instalamos VS Code:

```
sudo dnf check-update
```

```
sudo dnf install -y code
```

Dentro de VS Code, instalamos las extensiones **Docker, Azure Account, Bicep, y Python**





5. Instalar kubectl: Usaremos Azure CLI, ya que es el método más simple y es independiente del sistema operativo.

```
sudo az aks install-cli
```

Verificación: Ejecuta `kubectl version --client`.



```
stevenvallejo@fedora:~$ sudo az aks install-cli
[sudo] password for stevenvallejo:
/usr/lib/python3.13/site-packages/azure/cli/core/aaz/_command.py:132: FutureWarning: functools.partial will be a method descriptor in future Python versions; wrap it in staticmethod() if you want to preserve the old behavior
  if self._PREVIEW_INFO:
/usr/lib/python3.13/site-packages/azure/cli/core/aaz/_command.py:133: FutureWarning: functools.partial will be a method descriptor in future Python versions; wrap it in staticmethod() if you want to preserve the old behavior
  self.preview_info = self._PREVIEW_INFO(cli_ctx=self.cli_ctx)
/usr/lib/python3.13/site-packages/azure/cli/core/aaz/_command.py:50: FutureWarning: functools.partial will be a method descriptor in future Python versions; wrap it in staticmethod() if you want to preserve the old behavior
  if self._PREVIEW_INFO:
/usr/lib/python3.13/site-packages/azure/cli/core/aaz/_command.py:51: FutureWarning: functools.partial will be a method descriptor in future Python versions; wrap it in staticmethod() if you want to preserve the old behavior
  self.group_kwargs['preview_info'] = self._PREVIEW_INFO(cli_ctx=self.cli_ctx)
The detected architecture of current device is "x86_64", and the binary for "amd64" will be downloaded. If the detection is wrong, please download and install the binary corresponding to the appropriate architecture.
No version specified, will get the latest version of kubectl from "https://dl.k8s.io/release/stable.txt"
Downloading client to "/usr/local/bin/kubectl" from "https://dl.k8s.io/release/v1.34.1/bin/linux/amd64/kubectl"
Please ensure that /usr/local/bin is in your search PATH, so the "kubectl" command can be found.
No version specified, will get the latest version of kubelogin from "https://api.github.com/repos/Azure/kubelogin/releases/latest"
Downloading client to "/tmp/tmps42n6c8h/kubelogin.zip" from "https://github.com/Azure/kubelogin/releases/download/v0.2.10/kubelogin.zip"
Moving binary to "/usr/local/bin/kubelogin" from "/tmp/tmps42n6c8h/bin/linux_amd64/kubelogin"
Please ensure that /usr/local/bin is in your search PATH, so the "kubelogin" command can be found.
stevenvallejo@fedora:~$ kubectl version --client
Client Version: v1.34.1
Kustomize Version: v5.7.1
stevenvallejo@fedora:~$
```

Ahora **Fedora** tiene **az**, **docker**, **code** y **kubectl** para trabajar y nuestro comando docker se ejecuta sin sudo.

Punto 1.2: Crear y "Containerizar" los microservicios

Objetivo: Crear la estructura de carpetas y todos los archivos de código fuente y de configuración de Docker para nuestros dos microservicios: backend y frontend.

Estructura del proyecto:

```
aks-modernization/
```

```
    └── frontend/
```

```
        |   └── app.py
```

```
        |   └── requirements.txt
```

```
        |   └── templates/
```

```
            |       └── index.html
```

```
        |   └── Dockerfile
```

```
    └── backend/
```

```
        └── app.py
```

```
        └── requirements.txt
```

```
        └── Dockerfile
```

Crear la Estructura de directorios

Primero, abre tu terminal. Asegúrate de estar en tu directorio home o donde quieras crear el proyecto.

1. Creamos la carpeta principal del proyecto:

```
mkdir aks-modernization
```

```
cd aks-modernization
```

2. Creamos las carpetas para cada microservicio y sus plantillas:

```
mkdir backend
```

```
mkdir frontend
```

```
mkdir frontend/templates
```

3. Verifica la estructura: Ejecuta el comando ls -R. Te saldrá esto:



```
stevenvallejo@fedora:~/aks-modernization
stevenvallejo@fedora:~/aks-modernization$ ls -R
.:
backend  frontend

./backend:
./frontend:
./frontend/templates:
stevenvallejo@fedora:~/aks-modernization$
```

Crear los archivos del microservicio backend

Navega a la carpeta del backend: cd backend.

1. Archivo de dependencias (requirements.txt):

- Crea el archivo. Usamos el comando touch requirements.txt y luego abrirlo en VS Code, o directamente crear el archivo desde VS Code.

Pega lo siguiente:

```
# backend/requirements.txt
fastapi
uvicorn[standard]
redis
```

- **fastapi**: El framework para construir nuestra API.
- **uvicorn**: El servidor que ejecutará nuestra aplicación FastAPI.
- **redis**: La librería de Python para comunicarnos con la base de datos Redis.

2. Archivo de la Aplicación ([app.py](#)):

- Crea el archivo con touch [app.py](#).

Pega el siguiente código:

```
# backend/app.py
import os
from fastapi import FastAPI
from redis import Redis
app = FastAPI()
```

```
# Conexión a Redis. El nombre 'redis-service' será el que usemos en
Kubernetes.
```

```
# Para pruebas locales, usaremos 'localhost' por defecto.
```

```
redis_host = os.environ.get('REDIS_HOST', 'localhost')
```

```
redis_conn = Redis(host=redis_host, port=6379, db=0,  
decode_responses=True)
```

```
@app.get("/")
```

```
def read_root():
```

```
    return {"message": "Backend API is running"}
```

```
@app.post("/vote/{option}")
```

```
def cast_vote(option: str):
```

```
    # Incrementa el contador para la opción votada ('dogs' o 'cats')
```

```
    count = redis_conn.incr(option)
```

```
    return {"option": option, "count": count}
```

```
@app.get("/results")
```

```
def get_results():
```

```
    dogs_count = redis_conn.get('dogs') or 0
```

```
    cats_count = redis_conn.get('cats') or 0
```

```
    return {"dogs": dogs_count, "cats": cats_count}
```

3. El Plano del Contenedor (Dockerfile):

Crea un archivo llamado Dockerfile (sin extensión).

Pega el siguiente contenido:

```
# Build
```

```
FROM python:3.9-slim as builder
```

```
# Previene que Python escriba archivos .pyc y almacene en búfer la salida
```

```
ENV PYTHONDONTWRITEBYTECODE 1
```

```
ENV PYTHONUNBUFFERED 1
```

```
WORKDIR /app
```

```
COPY requirements.txt .
```

```
# Crear un entorno virtual, activarlo y usarlo para instalar dependencias
```

```
RUN python -m venv /opt/venv
```

```
ENV PATH="/opt/venv/bin:$PATH"
```

```
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Final
```

```
FROM python:3.9-slim
```

```
WORKDIR /app
```

```
# Copia el entorno virtual completo de la fase de construcción
```

```
COPY --from=builder /opt/venv /opt/venv
```

```
# Copia el código fuente de la aplicación
```

```
COPY ..
```

```
# Activa el entorno virtual para que el CMD pueda encontrar unicorn
```

```
ENV PATH="/opt/venv/bin:$PATH"
```

```
EXPOSE 80
```

```
CMD ["unicorn", "app:app", "--host", "0.0.0.0", "--port", "80"]
```

Se **crearon** los **3 archivos** necesarios para el **servicio de API**. La carpeta backend está completa.

Crear los archivos del microservicio frontend

Regresamos a la carpeta principal (cd ..) y luego entra en la del frontend (cd frontend).

1. Archivo de dependencias (requirements.txt):

- Crea el archivo requirements.txt.

Pega el siguiente texto:

```
# frontend/requirements.txt
flask
requests
```

- **flask**: Un framework web simple para servir nuestra página HTML.
- **requests**: La librería que usaremos para hacer llamadas a nuestra API de backend.

2. Plantilla HTML (templates/index.html):

- Navega a la carpeta templates.
- Crea el archivo index.html.

código HTML:

```
<!-- frontend/templates/index.html -->

<!DOCTYPE html>

<html>

<head>

<title>Azure Voting App</title>

<style>

body { font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto,
Helvetica, Arial, sans-serif; text-align: center; background-color: #f0f8ff; margin-top: 50px; }
```

```
.container { max-width: 600px; margin: auto; padding: 20px; background: white; border-radius: 8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); }

button { background-color: #0078d4; color: white; border: none; padding: 15px 30px; margin: 10px; border-radius: 5px; cursor: pointer; font-size: 16px; transition: background-color 0.2s; }

button:hover { background-color: #005a9e; }

h2 { color: #333; }

#results { font-size: 18px; color: #555; margin-top: 20px; }

</style>

</head>

<body>

<div class="container">

<h2>¿Qué prefieres?</h2>

<form action="/vote" method="post">

<button name="vote" value="cats">Gatos</button>

<button name="vote" value="dogs">Perros</button>

</form>

<div id="results">

<p>Gatos: <b>{{ results.cats }}</b> | Perros: <b>{{ results.dogs }}</b></p>

</div>

</div>

</body>

</html>
```

3. Archivo de la aplicación (app.py):

- Regresa a la carpeta frontend (cd ..).
- Crea el archivo [app.py](#).

Código:

```
# frontend/app.py

import os
import requests
from flask import Flask, render_template, request, redirect, url_for

app = Flask(__name__)

# La URL del backend será inyectada como variable de entorno en Kubernetes.

# Para pruebas locales, apuntará al puerto del backend que expondremos.

backend_url = os.environ.get('BACKEND_API_URL', 'http://localhost:8000')

@app.route("/")
def index():

    try:

        # Obtiene los resultados actuales de la API de backend

        response = requests.get(f"{backend_url}/results")

        results = response.json()

    except requests.exceptions.ConnectionError:

        print(f"ERROR: No se pudo conectar a la API de backend en {backend_url}")

        results = {"dogs": "N/A", "cats": "N/A"}
```

```
return render_template('index.html', results=results)

@app.route("/vote", methods=['POST'])

def vote():

    option = request.form['vote']

    try:

        # Envia el voto a la API de backend

        requests.post(f'{backend_url}/vote/{option}')

    except requests.exceptions.ConnectionError:

        print(f"ERROR: No se pudo conectar a la API de backend en {backend_url}")


```

```
return redirect(url_for('index'))

if __name__ == '__main__':

    # Puerto 8080.

    app.run(host='0.0.0.0', port=8080, debug=True)
```

4. El Plano del Contenedor (Dockerfile):

- Crea el archivo Dockerfile en frontend/.

Contenido:

```
# Build

FROM python:3.9-slim as builder

# Previene que Python escriba archivos .pyc y almacene en búfer la salida
```

```
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

WORKDIR /app
COPY requirements.txt .

# Crea un entorno virtual, activarlo y usarlo para instalar dependencias
RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"
RUN pip install --no-cache-dir -r requirements.txt

# Final
FROM python:3.9-slim

WORKDIR /app

# Copia el entorno virtual completo de la fase de construcción
COPY --from=builder /opt/venv /opt/venv

# Copia el código fuente de la aplicación
COPY ..
```

```
# Activar el entorno virtual
```

```
ENV PATH="/opt/venv/bin:$PATH"
```

```
EXPOSE 8080
```

```
CMD ["python", "app.py"]
```



The screenshot shows a terminal window with the following details:

- Title Bar:** stevenvallejo@fedora:~/aks-modernization
- Path:** -/aks-modernization
- Content:** The terminal displays a file tree structure:

```
stevenvallejo@fedora:~/aks-modernization/backend$ 
stevenvallejo@fedora:~/aks-modernization/backend$ cd ..
stevenvallejo@fedora:~/aks-modernization$ cd frontend/
stevenvallejo@fedora:~/aks-modernization/frontend$ touch requirements.txt
stevenvallejo@fedora:~/aks-modernization/frontend$ nano requirements.txt
stevenvallejo@fedora:~/aks-modernization/frontend$ nano requirements.txt
stevenvallejo@fedora:~/aks-modernization/frontend$ cd ..
stevenvallejo@fedora:~/aks-modernization$ cd frontend/
stevenvallejo@fedora:~/aks-modernization/frontend$ cd templates/
stevenvallejo@fedora:~/aks-modernization/frontend/templates$ touch index.html
stevenvallejo@fedora:~/aks-modernization/frontend/templates$ nano index.html
stevenvallejo@fedora:~/aks-modernization/frontend/templates$ nano index.html
stevenvallejo@fedora:~/aks-modernization/frontend/templates$ cd ..
stevenvallejo@fedora:~/aks-modernization/frontend$ touch app.py
stevenvallejo@fedora:~/aks-modernization/frontend$ nano app.py
stevenvallejo@fedora:~/aks-modernization/frontend$ nano app.py
stevenvallejo@fedora:~/aks-modernization/frontend$ nano app.py
stevenvallejo@fedora:~/aks-modernization/frontend$ touch Dockerfile
stevenvallejo@fedora:~/aks-modernization/frontend$ nano Dockerfile
stevenvallejo@fedora:~/aks-modernization/frontend$ cd ..
stevenvallejo@fedora:~/aks-modernization$ ls -R
.:
backend  frontend

./backend:
app.py  Dockerfile  requirements.txt

./frontend:
app.py  Dockerfile  requirements.txt  templates

./frontend/templates:
index.html
stevenvallejo@fedora:~/aks-modernization$
```

Punto 1.3: Construir y probar las imágenes localmente

Objetivo: Compilar los Dockerfiles en imágenes de contenedor funcionales y ejecutarlas juntas en tu máquina Fedora para verificar que la aplicación completa funciona antes de tocar Azure.

