

DevOps: Trabajo Práctico 2

Bienvenido al repositorio del trabajo práctico 2 del cursado 2025 de DevOps, realizado por:

- Aldo Omar Andres.
- Agustín Nicolás Bravo Pérez.
- Ariano Miranda.

Links relevantes:

- [Consigna](#).
- [Repositorio](#).
- [Sitio web](#).
- [Dashboard de la App](#) (usuario `admin` y clave `prom-operator`).
- [Trazas](#).

Aplicación: Lista de Tareas

Construimos una simple *todo application* con los siguientes componentes:

- Una app web desarrollada con React y Vite.
- Un servidor desarrollado con TypeScript y Express.js.
- Una base de datos Redis.

La aplicación permite ver la lista de tareas, crear tareas nuevas y actualizar o eliminar tareas existentes. Su arquitectura de software es la siguiente:



Flujo de datos:

1. El usuario interactúa con la UI (frontend).
2. React hace peticiones a la API REST (backend).
3. El backend procesa las peticiones y envía peticiones a Redis.
4. Redis responde las peticiones del backend, quien luego responde al frontend.

Estructura del Proyecto

```
utn-devops-tp2
├── .github      # Definición de la GitHub Actions
├── backend      # Servidor backend con TypeScript y Express.js
│   ├── package.json
│   └── Dockerfile
└── frontend     # App web frontend con React y Vite
    ├── src
    │   └── index.jsx
    └── package.json
```

```
|   └── Dockerfile  
├── k8s           # Manifiestos para el cluster k3s  
├── docker-compose.yml  
└── README.md
```

Desarrollo

Requisitos para levantar el proyecto:

- Docker.

1. Clonar el repositorio:

```
git clone https://github.com/agustinbravop/utn-devops-tp2.git  
cd utn-devops-tp2
```

2. Construir y ejecutar la aplicación usando Docker Compose:

```
docker compose up
```

3. Visitar la UI en `http://localhost:3000` y la API en `http://localhost:3001`.

Se pueden definir las siguientes variables de entorno:

- `frontend/.env` :

```
VITE_API_URL=http://localhost:3001/api  
VITE_OTEL_ENDPOINT=http://localhost:4318/v1/traces
```

- `backend/.env` :

```
REDIS_URL=redis://localhost:6379  
PORT=80  
OTEL_EXPORTER_OTLP_ENDPOINT=http://localhost:4318/v1/traces  
LOG_LEVEL=info
```

Infraestructura

Se utiliza Microsoft Azure para desplegar la aplicación en un cluster de Kubernetes. Para respetar la consigna, en lugar de utilizar Azure Kubernetes Service, vamos a instalar k3s en una máquina virtual. Existen recursos que se deben crear manualmente mediante la CLI de Azure:

```
# Previamente se debe haber instalado Azure CLI.  
# Ver: https://learn.microsoft.com/en-us/cli/azure/install-azure-cli.  
#     brew install azure-cli  
  
# Iniciar sesión con el correo académico y elegir la suscripción "Azure para estudiantes".  
az login  
  
# Registrarse en proveedores de Azure que "Azure para estudiantes" no da por defecto.  
export RESOURCE_GROUP="utn-devops-tp2"
```

```

export LOCATION="eastus"
export SERVER_VM="k3s-server"
export AGENT_VM="k3s-agent"

# Crear un Resource Group para agrupar todos los recursos a crear.
az group create --name $RESOURCE_GROUP --location $LOCATION

# Crear máquinas virtuales (un server y un agent según la arquitectura de k3s).
az vm create \
    --resource-group $RESOURCE_GROUP \
    --name $SERVER_VM \
    --image Ubuntu2404 \
    --size Standard_B2s \
    --admin-username azureuser \
    --generate-ssh-keys
az vm create \
    --resource-group $RESOURCE_GROUP \
    --name $AGENT_VM \
    --image Ubuntu2404 \
    --size Standard_B2s \
    --admin-username azureuser \
    --generate-ssh-keys

# Abrir puertos para web HTTP, redis insight, la API de Kubernetes y el supervisor de k3s.
az vm open-port --resource-group $RESOURCE_GROUP --name $SERVER_VM --port 6443,10250
az vm open-port --resource-group $RESOURCE_GROUP --name $AGENT_VM --port 80,30540,6443,10250

# Instalar k3s en el server (--tls-san se usa para permitir el acceso mediante la IP pública).
$SERVER_PUBLIC_IP=$(az vm show --name $SERVER_VM --resource-group $RESOURCE_GROUP --show-details --query "publicIp")
az vm run-command invoke \
    --resource-group $RESOURCE_GROUP \
    --name $SERVER_VM \
    --command-id RunShellScript \
    --scripts "curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC='server --tls-san ${SERVER_PUBLIC_IP}' sh -"

# Obtener el token del server (lo necesita el agent).
$K3S_TOKEN=$(az vm run-command invoke \
    --resource-group $RESOURCE_GROUP \
    --name $SERVER_VM \
    --command-id RunShellScript \
    --scripts "sudo cat /var/lib/rancher/k3s/server/node-token" \
    --query "value[0].message" \
    --output tsv \
    | head -n -3 | tail -n +3) # Quedarse solo con stdout

$SERVER_PRIVATE_IP=$(az vm show --name $SERVER_VM --resource-group $RESOURCE_GROUP --show-details --query "privateIp")

# Instalar k3s en el agent.
az vm run-command invoke \
    --resource-group $RESOURCE_GROUP \
    --name $AGENT_VM \
    --command-id RunShellScript \
    --scripts "curl -sfL https://get.k3s.io | K3S_URL=https://$SERVER_PRIVATE_IP:6443 K3S_TOKEN=$K3S_TOKEN sh -"

