

PROYECTO FINAL

SISTEMA DE VENTA DE BOLETTOS CON MICROSERVICIOS

COMPUTACION TOLERANTE A FALLAS
CICLO 2025B

DANIEL GAITAN CHAVEZ
219294005

DISEÑO DE MICROSERVICIOS

- Frontend Service (Node.js)
 - Gateway de usuario. Manejo de interfaz y degradación de errores.
- Logic Service (Python/Flask)
 - Núcleo de procesamiento.
 - Aislamiento total de internet (ClusterIP).
- Comunicación
 - Uso de DNS interno
 - de Kubernetes y API REST.

```
● [+] Building 22.7s (11/11) FINISHED
=> [internal] load build definition from dockerfile
=> => transferring dockerfile: 209B
=> [internal] load metadata for docker.io/library/python:3.9-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 1.57kB
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:2d97f6910b16bd338d3060f261f53f144965f755599aab1acda1e13cf1731b1b
=> => resolve docker.io/library/python:3.9-slim@sha256:2d97f6910b16bd338d3060f261f53f144965f755599aab1acda1e13cf1731b1b
=> => sha256:ea56f685404adf81680322f152d2cfec62115b30dda481c2c450078315beb508 251B / 251B
=> => sha256:fc74430849022d13b0d44b8969a953f842f59c6e9d1a0c2c83d710affa286c08 13.88MB / 13.88MB
=> => sha256:b3ec39b36ae8c03a3e09854de4ec4aa08381dfed84a9daa075048c2e3df3881d 1.29MB / 1.29MB
=> => sha256:38513bd7256313495cd83b3b0915a633cfa475dc2a07072ab2c8d191020ca5d 29.78MB / 29.78MB
=> => extracting sha256:38513bd7256313495cd83b3b0915a633cfa475dc2a07072ab2c8d191020ca5d
=> => extracting sha256:b3ec39b36ae8c03a3e09854de4ec4aa08381dfed84a9daa075048c2e3df3881d
=> => extracting sha256:fc74430849022d13b0d44b8969a953f842f59c6e9d1a0c2c83d710affa286c08
=> => extracting sha256:ea56f685404adf81680322f152d2cfec62115b30dda481c2c450078315beb508
=> [2/5] WORKDIR /app
=> [3/5] COPY requirements.txt .
=> [4/5] RUN pip install --no-cache-dir -r requirements.txt
=> [5/5] COPY .
=> exporting to image
=> => exporting layers
=> => exporting manifest sha256:a6c19f93454b5489bd91f51e823921df68c197ca6b5bf26500ea7b6a12142961
=> => exporting config sha256:30c2d62f0a9f4df2d398d6ea3fabba41e97d8366e8d189e67d069ec6333925b
=> => exporting attestation manifest sha256:8e72a1ab33d60bc6902aa4f9642141257d461f14274ca3310e0b3c6a918e4a30
=> => exporting manifest list sha256:621fd39e2e3a0b03a97583f90143445768cb5d81ad011b13341a90056acf12c6
=> => naming to docker.io/library/flame-logic:v1
=> => unpacking to docker.io/library/flame-logic:v1
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas>
```

```
xt... -a---- 12/8/2025 8:16 PM 961 Readme.md
lo...
ic...
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas> kubectl apply -f ./Proyecto_Final/k8s/main-deployment.yaml
error: the path "./Proyecto_Final/k8s/main-deployment.yaml" does not exist
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas> docker build -t flame-frontend:v1 ./Proyecto_Final/service-frontend
docke...
❖ [+] Building 18.3s (5/10)
=> [internal] load build definition from dockerfile
=> => transferring dockerfile: 170B
=> [internal] load metadata for docker.io/library/node:18-alpine
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/5] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e
=> => resolve docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d118945588d0a35cb9bc4b8ca09d9e
=> => sha256:1e5a4c89cee5c0826c540ab06d4b6b491c96eda01837f430bd47f0d26702d6e3 1.26MB / 1.26MB
=> => sha256:25ff2da83641908f65c3a74d80409d6b1b62ccfaab220b9ea70b80df5a2e0549 446B / 446B
=> => sha256:dd71dde834b5c203d162902e6b8994cb2309ae049a0eabc4efeca161b2b5a3d0e 40.01MB / 40.01MB
=> => sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870 3.64MB / 3.64MB
=> => extracting sha256:f18232174bc91741fdf3da96d85011092101a032a93a388b79e99e69c2d5c870
=> [internal] load build context
=> => transferring context: 3.39kB
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas>
```

- - **1. SELF-HEALING (AUTORECUPERACIÓN)**
REINICIO AUTOMÁTICO DE PODS ANTE FALLOS.