Construir la imagen del backend

1. Abre tu terminal y ve al directorio aks-modernization.

Navega a la carpeta del backend:

```
cd backend/
```

2. Ejecuta el comando de construcción de Docker:

```
docker build -t voting-app-backend .
```

- docker build: Comando para construir una imagen.
- -t voting-app-backend: La opción -t (tag) le da un nombre legible (voting-app-backend) a la imagen.
- .: Le dice a Docker que use el Dockerfile que se encuentra en el directorio actual.

3. Esto puede tardar un minuto la primera vez, ya que descargó la imagen base de Python.

Construir la imagen del frontend

1. Regresa a la carpeta raíz del proyecto y navega a la del frontend:

```
cd ..\frontend\
```

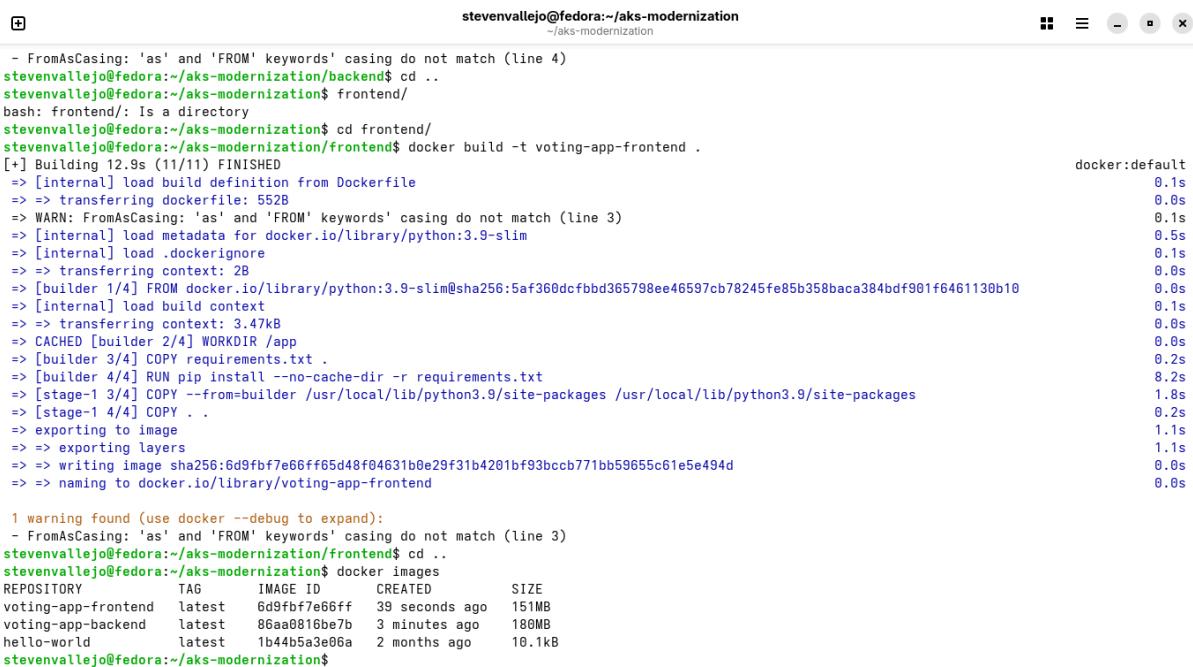
2. Ejecuta el comando de construcción para el frontend:

```
docker build -t voting-app-frontend .
```

Verificar que las imágenes se crearon

Ejecuta el siguiente comando para listar todas las imágenes de Docker en la terminal:

```
docker images
```



```
stevenvallejo@fedora:~/aks-modernization
~/aks-modernization$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
docker              default  6d9fbf7e66ff  39 seconds ago  151MB
voting-app-frontend latest   6d9fbf7e66ff  39 seconds ago  151MB
voting-app-backend  latest   86aa0816be7b  3 minutes ago   180MB
hello-world         latest   1b44b5a3e06a  2 months ago   10.1kB
```

Deberías ver voting-app-frontend y voting-app-backend en la parte superior de la lista.

Si ves las **dos imágenes** en la lista, significa que la construcción se realizó con éxito

Ejecutar la aplicación completa localmente

Lanzaremos los tres contenedores para que trabajen juntos: la base de datos Redis, el backend y frontend.

1. Iniciar el contenedor de redis:

Usaremos una imagen oficial de Redis de Docker Hub. Docker la descargará automáticamente.

```
docker run --name redis -p 6379:6379 -d redis:alpine
```

- --name redis: Nombra al contenedor "redis" para identificarlo fácilmente.
- -p 6379:6379: Mapea el puerto 6379 de la computadora al puerto 6379 dentro del contenedor.
- -d: Modo "detached", para que se ejecute en segundo plano.
- redis:alpine: La imagen de Redis, versión alpine muy ligera.

2. Iniciar el contenedor del backend:

```
docker run --name backend-container -p 8000:80 -d voting-app-backend
```

- --name backend-container: Nombra al contenedor.
- -p 8000:80: Mapea el puerto 8000 de la computadora al puerto 80 del contenedor donde está escuchando unicorn. El código de frontend buscará la API en localhost:8000.

3. Iniciar el Contenedor del Frontend:

```
docker run --name frontend-container -p 8080:8080 -d voting-app-frontend
```

- --name frontend-container: Nombra al contenedor.
- -p 8080:8080: Mapea el puerto 8080 de la computadora al puerto 8080 del contenedor donde escucha Flask. Este es el puerto que abriremos en el navegador.

4. Verificar que los contenedores están corriendo:

```
docker ps
```

Deberías ver una lista con los tres contenedores (redis, backend-container, frontend-container) en estado "Up".

```
stevenvallejo@fedora:~/aks-modernization
~/aks-modernization

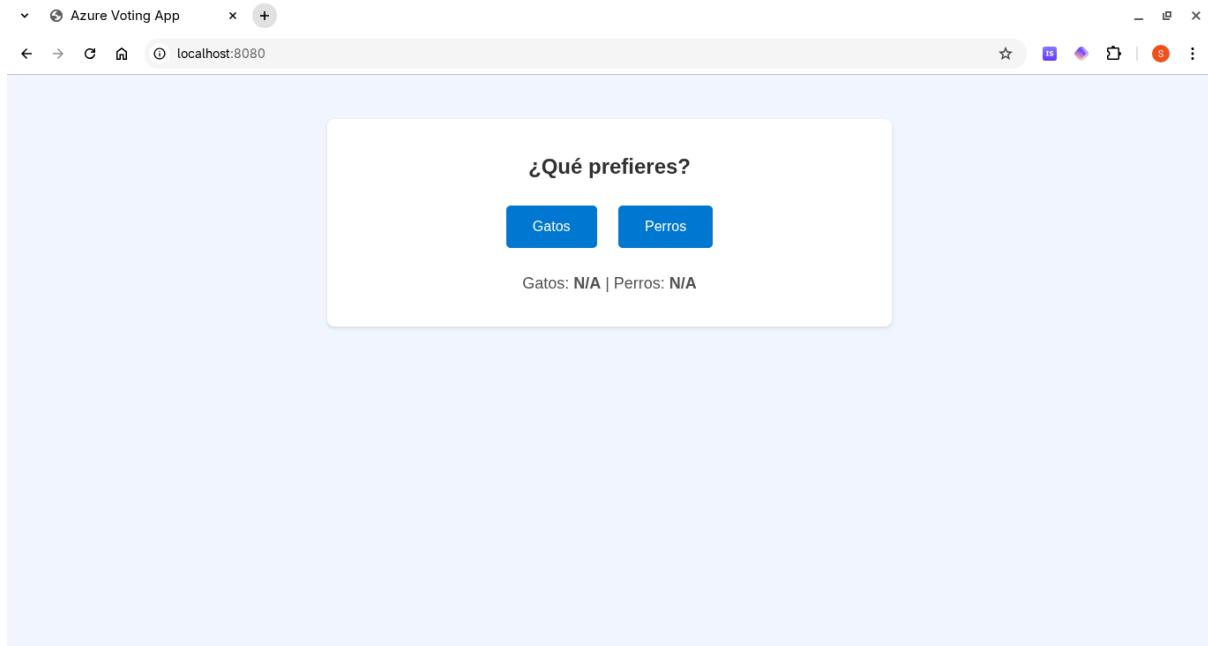
=> [internal] load build context
=> => transferring context: 1.16kB
=> [builder 1/5] FROM docker.io/library/python:3.9-slim@sha256:5af360dcfbbd365798ee46597cb78245fe85b358baca384bdf901f6461130b10
=> CACHED [builder 2/5] WORKDIR /app
=> CACHED [builder 3/5] COPY requirements.txt .
=> [builder 4/5] RUN python -m venv /opt/venv
=> [builder 5/5] RUN pip install --no-cache-dir -r requirements.txt
=> [stage-1 3/4] COPY --from=builder /opt/venv /opt/venv
=> [stage-1 4/4] COPY . .
=> exporting to image
=> => exporting layers
=> => writing image sha256:e3ff08fdf3a091ab510a7b6d4b11b89506ae91ca467bb117c59dcb7462ffa12e
=> => naming to docker.io/library/voting-app-frontend
=> => naming to docker.io/library/voting-app-backend
=> => naming to docker.io/library/redis

3 warnings found (use docker --debug to expand):
- LegacyKeyValueFormat: "ENV key=value" should be used instead of legacy "ENV key value" format (line 6)
- FromAsCasing: 'as' and 'FROM' keywords casing do not match (line 2)
- LegacyKeyValueFormat: "ENV key=value" should be used instead of legacy "ENV key value" format (line 5)

stevenvallejo@fedora:~/aks-modernization/frontend$ cd ..
stevenvallejo@fedora:~/aks-modernization$ docker run --name redis -p 6379:6379 -d redis:alpine
857bda560382718484a93e4fc22f551b499a6acf0fc7d79cc9d16b1962de965b
stevenvallejo@fedora:~/aks-modernization$ docker run --name backend-container -p 8000:80 -d voting-app-backend
3b20c4b2a53d770ea5182793c90c5c5823f11d94eba39c4b70049e3f6a0cc27
stevenvallejo@fedora:~/aks-modernization$ docker run --name frontend-container -p 8080:8080 -d voting-app-frontend
24e78ec13a5216ea2c881adb299fcc35a19b684b1ff3860409d543b89baac67
stevenvallejo@fedora:~/aks-modernization$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
24e78ec13a52 voting-app-frontend "python app.py" 7 seconds ago Up 6 seconds 0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp frontend-container
3b20c4b2a53d voting-app-backend "uvicorn app:app --h..." 14 seconds ago Up 13 seconds 0.0.0.0:8000->80/tcp, [::]:8000->80/tcp backend-container
857bda560382 redis:alpine "docker-entrypoint.s..." 24 seconds ago Up 24 seconds 0.0.0.0:6379->6379/tcp, [::]:6379->6379/tcp redis
stevenvallejo@fedora:~/aks-modernization$
```

Verificación

1. Abre tu navegador web.
2. Ve a la dirección: <http://localhost:8080>



Limpieza del entorno

Dejaremos nuestro espacio de trabajo limpio.

Detenemos los contenedores con:

```
docker stop frontend-container backend-container redis
```

Eliminar los contenedores con:

```
docker rm frontend-container backend-container redis
```

```
stevenvallejo@fedora:~/aks-modernization$ docker stop frontend-container backend-container redis
frontend-container
backend-container
redis
stevenvallejo@fedora:~/aks-modernization$ docker rm frontend-container backend-container redis
frontend-container
backend-container
redis
stevenvallejo@fedora:~/aks-modernization$ docker ps
CONTAINER ID   IMAGE      COMMAND   CREATED    STATUS     PORTS      NAMES
stevenvallejo@fedora:~/aks-modernization$
```

Fase 2: Infraestructura como código (IaC) con Bicep

Objetivo: Definir y desplegar toda la infraestructura de Azure (Red, Registro de Contenedores, Clúster de Kubernetes) utilizando Bicep. Al finalizar esta fase, tendremos un clúster de AKS vacío, monitoreado y listo para recibir nuestra aplicación. **Advertencia:** A partir de este momento, iniciamos el consumo del crédito de Azure.