```

Una vez creadas las máquinas virtuales e instalado k3s, necesitamos conectarnos al cluster de Kubernetes:

```

# Previamente se debe haber instalado `kubectl`, la CLI de Kubernetes.
# Ver: https://kubernetes.io/docs/tasks/tools/#kubectl.
# brew install kubectl

# Obtener el archivo kubeconfig del server (asociado al superusuario admin).
az vm run-command invoke \
    --resource-group $RESOURCE_GROUP \
    --name $SERVER_VM \
    --command-id RunShellScript \
    --scripts "sudo cat /etc/rancher/k3s/k3s.yaml | sudo base64" \
    --query "value[0].message" \
    --output tsv \

```

```

| head -n -3 \
| tail -n +3 \
| base64 --decode \
| sed "s/127.0.0.1/$SERVER_PUBLIC_IP/" > kubeconfig.yaml

# Probar la conexión al cluster recién creado.
export KUBECONFIG=kubeconfig.yaml
kubectl get nodes

```

Para eliminar todos los recursos creados:

```
az group delete --name $RESOURCE_GROUP --yes
```

Kubernetes

El resto de servicios se despliegan sobre el cluster de Kubernetes, por lo que nos abstraemos de Microsoft Azure. Se puede acceder al cluster utilizando el archivo `kubeconfig.yaml` generado anteriormente. En la carpeta `/k8s/app` se definen los manifiestos de la aplicación. Para desplegar todos los manifiestos en Kubernetes:

```
kubectl apply -k k8s/
```

Archivos:

```

k8s/
└── app/
    ├── backend-deployment.yaml      # Deploys backend API
    ├── backend-hpa.yaml            # Horizontal scaling for backend based on CPU/memory
    ├── backend-service.yaml        # Exposes backend on port 80
    ├── configmap.yaml              # Environment variables
    ├── frontend-deployment.yaml    # Deploys frontend React SPA on nginx
    ├── frontend-service.yaml       # Exposes frontend
    ├── ingress.yaml                # Routes traffic to frontend (/) and backend (/api)
    ├── kustomization.yaml
    ├── namespace.yaml
    ├── redis-deployment.yaml      # Deploys Redis cache
    ├── redis-insight-deployment.yaml # Deploys Redis Insight GUI
    ├── redis-insight-service.yaml # Exposes Redis Insight
    └── redis-service.yaml         # Exposes Redis

    └── kustomization.yaml

└── monitoring/
    ├── app-dashboard.json          # Grafana custom dashboard for the app
    ├── backend-servicemonitor.yaml  # Prometheus ServiceMonitor for backend metrics
    ├── grafana-loki-datasource.yaml # Datasource config for Loki logs
    ├── grafana-tempo-datasource.yaml # Datasource config for Tempo traces
    ├── ingress.yaml                 # Ingress for Grafana UI at /grafana
    ├── kustomization.yaml
    ├── loki-values.yaml            # Helm values for Loki log aggregation
    ├── namespace.yaml
    ├── otel-collector-http-service.yaml # Exposes OpenTelemetry Collector HTTP endpoint
    ├── otel-collector-ingress.yaml   # Ingress for OTLP traces at /otel endpoint
    ├── otel-collector-middleware.yaml # Traefik middleware to strip /otel prefix
    ├── promtail-values.yaml         # Helm values for Promtail log shipping
    ├── values-monitoring.yaml      # Helm values for Prometheus/Grafana stack
    ├── values-otel-collector.yaml  # Helm values for OpenTelemetry Collector
    └── values-tempo.yaml           # Helm values for Tempo distributed tracing

```

Observabilidad

En la observabilidad son fundamentales las métricas, los logs y las trazas.