Punto 2.1: Definir la arquitectura y estructura del proyecto

1. **Convenciones de nomenclatura:** Usaremos un nombre base para agrupar visualmente todos nuestros recursos en el portal de Azure.
 - **Nombre base:** votingapp
 - **Grupo de recursos:** votingapp-rg
 - **Red virtual:** votingapp-vnet
 - **Registro de contenedores (ACR):** votingappacr<stevenvallejo10190>. **Importante:** El nombre de ACR debe ser único a nivel global en todo Azure y solo puede contener letras minúsculas y números.
 - **Clúster de Kubernetes (AKS):** votingapp-aks
2. **Estructura de Archivos Bicep:** La modularidad es clave para la reutilización y la legibilidad. Dentro de tu carpeta aks-modernization, crea la siguiente estructura:
 - Primero, crea la carpeta infra, asegúrate de estar en la raíz del proyecto: aks-modernization/

```
mkdir infra
```
 - Ahora, crea la carpeta de módulos y los archivos vacíos:

```
cd infra
```

```
mkdir modules
```

```
touch main.bicep main.parameters.json modules/acr.bicep modules/aks.bicep  
modules/vnet.bicep
```

Verifica la estructura: Ejecuta ls -R

- **main.bicep**: Será el orquestador que llamará a nuestros módulos.
- **main.parameters.json**: Separará la configuración (nombres, ubicaciones) del código, permitiendo desplegar el mismo código en diferentes entornos (ej. dev, prod) solo cambiando este archivo.
- **modules/**: Contendrá nuestros ladrillos de Bicep, cada uno responsable de un único recurso.



```
stevenvallejo@fedora:~/aks-modernization/infra  
stevenvallejo@fedora:~/aks-modernization$ mkdir infra  
stevenvallejo@fedora:~/aks-modernization$ cd infra/  
stevenvallejo@fedora:~/aks-modernization/infra$ mkdir modules  
stevenvallejo@fedora:~/aks-modernization/infra$ touch main.bicep main.parameters.json modules/acr.bicep modules/aks.bicep modules/vnet.bicep  
stevenvallejo@fedora:~/aks-modernization/infra$ ls -R  
.:  
main.bicep main.parameters.json modules  
  
./modules:  
acr.bicep aks.bicep vnet.bicep  
stevenvallejo@fedora:~/aks-modernization/infra$
```

Punto 2.2: Escribir el código de la infraestructura (Bicep)

Abre la carpeta infra en VS Code para una mejor edición.

1. El archivo de parámetros (main.parameters.json)

Este archivo define el "qué" y el "dónde".

Pega el siguiente contenido en infra/main.parameters.json:

```
{
```

```
"$schema":  
"https://schema.management.azure.com/schemas/2019-04-01/deploymentParameters.json#",  
  
"contentVersion": "1.0.0.0",  
  
"parameters": {  
  
    "resourceNamePrefix": {  
  
        "value": "votingapp"  
  
    },  
  
    "location": {  
  
        "value": "eastus"  
  
    },  
  
    "acrName": {  
  
        "value": "votingappacrsv2510"  
  
    }  
}  
}
```

- Cambia votingappacr<REEMPLAZAR> por tu nombre único.

2. El Módulo de Red (modules/vnet.bicep)

Creamos una red aislada para nuestro clúster.

Contenido en infra/modules/vnet.bicep:

```
// Define el nombre de la red virtual que se creará

param vnetName string

// Define la ubicación (región de Azure) donde se creará la red

param location string

// Define el nombre de la subred para los nodos de AKS

param aksSubnetName string = 'aks-subnet'

// Creación del recurso de la Red Virtual (VNet)

resource vnet 'Microsoft.Network/virtualNetworks@2023-05-01' = {

    name: vnetName

    location: location

    properties: {

        addressSpace: {

            addressPrefixes: [
                '10.0.0.0/16' // Rango de IPs principal para toda la red
            ]
        }
    }

    subnets: [
        {
            name: aksSubnetName
        }
    ]
}
```

```

    properties: {

        addressPrefix: '10.0.0.0/24' // Rango de IPs específico para
los nodos de AKS

    }

}

]

}

}

// 'output' expone información del recurso creado para que otros
módulos puedan usarla.

// Exponemos el ID de la subred, que el clúster de AKS necesitará.

output aksSubnetId string = vnet.properties.subnets[0].id

```

3. El módulo de registro de contenedores (modules/acr.bicep)

El almacén privado para nuestras imágenes de Docker.

Contenido en infra/modules/acr.bicep:

```

// Define el nombre único global para el Azure Container Registry (ACR)

param acrName string

// Define la ubicación (debe ser la misma que la del resto de recursos)

param location string

// Creación del recurso ACR

```

```
resource acr 'Microsoft.ContainerRegistry/registries@2023-07-01' = {
    name: acrName
    location: location
    // SKU 'Basic' es el más económico y suficiente para este proyecto
    sku: {
        name: 'Basic'
    }
    properties: {
        // Habilitamos el usuario administrador. Facilita el inicio de
        // sesión desde la CLI local,
        adminUserEnabled: true
    }
}

// Exponemos el ID del ACR. El clúster de AKS lo necesita para obtener
// permisos de extracción.

output acrId string = acr.id
```

4. El Módulo del Clúster de Kubernetes (modules/aks.bicep)

El cerebro de nuestra operación. Este es el módulo más complejo.

Contenido en infra/modules/aks.bicep:

```
// Define el nombre para el clúster de AKS

param clusterName string


// Define la ubicación

param location string


// Recibe el ID de la subred que creamos en el módulo vnet.bicep

param aksSubnetId string


// CORRECCIÓN: Nuevo parámetro para recibir el ID del Log Analytics
// Workspace

param logAnalyticsWorkspaceId string


// Creación del recurso Clúster de Kubernetes (AKS)

resource aksCluster
'Microsoft.ContainerService/managedClusters@2023-09-01' = {

    name: clusterName

    location: location

    identity: {

        type: 'SystemAssigned'

    }

    properties: {

        dnsPrefix: clusterName
```

```
agentPoolProfiles: [
  {
    name: 'agentpool',
    count: 1,
    vmSize: 'Standard_B2s',
    osType: 'Linux',
    mode: 'System',
    vnetSubnetID: aksSubnetId
  }
]

networkProfile: {
  networkPlugin: 'azure',
  serviceCidr: '10.1.0.0/16',
  dnsServiceIP: '10.1.0.10'
}

// Habilitamos el monitoreo con Azure Monitor for Containers

addonProfiles: {
  omsagent: {
    enabled: true,
    config: {
      // CORRECCIÓN: Usamos el ID del workspace que nos pasan como
      // parámetro
      logAnalyticsWorkspaceResourceID: logAnalyticsWorkspaceId
    }
  }
}
```

```
    }

}

}

// Exponemos el Principal ID de la identidad del clúster.

// Lo necesitamos para asignarle roles.

output aksPrincipalId string = aksCluster.identity.principalId
```

5. Crear el módulo loganalytics.bicep

Este archivo es el almacén de logs

Contenido en infra/loganalytics.bicep:

```
// Define el nombre para el Log Analytics Workspace

param workspaceName string


// Define la ubicación

param location string


// Creación del recurso Log Analytics Workspace

resource logAnalytics
'Microsoft.OperationalInsights/workspaces@2022-10-01' = {

    name: workspaceName

    location: location

    properties: {

        sku: {
```

```
        name: 'PerGB2018' // SKU estándar y de pago por uso

    }

    retentionInDays: 30 // Retenemos los logs por 30 días (el mínimo y
más barato)

}

}

// Exponemos el ID del recurso para que otros módulos lo puedan usar
output workspaceId string = logAnalytics.id
```

6. El orquestador principal (main.bicep)

Este archivo une todo.

Contenido en infra/main.bicep:

```
// PARÁMETROS: Entradas para nuestro despliegue.

param resourceNamePrefix string

param location string

param acrName string

// VARIABLES: Nombres construidos que usaremos en los recursos.

var vnetName = '${resourceNamePrefix}-vnet'
```

```
var aksClusterName = '${resourceNamePrefix}-aks'

var logAnalyticsWorkspaceName = '${resourceNamePrefix}-logs'

// MÓDULOS: Despliegue de los componentes de la infraestructura.

module virtualNetwork 'modules/vnet.bicep' = {

    name: 'virtualNetworkDeployment'

    params: {

        vnetName: vnetName

        location: location

    }

}

module containerRegistry 'modules/acr.bicep' = {

    name: 'containerRegistryDeployment'

    params: {

        acrName: acrName

        location: location

    }

}

module logAnalytics 'modules/loganalytics.bicep' = {

    name: 'logAnalyticsDeployment'

    params: {
```

```
workspaceName: logAnalyticsWorkspaceName

location: location

}

}

module kubernetesCluster 'modules/aks.bicep' = {

name: 'kubernetesClusterDeployment'

params: {

clusterName: aksClusterName

location: location

aksSubnetId: virtualNetwork.outputs.aksSubnetId

logAnalyticsWorkspaceId: logAnalytics.outputs.workspaceId

}

}

// RECURSO: Obtenemos una referencia al ACR para usarlo en el 'scope'.

resource acr 'Microsoft.ContainerRegistry/registries@2023-07-01'
existing = {

name: acrName

}

// ASIGNACIÓN DE ROL: Conexión final entre AKS y ACR.
```

```
resource assignAcrPullRole
'Microsoft.Authorization/roleAssignments@2022-04-01' = {
  name: guid(resourceGroup().id, acrName, aksClusterName, 'AcrPull')
  scope: acr
  properties: {
    // Usando el ID del rol 'AcrPull' de la suscripción.
    roleDefinitionId:
    subscriptionResourceId('Microsoft.Authorization/roleDefinitions',
    '7f951dda-4ed3-4680-a7ca-43fe172d538d')
    principalId: kubernetesCluster.outputs.aksPrincipalId
    principalType: 'ServicePrincipal'
  }
}
```

Se ha escrito todo el código IaC. **Revisa** que cada archivo esté en su lugar y tenga el contenido correcto, especialmente el nombre único de tu ACR en main.parameters.json.

Quedaría así:

The screenshot shows a code editor interface with the following details:

- Title Bar:** aks-modernization
- Explorer:** AKS-MODERNIZATION folder structure:
 - backend
 - frontend
 - infra
 - modules
 - acr.bicep
 - aks.bicep
 - loganalytics.bicep
 - vnet.bicep
 - main.bicep
 - main.parameters.json
- Code Editor:** The main.bicep file is open, showing Bicep code for deploying infrastructure. The code includes parameters for resourceNamePrefix, location, and acrName, and defines modules for virtual network and container registry deployment.
- Status Bar:** Ln 68, Col 2 | Spaces: 2 | UTF-8 | LF | {} Bicep | Go Live

Punto 2.3: Desplegar la infraestructura en Azure

Objetivo: Ejecutar un único comando que leerá nuestros archivos Bicep y construirá toda la arquitectura definida en nuestra suscripción de Azure.

Paso 1: Abrir la terminal en la ubicación correcta

1. Abre la terminal de Fedora.

Asegúrate de que estás en el directorio infra de tu proyecto. La ruta debería ser similar a ~/aks-modernization/infra. Si no estás ahí, navega a esa carpeta:

```
cd ~/aks-modernization/infra
```

Paso 2: Autenticarse y establecer la suscripción (Si es que es necesario)

Confirma que has iniciado sesión en Azure:

```
az account show
```

1. Si no te muestra la suscripción correcta, ejecuta az login de nuevo.
2. Si tienes varias suscripciones, asegúrate de que la que tiene el crédito gratuito está activa. Puedes listarlas con az account list y establecer la correcta con az account set --subscription "NOMBRE_O_ID_DE_LA_SUSCRIPCION".

Paso 3: Ejecutar el comando de despliegue

1. Crear el Grupo de Recursos

Este comando creará el "contenedor" vacío en la región correcta. Debería completarse en pocos segundos.

```
az group create --name 'votingapp-rg' --location 'eastus'
```

Verás una salida JSON con "provisioningState": "Succeeded" confirmando que el grupo se ha creado.

```
stevenvallejo@fedora:~/aks-modernization/infra — /usr/bin/python3 -sP /usr/bin/az deployment group create --name aks-infra-deployment-01 --reso...
stevenvallejo@fedora:~/aks-modernization/infra$ az group create --name 'votingapp-rg' --location 'eastus'
{
  "id": "/subscriptions/4f7efdc6-c5ab-4b53-9b54-58b1358993f7/resourceGroups/votingapp-rg",
  "location": "eastus",
  "managedBy": null,
  "name": "votingapp-rg",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

Importante: Registrar los proveedores de Azure

1. Abre tu terminal.

Ejecuta el siguiente comando para registrar el proveedor de servicios de Kubernetes. **Este comando puede tardar varios minutos (2 a 5 min) en completarse.** La terminal parecerá que no hace nada, pero está trabajando.

```
az provider register --namespace Microsoft.ContainerService
```

2. Ahora, registra el proveedor para Log Analytics (Azure Monitor). Este también puede tardar unos minutos.

```
az provider register --namespace Microsoft.OperationsManagement
```

3. **Verifica el estado del registro.** Puedes comprobar si ya han terminado con los siguientes comandos. Sigue ejecutándolos cada minuto hasta que el "registrationState" cambie de "Registering" a "Registered".

```
az provider show -n Microsoft.ContainerService -o table
```

```
az provider show -n Microsoft.OperationsManagement -o table
```

4. **No continúes** hasta que ambos comandos muestren "Registered" en la columna State.

```
stevenvallejo@fedora:~/aks-modernization/infra$ az provider register --namespace Microsoft.ContainerService
stevenvallejo@fedora:~/aks-modernization/infra$ az provider register --namespace Microsoft.OperationsManagement
Registering is still on-going. You can monitor using 'az provider show -n Microsoft.OperationsManagement'
stevenvallejo@fedora:~/aks-modernization/infra$ az provider show -n Microsoft.ContainerService -o table
az provider show -n Microsoft.OperationsManagement -o table
Namespace          RegistrationPolicy   RegistrationState
Microsoft.ContainerService  RegistrationRequired  Registered
Namespace          RegistrationPolicy   RegistrationState
Microsoft.OperationsManagement RegistrationRequired  Registering
stevenvallejo@fedora:~/aks-modernization/infra$ az provider show -n Microsoft.ContainerService -o table
Namespace          RegistrationPolicy   RegistrationState
Microsoft.ContainerService  RegistrationRequired  Registered
stevenvallejo@fedora:~/aks-modernization/infra$ az provider show -n Microsoft.OperationsManagement -o table
Namespace          RegistrationPolicy   RegistrationState
Microsoft.OperationsManagement RegistrationRequired  Registered
stevenvallejo@fedora:~/aks-modernization/infra$
```

2. Desplegar la infraestructura dentro del grupo

Ahora que el grupo de recursos existe, ejecutamos nuestro comando de despliegue

```
az deployment group create \
--name 'aks-infra-deployment-01' \
--resource-group 'votingapp-rg' \
--template-file 'main.bicep' \
--parameters 'main.parameters.json'
```

Empezará el proceso de 10-15 minutos para crear los recursos.

Paso 4: Observar y esperar

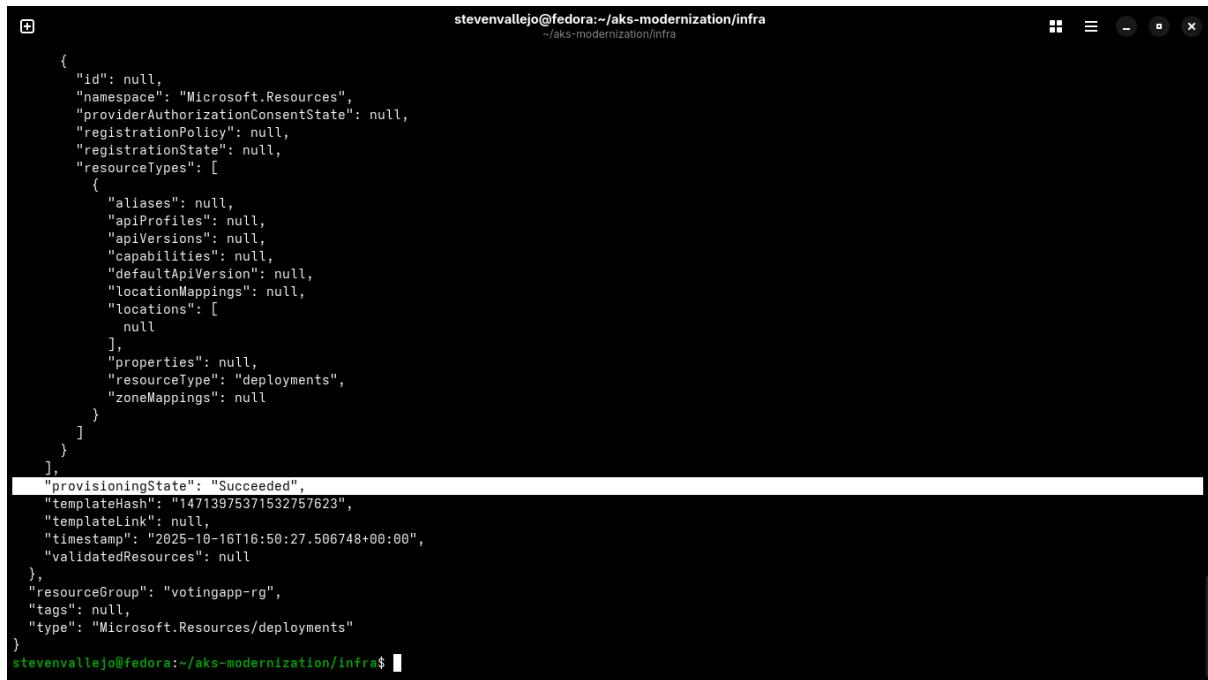
1. **No interrumpas el proceso.** La terminal mostrará un mensaje - Running
2. **Tiempo de ejecución:** Este proceso **tardará entre 10 y 15 minutos.**
3. **Opcional:**
 - Abre tu navegador y ve a <https://portal.azure.com>.
 - En la barra de búsqueda, escribe "Grupos de recursos".
 - Verás aparecer un nuevo grupo de recursos llamado votingapp-rg.
 - Si haces clic en él, verás cómo los recursos (VNet, ACR, AKS, Log Analytics) van apareciendo uno por uno a medida que el despliegue avanza.

Name	Subscription	Location
MC_votingapp-rg_votingapp-aks_eastus	Azure subscription 1	East US
NetworkWatcherRG	Azure subscription 1	East US
votingapp-rg	Azure subscription 1	East US

El **despliegue** será **exitoso** cuando el comando en tu terminal termine y muestre una gran salida de texto en formato JSON. Lo más importante que debes buscar al final de esa salida es esta línea:

```
"provisioningState": "Succeeded",
```

Si es así, ha sido **exitoso**.



```
stevenvallejo@fedora:~/aks-modernization/infra
~/aks-modernization/infra
{
  "id": null,
  "namespace": "Microsoft.Resources",
  "providerAuthorizationConsentState": null,
  "registrationPolicy": null,
  "registrationState": null,
  "resourceTypes": [
    {
      "aliases": null,
      "apiProfiles": null,
      "apiVersions": null,
      "capabilities": null,
      "defaultApiVersion": null,
      "locationMappings": null,
      "locations": [
        null
      ],
      "properties": null,
      "resourceType": "deployments",
      "zoneMappings": null
    }
  ],
  "provisioningState": "Succeeded",
  "templateHash": "14713975371532757623",
  "templateLink": null,
  "timestamp": "2025-10-16T16:50:27.506748+00:00",
  "validatedResources": null
},
"resourceGroup": "votingapp-rg",
"tags": null,
"type": "Microsoft.Resources/deployments"
}
stevenvallejo@fedora:~/aks-modernization/infra$
```

Resumen:

En el grupo de recursos votingapp-rg ahora mismo existen:

1. Una **Red Virtual** (votingapp-vnet) que aísla el clúster.
2. Un **Log Analytics Workspace** (votingapp-logs) para recolectar toda la telemetría.
3. Un **Azure Container Registry** (votingappacrsv2510) que servirá como almacén privado de imágenes de Docker.
4. Un **Clúster de Kubernetes** (votingapp-aks) con un nodo, conectado a la red, configurado para enviar logs.
5. **La conexión:** El permiso (roleAssignment) que permite al clúster AKS extraer imágenes del ACR de forma segura.

Fase 3: Despliegue de la aplicación en AKS

Objetivo: Publicar nuestras imágenes de contenedor en nuestro registro privado en la nube (ACR) y luego instruir a Kubernetes para que las descargue, las ejecute y las exponga a Internet.

Punto 3.1: Publicar las imágenes de Docker en Azure Container Registry (ACR)

Nuestras imágenes voting-app-frontend y voting-app-backend solo existen en nuestra máquina Fedora. Necesitamos subirlas a nuestro ACR para que el clúster de AKS pueda acceder a ellas.

Paso 1: Iniciar sesión en nuestro ACR desde la terminal

Primero, debemos autenticar nuestro Docker local con nuestro registro en la nube.

1. Abre tu terminal.

Necesitamos el nombre de nuestro ACR. **Reemplaza <TU_ACR_UNICO> por el nombre que le diste** (ej. votingappacrv2510). O usa este comando para obtenerlo automáticamente:

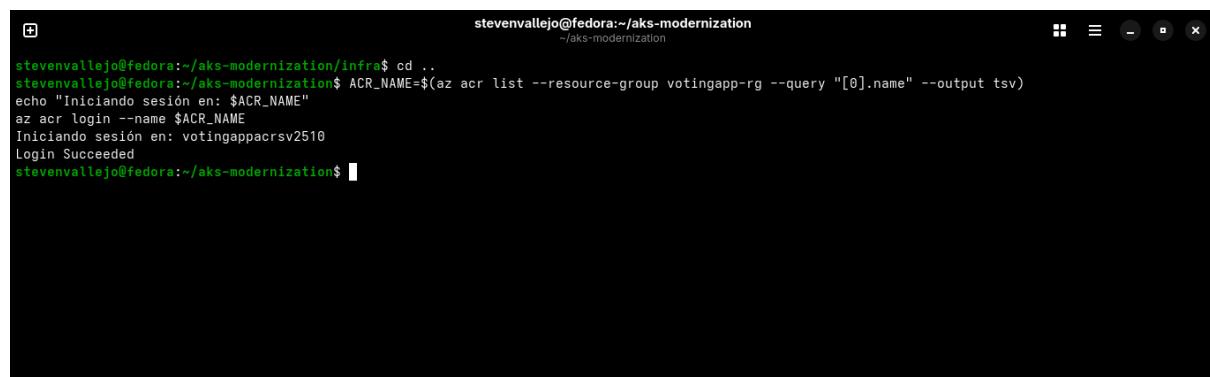
```
ACR_NAME=$(az acr list --resource-group votingapp-rg --query "[0].name" --output tsv)
```

```
echo "Iniciando sesión en: $ACR_NAME"
```

```
az acr login --name $ACR_NAME
```

2. El comando az acr login debería devolver Login Succeeded. Esto configura tu Docker local para que pueda "empujar" (push) imágenes a tu registro.

Paso 2: "Etiquetar" (Tag) las imágenes para el registro



```
stevenvallejo@fedora:~/aks-modernization$ cd ..
stevenvallejo@fedora:~/aks-modernization$ ACR_NAME=$(az acr list --resource-group votingapp-rg --query "[0].name" --output tsv)
echo "Iniciando sesión en: $ACR_NAME"
az acr login --name $ACR_NAME
Iniciando sesión en: votingappacrv2510
Login Succeeded
stevenvallejo@fedora:~/aks-modernization$
```

Docker necesita saber a qué registro debe subir cada imagen. Para ello, creamos una nueva "etiqueta" (un alias) para nuestras imágenes locales con el formato:
<nombre_del_acr>.azurecr.io/<nombre_de_la_imagen>:<version>.

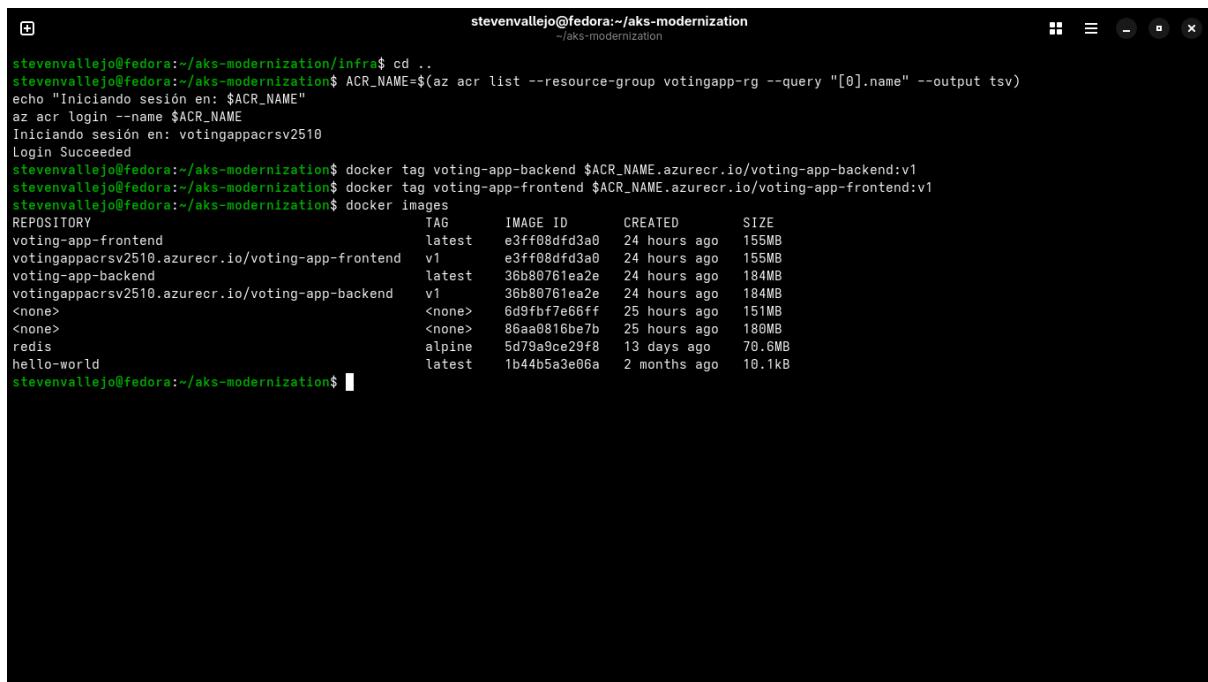
1. Etiquetar la imagen del backend:

```
docker tag voting-app-backend $ACR_NAME.azurecr.io/voting-app-backend:v1
```

2. Etiquetar la imagen del frontend:

```
docker tag voting-app-frontend $ACR_NAME.azurecr.io/voting-app-frontend:v1
```

3. Verificar las nuevas etiquetas: Ejecuta **docker images**. Ahora verás las imágenes originales y sus nuevos alias, que apuntan al mismo IMAGE ID.



The screenshot shows a terminal window with the following command history and output:

```
stevenvallejo@fedora:~/aks-modernization
~/aks-modernization
stevenvallejo@fedora:~/aks-modernization$ cd ..
stevenvallejo@fedora:~/aks-modernization$ ACR_NAME=$(az acr list --resource-group votingapp-rg --query "[0].name" --output tsv)
echo "Iniciando sesión en: $ACR_NAME"
az acr login --name $ACR_NAME
Iniciando sesión en: votingappacrv2510
Login Succeeded
stevenvallejo@fedora:~/aks-modernization$ docker tag voting-app-backend $ACR_NAME.azurecr.io/voting-app-backend:v1
stevenvallejo@fedora:~/aks-modernization$ docker tag voting-app-frontend $ACR_NAME.azurecr.io/voting-app-frontend:v1
stevenvallejo@fedora:~/aks-modernization$ docker images
REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
voting-app-frontend    latest   e3ff08dfd3a0  24 hours ago  155MB
votingappacrv2510.azurecr.io/voting-app-frontend    v1      e3ff08dfd3a0  24 hours ago  155MB
voting-app-backend    latest   36b80761ea2e  24 hours ago  184MB
votingappacrv2510.azurecr.io/voting-app-backend    v1      36b80761ea2e  24 hours ago  184MB
<none>              <none>   6d9fbf7e66ff  25 hours ago  151MB
<none>              <none>   86aa0816be7b  25 hours ago  180MB
redis                alpine   5d79a9cce29f8  13 days ago   70.6MB
hello-world          latest   1b44b5a3e06a  2 months ago  10.1kB
stevenvallejo@fedora:~/aks-modernization$
```

Paso 3: Empujar (Push) las imágenes a la nube

Es el equivalente a subir un archivo, pero para imágenes de Docker.

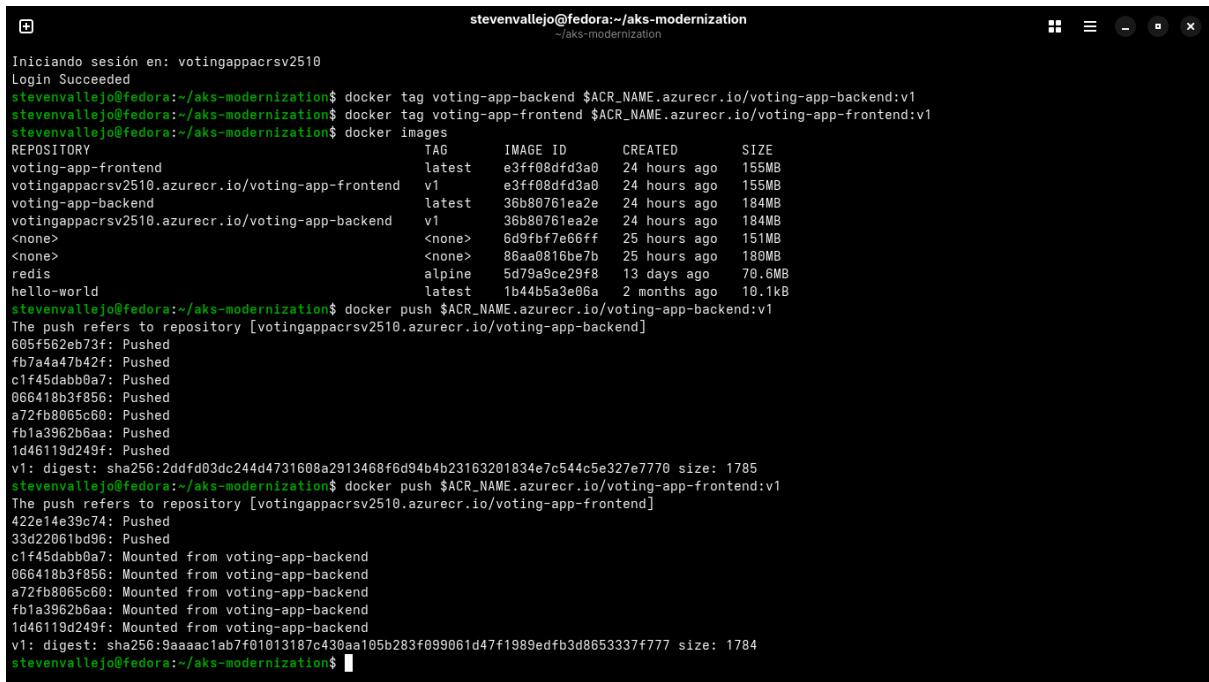
1. Subir la imagen del backend:

```
docker push $ACR_NAME.azurecr.io/voting-app-backend:v1
```

Verás una barra de progreso mientras las capas de la imagen se suben a tu ACR.

2. Subir la imagen del frontend:

```
docker push $ACR_NAME.azurecr.io/voting-app-frontend:v1
```



The screenshot shows a terminal window titled "stevenvallejo@fedora:~/aks-modernization". The session is starting with "Iniciando sesión en: votingappacrv2510" and "Login Succeeded". The user runs "docker tag voting-app-backend \$ACR_NAME.azurecr.io/voting-app-backend:v1" and "docker tag voting-app-frontend \$ACR_NAME.azurecr.io/voting-app-frontend:v1". Then, they run "docker images" to list the local images. Finally, they run "docker push \$ACR_NAME.azurecr.io/voting-app-backend:v1" and "docker push \$ACR_NAME.azurecr.io/voting-app-frontend:v1". The terminal shows the progress of the pushes, including the size of each layer (e.g., 1785, 1784, 1783, 1782, 1781, 1780, 1779, 1778, 1777) and the total size of the images (1785 and 1784).

```
Iniciando sesión en: votingappacrv2510
Login Succeeded
stevenvallejo@fedora:~/aks-modernization$ docker tag voting-app-backend $ACR_NAME.azurecr.io/voting-app-backend:v1
stevenvallejo@fedora:~/aks-modernization$ docker tag voting-app-frontend $ACR_NAME.azurecr.io/voting-app-frontend:v1
stevenvallejo@fedora:~/aks-modernization$ docker images
REPOSITORY          TAG      IMAGE ID   CREATED     SIZE
voting-app-frontend    latest   e3ff08dfd3a0  24 hours ago  155MB
votingappacrv2510.azurecr.io/voting-app-frontend   v1      e3ff08dfd3a0  24 hours ago  155MB
voting-app-backend    latest   36b80761ea2e  24 hours ago  184MB
votingappacrv2510.azurecr.io/voting-app-backend   v1      36b80761ea2e  24 hours ago  184MB
<none>              <none>  6d9fbf7e66ff  25 hours ago  151MB
<none>              <none>  86aa0816be7b  25 hours ago  180MB
redis                alpine   5d79a9cce29f8  13 days ago   70.6MB
hello-world          latest   1b44b5a3e06a  2 months ago  10.1kB
stevenvallejo@fedora:~/aks-modernization$ docker push $ACR_NAME.azurecr.io/voting-app-backend:v1
The push refers to repository [votingappacrv2510.azurecr.io/voting-app-backend]
605f562eb73f: Pushed
fb7a4a47b42f: Pushed
c1f45dabbba7: Pushed
066418b3f856: Pushed
a72fb8065c60: Pushed
fb1a3962b6aa: Pushed
1d46119d249f: Pushed
v1: digest: sha256:2ddfd03dc244d4731608a2913468f6d94b4b23163201834e7c544c5e327e7770 size: 1785
stevenvallejo@fedora:~/aks-modernization$ docker push $ACR_NAME.azurecr.io/voting-app-frontend:v1
The push refers to repository [votingappacrv2510.azurecr.io/voting-app-frontend]
422e14e39c74: Pushed
33d22061bd96: Pushed
c1f45dabbba7: Mounted from voting-app-backend
066418b3f856: Mounted from voting-app-backend
a72fb8065c60: Mounted from voting-app-backend
fb1a3962b6aa: Mounted from voting-app-backend
1d46119d249f: Mounted from voting-app-backend
v1: digest: sha256:9aaaac1ab7f01013187c430aa105b283f099061d47f1989edfb3d8653337f777 size: 1784
stevenvallejo@fedora:~/aks-modernization$
```

Para **verificar** que las **imágenes** se han subido correctamente, ve al **Portal de Azure**, busca tu **ACR**, ir a la sección "**Repositorios**" y verás voting-app-backend y voting-app-frontend listados.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the Azure logo, a search bar, and various icons. The main title is "votingappacrv2510 | Repositories". On the left, there's a sidebar with options like "Quick start", "Resource visualizer", "Events", "Settings", "Services", and "Repositories" (which is currently selected). The main content area displays a table with two rows:

Repositories ↑↓	Cache Rule
voting-app-backend	...
voting-app-frontend	...

Below the table, there's a note: "Add or remove favorites by pressing Ctrl+Shift+F".

Punto 3.2: Escribir los manifiestos de Kubernetes

Objetivo: Crear los archivos de configuración (.yaml) que le dicen a Kubernetes qué contenedores ejecutar, cómo conectarlos entre sí y cómo exponer la aplicación al mundo exterior.

Paso 1: Crear la carpeta para los manifiestos

Para mantener nuestro proyecto organizado, crearemos una nueva carpeta para estos archivos.

1. En tu terminal, asegúrate de estar en la raíz del proyecto (~/aks-modernization).

```
mkdir manifests
```

```
cd manifests
```

Paso 2: Crear el manifiesto de despliegue (deployment.yaml)

Este archivo contendrá las instrucciones para los tres componentes de nuestra aplicación: **Redis, Backend y Frontend.**

1. Dentro de la carpeta manifests, crea un archivo llamado deployment.yaml.
Contenido completo en el archivo.

```
# manifests/deployment.yaml

# Definición del Deployment para REDIS
# Un Deployment se encarga de mantener un número deseado de
# copias (pods) de nuestra aplicación funcionando.

apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis-deployment # Nombre del deployment
spec:
  replicas: 1 # Una sola copia de Redis
  selector:
    matchLabels:
```

```
1    app: redis # Este selector conecta el Deployment con los
2    Pods que gestiona
3
4    template:
5
6      metadata:
7
8        labels:
9
10       app: redis # Los Pods creados tendrán esta etiqueta
11
12   spec:
13
14     containers:
15
16       - name: redis
17
18         image: redis:alpine # La imagen de Docker a usar (de
19         Docker Hub)
20
21     ports:
22
23       - containerPort: 6379 # El puerto que expone el contenedor
24       de Redis
25
26
27   ---
28
29
30
31   # Definición del Service para REDIS
32
33   # Un Service proporciona una dirección de red estable (un nombre
34   DNS interno) para acceder a los Pods.
35
36   apiVersion: v1
37
38   kind: Service
39
40   metadata:
41
42     name: redis-service # Nombre del service. El backend usará este
43     nombre para conectarse.
44
45   spec:
46
47     ports:
48
49       - port: 6379 # El puerto que expone el Service
50
51     selector:
52
53       app: redis # Redirige el tráfico a cualquier Pod con la
54       etiqueta app=redis
55
56
57   ---
58
59
60   # Deployment para el BACKEND
61
62   apiVersion: apps/v1
63
64   kind: Deployment
```

```
1 metadata:
2   name: backend-deployment
3
4 spec:
5   replicas: 1 # Empezamos con una copia
6   selector:
7     matchLabels:
8       app: backend
9
10 template:
11   metadata:
12     labels:
13       app: backend
14
15 spec:
16   containers:
17     - name: backend
18       image: votingappacrsrv2510.azurecr.io/voting-app-backend:v1
19       ports:
20         - containerPort: 80
21       env:
22         - name: REDIS_HOST
23           value: "redis-service" # Le decimos al backend que Redis
24           se encuentra en 'redis-service'
25
26 ---
27
28 # Definición del Service para el BACKEND
29 apiVersion: v1
30 kind: Service
31 metadata:
32   name: backend-service
33
34 spec:
35   ports:
36     - port: 80
37   selector:
38     app: backend
39
40 ---
41
```

```
01
02 # Deployment para el FRONTEND
03 apiVersion: apps/v1
04 kind: Deployment
05 metadata:
06   name: frontend-deployment
07 spec:
08   replicas: 1
09   selector:
10     matchLabels:
11       app: frontend
12   template:
13     metadata:
14       labels:
15         app: frontend
16   spec:
17     containers:
18       - name: frontend
19         image:
20           votingappacrv2510.azurecr.io/voting-app-frontend:v1
21         ports:
22           - containerPort: 8080
23         env:
24           - name: BACKEND_API_URL
25             value: "http://backend-service" # Le decimos al
26             frontend que la API está en 'http://backend-service'
27
28   ---
29
30 # Service para el FRONTEND
31 # Se expone nuestra aplicación a Internet.
32 apiVersion: v1
33 kind: Service
34 metadata:
35   name: frontend-service
36 spec:
```