Métricas

Para métricas se utiliza Prometheus (que scrapea las métricas de cada servicio) y Grafana (que las visualiza en dashboards). En `k8s/monitoring` se definen algunos manifiestos adicionales, pero la instalación es mediante helm:

```
# Prometheus y Grafana se instalan mediante Helm, un gestor de "paquetes" de Kubernetes.
# Ver: https://helm.sh/docs/intro/install.
#       brew install helm
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm upgrade --install monitoring prometheus-community/kube-prometheus-stack \
--namespace monitoring \
--values k8s/monitoring/values-monitoring.yaml

# Obtener usuario y contraseña de admin.
kubectl --namespace monitoring get secret monitoring-grafana -o jsonpath=".data.admin-user" | base64 --decode
kubectl --namespace monitoring get secret monitoring-grafana -o jsonpath=".data.admin-password" | base64 --dec
```

El backend genera dos métricas `http_requests_total` y `http_request_duration_seconds_bucket` en el endpoint `/api/metrics`. Estas métricas se pueden ver en un [dashboard de Grafana](#). Las credenciales por defecto de grafana son usuario `admin` y contraseña `prom-operator`.

Trazas

Las trazas se capturan con OpenTelemetry Collector y se almacenan en Tempo. Ambos servicios se instalan con `helm`:

```
helm repo add open-telemetry https://open-telemetry.github.io/opentelemetry-helm-charts
helm repo update

helm upgrade --install otel-collector open-telemetry/opentelemetry-collector \
--namespace monitoring \
--values k8s/monitoring/values-otel-collector.yaml

helm upgrade --install tempo grafana/tempo \
--namespace monitoring \
--values k8s/monitoring/values-tempo.yaml
```

Las trazas se pueden [explorar desde Grafana](#).

Logs

Para logs se utiliza Promtail (que scrapea los logs de los pods) y Loki (que los almacena). Luego se puede consultar estos logs desde Grafana. Promtail y Loki se instalan mediante `helm`:

```
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update

helm upgrade --install loki grafana/loki \
--namespace monitoring \
--create-namespace \
--values k8s/monitoring/loki-values.yaml

helm upgrade --install promtail grafana/promtail \
```

```
--namespace monitoring \
--values k8s/monitoring/promtail-values.yaml
```

Despliegue Continuo

Se tiene una GitHub Action para la integración continua y despliegue continuo. Este workflow requiere los siguientes Repository Secrets:

```
DOCKERHUB_USERNAME
DOCKERHUB_TOKEN
KUBECONFIG_BASE64
```

Pasos de un despliegue al hacer un `git push`:

1. GitHub Actions ejecuta todos los pasos de integración continua.
2. GitHub Actions construye las imágenes de contenedores y las publica en Docker Hub.
3. GitHub Actions se conecta al cluster de Kubernetes para redesplegar los servicios, quienes descargan la nueva imagen de Docker Hub.



Tareas Pendientes

Esta lista NO es exhaustiva!

- Instalar un cluster de Kubernetes con k3s en Microsoft Azure.
- Implementar despliegue continuo de la aplicación base.
- Exponer una acción que genere carga controlada.
- Desplegar los servicios en Pods (conviene utilizar un Deployment).
- Desplegar un servicio o ingress para exponer a la web.
- Configurar alta disponibilidad para que se levanten nuevos nodos conforme aumenta la carga de la app.
- Implementar Loki para logs.
- Emitir logs estructurados en cada servicio de la app.
- Agregar logs al dashboard de la app (también agregar un panel más para la métrica `http_request_duration_seconds`).
- Implementar OpenTelemetry para trazas.
- Implementar Prometheus para métricas.
- Agregar una métrica que sea un indicador de la aplicación.
- Implementar Grafana para visualización con gráficos y paneles.
- Opcional: implementar IaC con Terraform para aprovisionar un cluster de Kubernetes.
- Opcional: exponer la aplicación en un dominio (evitando así la URL HTTP cruda).
- Opcional: redesplegar servicios SOLO cuando se rebuildea su imagen. Rebuildear imágenes SOLO si cambia el código fuente de ese servicio.
- Opcional: migrar de Promtail a Alloy (Promtail fue deprecado a inicio de año).
- Opcional: escalado horizontal de Redis (acá habría que investigar si es suficiente el Deployment de redis).