```
    # 'LoadBalancer' es el tipo de Service que automáticamente
    crea un balanceador de carga
14.   # en Azure con una dirección IP pública para recibir tráfico
     externo.
15.   type: LoadBalancer
16.   ports:
17.     - port: 80 # El Load Balancer recibirá tráfico en el puerto 80
       (HTTP estándar)
18.     targetPort: 8080 # Y lo redirigirá al puerto 8080 de
       nuestros contenedores frontend
19.   selector:
20.     app: frontend # Redirige el tráfico a cualquier Pod con la
       etiqueta app=frontend
21.
```

Se ha creado un **archivo deployment.yaml** completo que **describe** toda tu **aplicación** en el lenguaje de Kubernetes y lo has **personalizado** con la ubicación de tus imágenes en tu **ACR**.

Punto 3.3: Conectarse al clúster y desplegar la aplicación

Objetivo: Establecer una conexión segura desde la terminal de Fedora a tu clúster de AKS en Azure y aplicar nuestro manifiesto para que Kubernetes construya y lance la aplicación.

Paso 1: Obtener las credenciales de acceso al clúster

Necesitamos descargar un archivo de configuración especial (kubeconfig) que contiene las credenciales y la dirección del clúster.

1. Abre tu terminal.

Ejecuta el siguiente comando. Reemplaza votingapp-rg y votingapp-aks si usaste nombres diferentes.

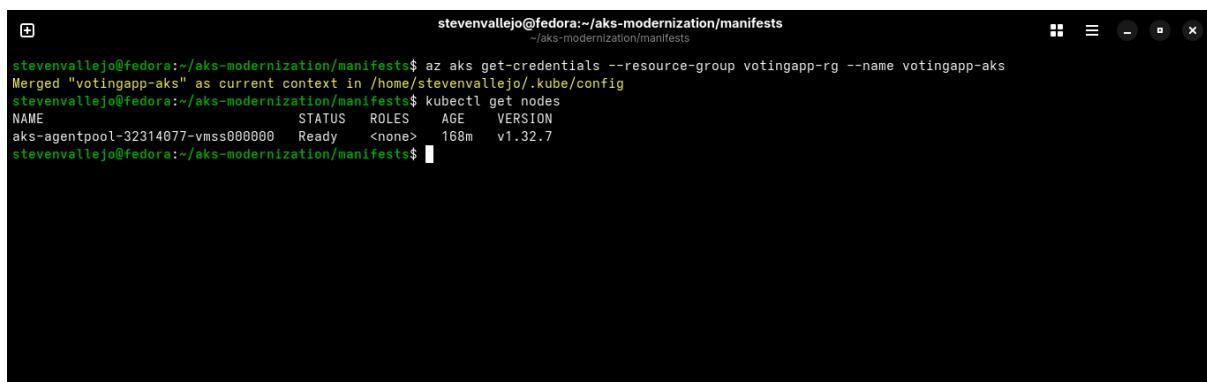
```
az aks get-credentials --resource-group votingapp-rg --name votingapp-aks
```

2. Este comando contactará a Azure, descargará las credenciales y las fusionará automáticamente con tu configuración de kubectl local en el archivo `~/.kube/config`. Deberías ver un mensaje como: Merged "votingapp-aks" as current context in `/home/stevenvallejo/.kube/config`.

Para verificar que kubectl ahora puede comunicarse con tu clúster, ejecuta este comando:

```
kubectl get nodes
```

Deberías ver **un nodo** en la lista, con el estado Ready.



```
stevenvallejo@fedora:~/aks-modernization/manifests$ az aks get-credentials --resource-group votingapp-rg --name votingapp-aks
Merged "votingapp-aks" as current context in /home/stevenvallejo/.kube/config
stevenvallejo@fedora:~/aks-modernization/manifests$ kubectl get nodes
NAME           STATUS   ROLES   AGE    VERSION
aks-agentpool-32314077-vmss000000  Ready    <none>  168m  v1.32.7
stevenvallejo@fedora:~/aks-modernization/manifests$
```

Paso 2: Desplegar la aplicación usando el manifiesto

Ahora que tenemos una conexión, podemos enviarle instrucciones.

1. Navega a la carpeta donde guardaste tu archivo de manifiesto:

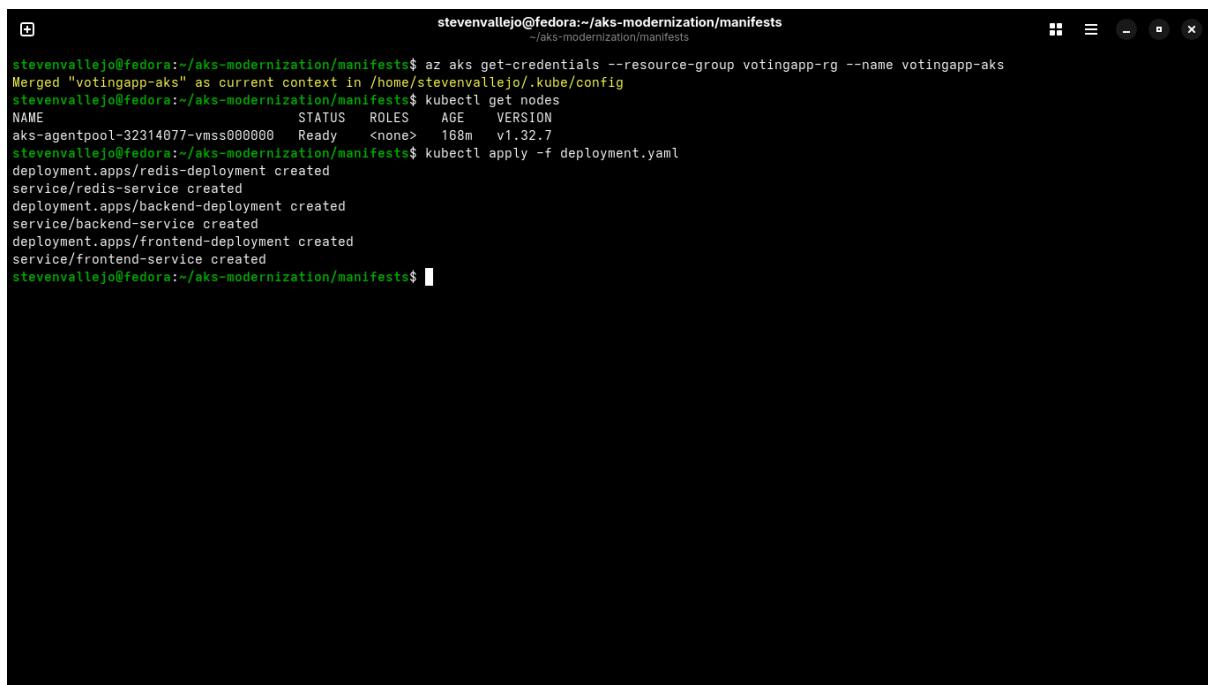
```
cd ~/aks-modernization/manifests
```

2. Ejecuta el comando apply de kubectl. Este comando lee el archivo y le dice a Kubernetes que coincide con lo que está en el manifiesto.

```
kubectl apply -f deployment.yaml
```

kubectl te **confirmará** que has **creado** todos los **recursos** definidos en el archivo.

Saldrá así:



The screenshot shows a terminal window with the following content:

```
stevenvallejo@fedora:~/aks-modernization/manifests$ az aks get-credentials --resource-group votingapp-rg --name votingapp-aks
Merged "votingapp-aks" as current context in /home/stevenvallejo/.kube/config
stevenvallejo@fedora:~/aks-modernization/manifests$ kubectl get nodes
NAME           STATUS   ROLES   AGE     VERSION
aks-agentpool-32314077-vmss000000  Ready    <none>   168m   v1.32.7
stevenvallejo@fedora:~/aks-modernization/manifests$ kubectl apply -f deployment.yaml
deployment.apps/redis-deployment created
service/redis-service created
deployment.apps/backend-deployment created
service/backend-service created
deployment.apps/frontend-deployment created
service/frontend-service created
stevenvallejo@fedora:~/aks-modernization/manifests$
```

Paso 3: Verificar el despliegue

Le hemos dado las órdenes a Kubernetes. Ahora, está trabajando para hacerlas realidad. Kubernetes tiene que:

1. Programar los "Pods" (contenedores) para que se ejecuten en nuestro nodo.
2. El nodo tiene que descargar las imágenes de tu ACR (esto puede tardar 1-2 minutos).
3. Iniciar los contenedores.
4. Azure tiene que aprovisionar un balanceador de carga y asignarle una IP pública (esto puede tardar otros 2-3 minutos).

Usa los siguientes comandos para monitorizar el proceso..

1. Ver el estado de los Pods:

```
kubectl get pods
```

Al principio, puede que veas el estado ContainerCreating. Después de un minuto o dos, todos deberían pasar al estado Running.

Si backend y frontend no están corriendo, se debe a que Azure intentó descargarlos pero falló o no tiene permiso y los puso en espera. Para solucionar esto ejecutaremos el siguiente código:

Primero, obtenemos el nombre del ACR para no tener que escribirlo a mano

```
ACR_NAME=$(az acr list --resource-group votingapp-rg --query "[0].name"  
--output tsv)
```

Este es el comando que **arregla la conexión**

```
az aks update --name votingapp-aks --resource-group votingapp-rg  
--attach-acr $ACR_NAME
```

Verificamos y nos saldrá esto:

```
stevenvallejo@fedora:~/aks-modernization/manifests -- watch kubectl get pods
~/aks-modernization/manifests
Every 2.0s: kubectl get pods

NAME          READY   STATUS    RESTARTS   AGE
backend-deployment-bdb6bf4d4-k4mzb   1/1     Running   0          12m
frontend-deployment-67b88497db-bhlm4  1/1     Running   0          12m
redis-deployment-5f86f8f9c7-xfnmt    1/1     Running   0          12m

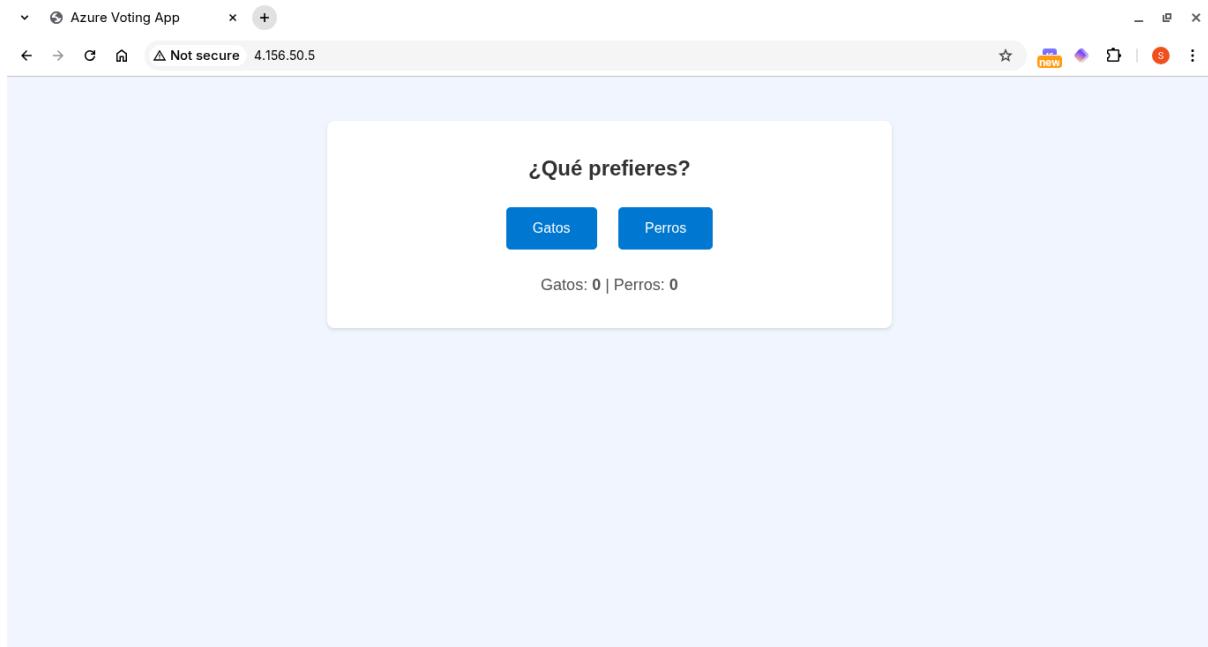
fedora: Thu Oct 16 14:03:17 2025
```

2. Ver el estado de los services y obtener la IP Pública:

```
kubectl get service
```

```
stevenvallejo@fedora:~/aks-modernization/manifests
~/aks-modernization/manifests
stevenvallejo@fedora:~/aks-modernization/manifests$ kubectl get service frontend-service
NAME        TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)      AGE
frontend-service   LoadBalancer  10.1.160.231  4.156.50.5   80:30500/TCP  14m
stevenvallejo@fedora:~/aks-modernization/manifests$ kubectl get service
NAME        TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)      AGE
backend-service   ClusterIP   10.1.2.11    <none>       80/TCP      15m
frontend-service   LoadBalancer  10.1.160.231  4.156.50.5   80:30500/TCP  15m
kubernetes      ClusterIP   10.1.0.1    <none>       443/TCP     3h18m
redis-service    ClusterIP   10.1.148.10  <none>       6379/TCP    15m
stevenvallejo@fedora:~/aks-modernization/manifests$
```

Y ahora con la **IP pública**, pegamos en el **navegador** y saldrá la aplicación web



Si deseas **apagarlo** antes de pasar a la siguiente fase para **evitar consumos extras**, utiliza el siguiente código en la terminal:

```
az aks stop --name votingapp-aks --resource-group votingapp-rg
```

Fase 4: Validación, Monitoreo y Resiliencia

Objetivo: Comprender cómo observar el estado de nuestra aplicación, demostrar la capacidad de auto-reparación de Kubernetes y ver cómo escalar la aplicación para manejar más carga.

Punto 4.1: Reanudar el clúster de Kubernetes

1. Abre tu terminal.
2. Ejecuta el comando start. Este proceso tardará entre 2 y 4 minutos.

```
az aks start --name votingapp-aks --resource-group votingapp-rg
```

3. Una vez que el comando termine, espera un minuto adicional para que todos los componentes internos se estabilice.

Revisa que los pods vuelvan al estado Running y que el frontend-service obtenga una nueva IP pública (o la misma de antes, puede variar).

```
kubectl get pods
```

```
kubectl get service frontend-service
```

4. Copia la nueva EXTERNAL-IP y pégala en tu navegador para confirmar que la aplicación está de nuevo en línea.

Punto 4.2: Demostración de alta disponibilidad (Auto-Reparación)

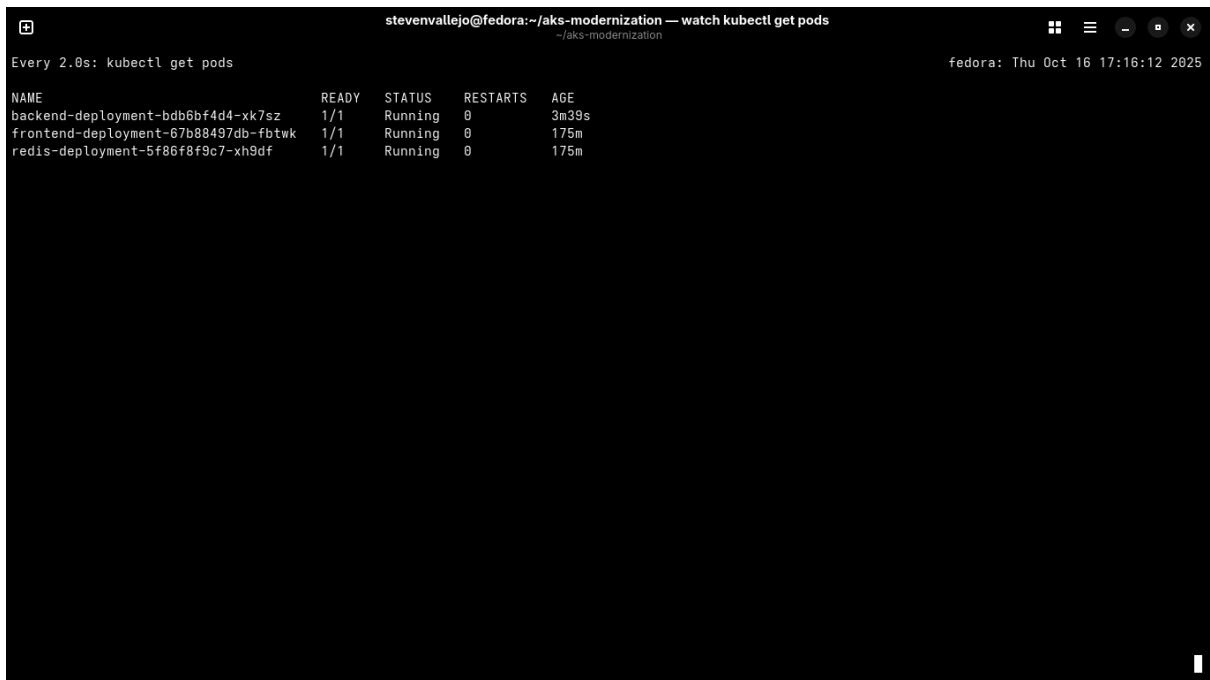
Esta es una de las características más poderosas de Kubernetes. Vamos a simular una falla catastrófica en nuestro backend y el sistema se recupera solo.

1. Abre dos terminales.

En la Terminal 1: Configura un "observador" para los pods.

```
watch kubectl get pods
```

Verás la lista de tus los pods en estado Running.



The screenshot shows a terminal window with the following content:

```
stevenvallejo@fedora:~/aks-modernization -- watch kubectl get pods
Every 2.0s: kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
backend-deployment-bdb6bf4d4-xk7sz   1/1     Running   0          3m39s
frontend-deployment-67b88497db-fbtwk  1/1     Running   0          175m
redis-deployment-5f86f8f9c7-xh9df    1/1     Running   0          175m
```

The terminal window has a dark background and light-colored text. It includes standard terminal navigation keys at the bottom and a timestamp in the top right corner: "fedora: Thu Oct 16 17:16:12 2025".

En la Terminal 2: Identifica el nombre exacto de tu pod de backend. Parecido algo como backend-deployment-bdb6bf4d4-k4mzb.

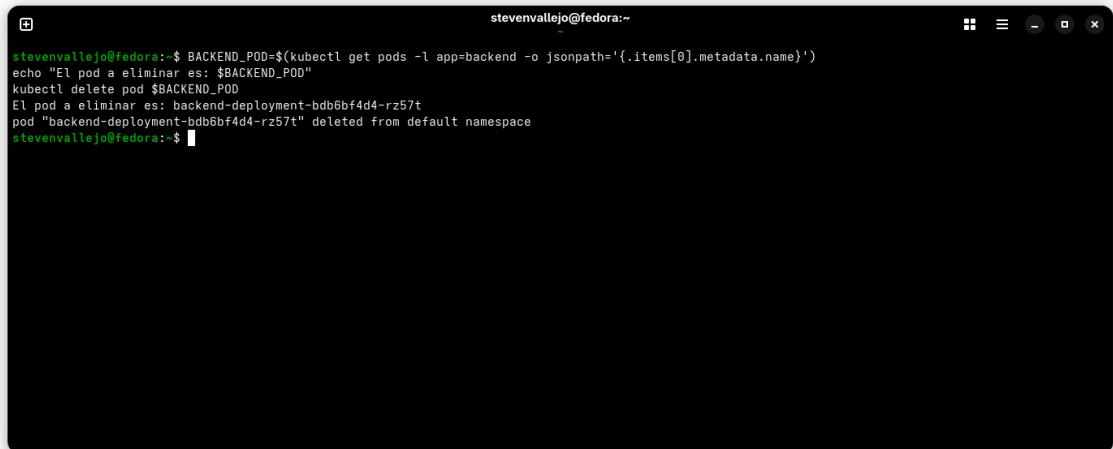
Primero obtén el **nombre exacto** del pod

```
BACKEND_POD=$(kubectl get pods -l app=backend -o jsonpath='{.items[0].metadata.name}')
```

```
echo "El pod a eliminar es: $BACKEND_POD"
```

Ahora se **simula** la falla

```
kubectl delete pod $BACKEND_POD
```



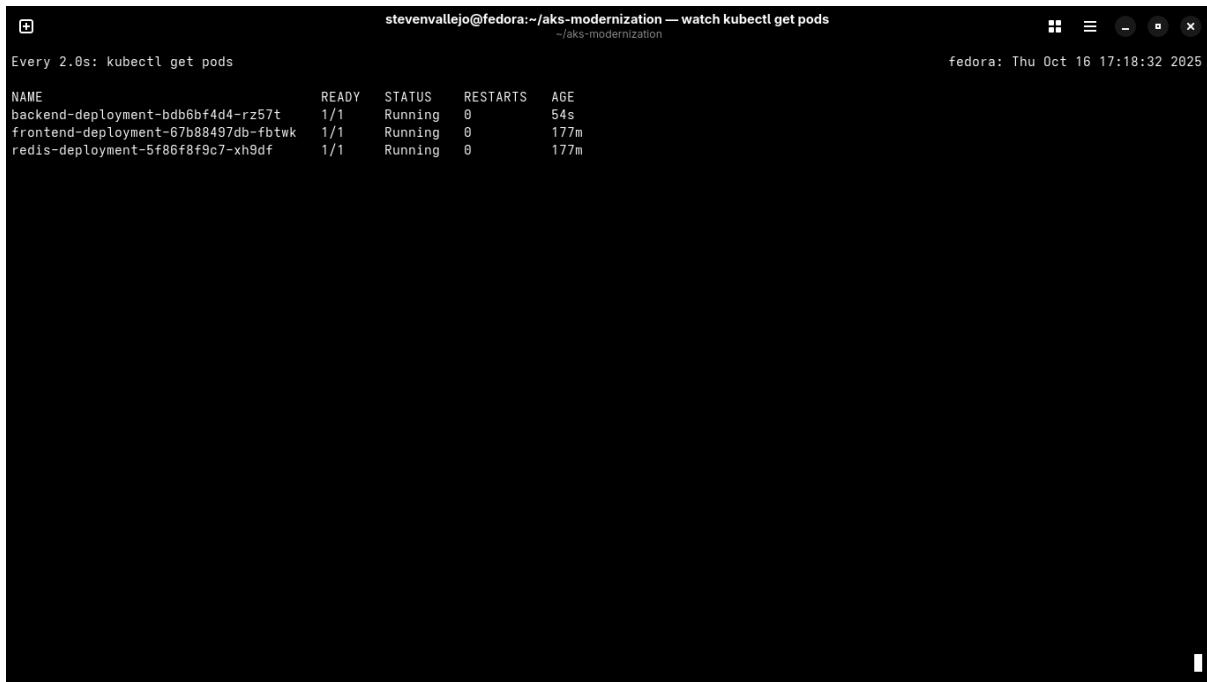
The screenshot shows a terminal window titled 'stevenvallejo@fedora:~'. The user has run a shell script that performs the following steps:

- It uses kubectl to get a list of pods labeled 'app=backend' and extracts the name of the first pod using a JSON path selector.
- It then prints the message "El pod a eliminar es: \$BACKEND_POD".
- Finally, it runs 'kubectl delete pod \$BACKEND_POD' which successfully deletes the pod named 'backend-deployment-bdb6bf4d4-rz57t' from the default namespace.

```
stevenvallejo@fedora:~$ BACKEND_POD=$(kubectl get pods -l app=backend -o jsonpath='{.items[0].metadata.name}')
echo "El pod a eliminar es: $BACKEND_POD"
kubectl delete pod $BACKEND_POD
El pod a eliminar es: backend-deployment-bdb6bf4d4-rz57t
pod "backend-deployment-bdb6bf4d4-rz57t" deleted from default namespace
stevenvallejo@fedora:~$
```

2. Observa la Terminal 1 (la del watch)

- El pod backend-... cambiará inmediatamente al estado Terminating.
- Casi al mismo tiempo, aparecerá un **nuevo** pod backend-... con un nombre diferente, y su estado será Pending, luego ContainerCreating, y finalmente Running.



A terminal window titled "stevenvallejo@fedora:~/aks-modernization — watch kubectl get pods -laks-modernization". The command "Every 2.0s: kubectl get pods" is running. The output shows three pods: backend-deployment-bdb6bf4d4-rz57t, frontend-deployment-67b88497db-fbtwk, and redis-deployment-5f86f8f9c7-xh9df. All three pods are in the "Running" state with 1/1 ready, 0 restarts, and an age of 54s or 177m. The terminal window has a dark background and light text, with standard Linux terminal icons at the top.

NAME	READY	STATUS	RESTARTS	AGE
backend-deployment-bdb6bf4d4-rz57t	1/1	Running	0	54s
frontend-deployment-67b88497db-fbtwk	1/1	Running	0	177m
redis-deployment-5f86f8f9c7-xh9df	1/1	Running	0	177m

Conclusión: El Deployment de Kubernetes detectó que el número de réplicas deseadas (1) no se cumplía y automáticamente creó un nuevo pod para reemplazar al que falló. **Esto es la auto-reparación en acción.** La aplicación probablemente no tuvo ninguna interrupción perceptible para el usuario.

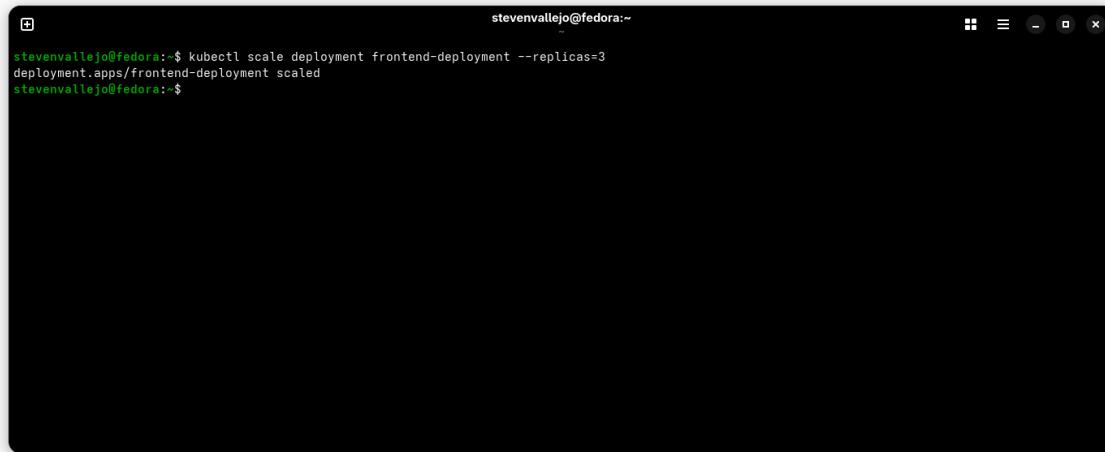
Punto 4.3: Demostrar la escalabilidad (Escalado Horizontal)

Digamos que la aplicación de votación se vuelve viral. Necesitamos más capacidad para manejar el tráfico.

1. Mantén la Terminal 1 con `watch kubectl get pods`.

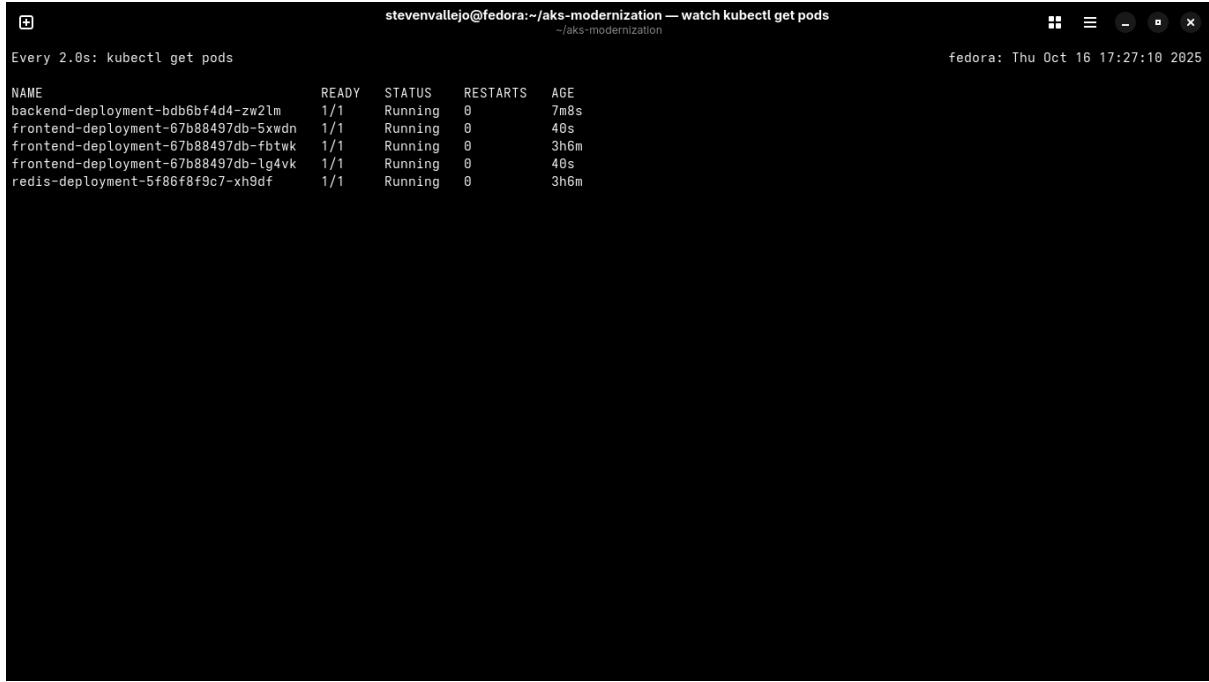
En la Terminal 2: Ejecuta el comando scale para decirle a Kubernetes que ahora quieres **tres** copias (réplicas) de tu frontend.

```
kubectl scale deployment frontend-deployment --replicas=3
```



```
stevenvallejo@fedora:~$ kubectl scale deployment frontend-deployment --replicas=3
deployment.apps/frontend-deployment scaled
stevenvallejo@fedora: $
```

2. **Observa la Terminal 1:** Inmediatamente, verás aparecer **dos nuevos pods** del frontend (frontend-deployment-...). Pasarán por los estados Pending, ContainerCreating y Running.



```
stevenvallejo@fedora:~/aks-modernization -- watch kubectl get pods
~/.aks-modernization
Every 2.0s: kubectl get pods
NAME           READY   STATUS    RESTARTS   AGE
backend-deployment-bdb6bf4d4-zw2lm  1/1     Running   0          7m8s
frontend-deployment-67bb8497db-5xwdn 1/1     Running   0          40s
frontend-deployment-67bb8497db-fbtwk 1/1     Running   0          3h6m
frontend-deployment-67bb8497db-lg4vk  1/1     Running   0          49s
redis-deployment-5f86f8f9c7-xh9df   1/1     Running   0          3h6m
```

En menos de un minuto, tendrás tres pods del frontend funcionando. El Service de tipo LoadBalancer que creamos automáticamente distribuirá el tráfico de los usuarios entre estos tres pods, triplicando la capacidad de tu aplicación.

Para volver a la normalidad y ahorrar recursos, ejecuta:

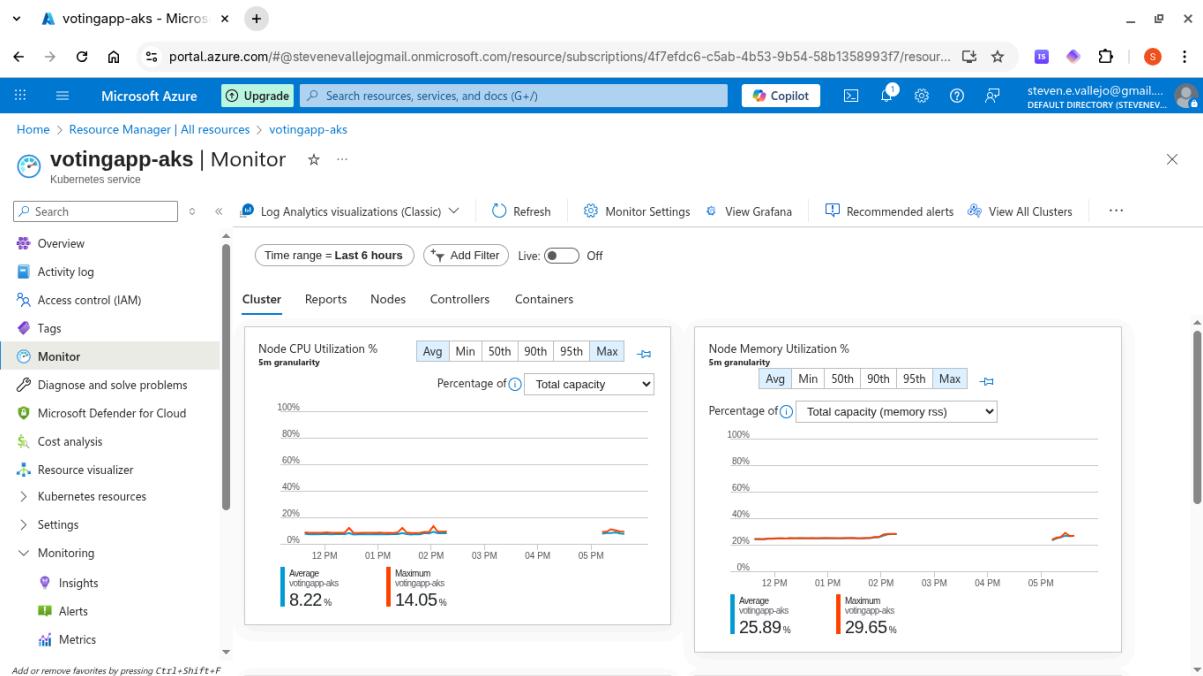
```
kubectl scale deployment frontend-deployment --replicas=1
```

Verás cómo los dos pods extra pasan al estado Terminating hasta que solo quede uno.

Punto 4.4: Explorar Azure Monitor for Containers (Observabilidad)

Ahora veamos el "diario de operaciones" que construimos en la Fase 2.

1. Ve al **Portal de Azure**.
2. Busca el clúster de Kubernetes, **votingapp-aks**.
3. En el menú de la izquierda, en la sección "**Monitoring**" (Monitoreo), haz clic en "**Insights**" (Información).
4. **Dale un minuto para que cargue los datos.** Verás un dashboard:
 - **Pestaña "Cluster":** Verás el uso promedio de CPU y memoria de tus nodos.
 - **Pestaña "Nodes":** Un desglose del rendimiento por cada nodo (en nuestro caso, solo uno).
 - **Pestaña "Controllers" y "Containers":** Aquí puedes profundizar y ver el uso de CPU y memoria de cada uno de tus pods (frontend, backend, redis).



5. Para ver los logs:

- Haz clic en "View Logs" o busca la pestaña "Logs".
- Se abrirá una ventana de consulta. En el panel izquierdo, puedes navegar por las tablas. La más importante es ContainerLog.

The screenshot shows the Azure AKS logs interface. On the left, there's a sidebar with navigation links like 'Resource visualizer', 'Kubernetes resources', 'Monitoring', 'Insights', 'Alerts', 'Metrics', 'Diagnostic settings', 'Advisor recommendations', 'Logs' (which is selected), 'Workbooks', 'Dashboards with Grafana (preview)', 'Automation', and 'Help'. The main area has a search bar and a 'New Query' button. Below that is a 'Queries' section with a search bar and a 'Favorites' section. The main content area shows a table titled 'ContainerLogs' with columns: TimeOfCommand [UTC], ContainerID, and LogEntry. The table displays several log entries from two containers, 'pool-32314077-vmss...' and 'pool-32314077-vmss...', showing log levels like INFO and DEBUG. A tooltip for the 'Container Logs' section explains that it finds a value in the ContainerLogs table based on a specified string in the LogEntry column.

Pega la siguiente consulta en la ventana y haz clic en "Run". Esto te mostrará los últimos 100 logs de tus contenedores:

```
ContainerLog
```

```
| take 100
```

```
| sort by TimeGenerated desc
```

- Puedes ver los print() que pusimos en el código de Python, lo que es invaluable para la depuración.

TimeGenerated [UTC]	Computer	TimeOfCommand [UTC]	ContainerID	LogEntry
> 10/16/2025, 10:47:06.692 PM	aks-agentpool-32314077-vms...	10/16/2025, 10:47:21.000 PM	449ec0167764841e0011b47be...	10.0.0.10 - - [16/Oct/2025:21:00:00 +0000] "GET / HTTP/1.1" 200 1234
> 10/16/2025, 10:47:06.690 PM	aks-agentpool-32314077-vms...	10/16/2025, 10:47:21.000 PM	9a8521975e19fcfa39d18f2d8c9...	INFO: 10.0.0.27:33012 - "GET / HTTP/1.1" 200 1234
> 10/16/2025, 10:20:04.888 PM	aks-agentpool-32314077-vms...	10/16/2025, 10:20:36.000 PM	9a8521975e19fcfa39d18f2d8c9...	INFO: Unicorn running on http://127.0.0.1:33012
> 10/16/2025, 10:20:04.880 PM	aks-agentpool-32314077-vms...	10/16/2025, 10:20:36.000 PM	9a8521975e19fcfa39d18f2d8c9...	INFO: Application startup completed.
> 10/16/2025, 10:20:04.880 PM	aks-agentpool-32314077-vms...	10/16/2025, 10:20:36.000 PM	9a8521975e19fcfa39d18f2d8c9...	INFO: Waiting for application to start up.
> 10/16/2025, 10:20:04.880 PM	aks-agentpool-32314077-vms...	10/16/2025, 10:20:36.000 PM	9a8521975e19fcfa39d18f2d8c9...	INFO: Started server process.
> 10/16/2025, 10:20:02.871 PM	aks-agentpool-32314077-vms...	10/16/2025, 10:20:06.000 PM	296bba14250c99fe360ba002ab...	INFO: Finished server process.
> 10/16/2025, 10:20:02.871 PM	aks-agentpool-32314077-vms...	10/16/2025, 10:20:06.000 PM	296bba14250c99fe360ba002ab...	INFO: Application shutdown.
> 10/16/2025, 10:20:02.871 PM	aks-agentpool-32314077-vms...	10/16/2025, 10:20:06.000 PM	296bba14250c99fe360ba002ab...	INFO: Waiting for application to shut down.

Ejecuta el siguiente comando para **eliminar** todos los **recursos** que hemos utilizado para este proyecto para evitar el pago innecesario.

```
az group delete --name votingapp-rg --yes --no-wait
```

- **--name votingapp-rg:** Especifica el grupo de recursos que contiene todo nuestro trabajo.
- **--yes:** Confirma la eliminación sin pedirte una segunda confirmación.
- **--no-wait:** Devuelve el control de la terminal inmediatamente y la eliminación se ejecuta en segundo plano.

ADVERTENCIA: Asegúrate que se haya eliminado visualizando en el Portal de Azure, de no ser así, **elimina manualmente**.

Conclusión del proyecto.-

Se modernizó exitosamente una aplicación a una arquitectura de microservicios en Azure Kubernetes Service (AKS). Usando Bicep (IaC), se desplegó toda la infraestructura de forma automatizada, creando una solución funcional, escalable y con capacidad de auto-reparación. El proyecto demostró los principios clave de la nube nativa y se completó con un costo operativo mínimo, validando la eficiencia de la plataforma.