- - **2. SEGURIDAD POR AISLAMIENTO**
BACKEND SIN IP PÚBLICA,
SOLO ACCESIBLE VÍA RED INTERNA.

- **3. OBSERVABILIDAD NATIVA**
MONITOREO DE LOGS EN TIEMPO REAL (KUBECTL LOGS).

ESTRATEGIAS DE TOLERANTE A FALLAS

SEGURIDAD Y COMUNICACIÓN INTER-SERVICIOS

PARA CUMPLIR CON LA SEGURIDAD, AISLAMOS EL SERVICIO DE LÓGICA.

NADIE DESDE INTERNET PUEDE TOCAR LA BASE DE DATOS NI EL CÓDIGO PYTHON DIRECTAMENTE, SOLO EL FRONTEND TIENE PERMISO PARA HABLARLE MEDIANTE LA RED INTERNA SEGURA DE KUBERNETES

SEGURIDAD Y COMUNICACIÓN INTERSERVICIOS

- AISLAMIENTO DE RED: IMPLEMENTACIÓN DE ARQUITECTURA "ZERO TRUST" PARCIAL. EL BACKEND (PYTHON) NO TIENE IP PÚBLICA; VIVE EN UNA RED PRIVADA (CLUSTERIP) ACCESIBLE ÚNICAMENTE POR EL FRONTEND.
- SERVICE DISCOVERY: USO DEL DNS INTERNO DE KUBERNETES ([HTTP://FLAME-LOGIC-SERVICE](http://FLAME-LOGIC-SERVICE)) PARA EVITAR EL USO DE IPS FIJAS Y VULNERABLES.
- PROTOCOLO: COMUNICACIÓN VÍA REST API (JSON) ESTANDARIZADA.

Kubernetes

Enable Kubernetes

Start a Kubernetes single or multi-node cluster when starting Docker Desktop.

Cluster



docker-desktop

kubeadm, 1 node, v1.34.1

● Running

Started 19 minutes ago

Reset cluster

Cluster settings

Choose cluster provisioning method

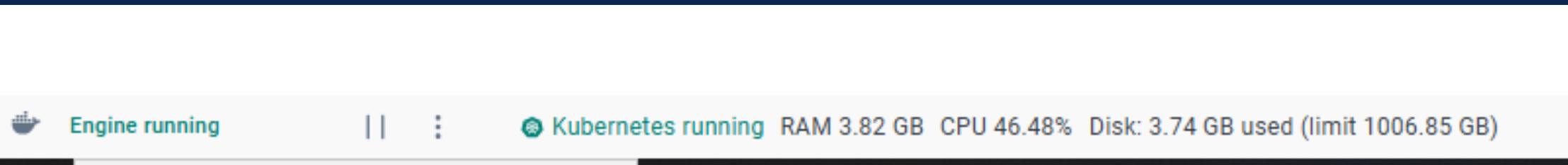
Kubeadm

Create a single-node cluster with kubeadm.

Version: v1.34.1

[Close](#)

[Apply](#)



ESTRATEGIA DE OBSERVABILIDAD (LOGGING)

"EN LUGAR DE INSTALAR HERRAMIENTAS PESADAS, UTILIZAMOS LA OBSERVABILIDAD NATIVA DE KUBERNETES. CONFIGURAMOS LOS CONTENEDORES PARA EMITIR LOGS ESTRUCTURADOS QUE NOS PERMITEN RASTREAR CADA TRANSACCIÓN DE VENTA Y DETECTAR ERRORES EN TIEMPO REAL SIN ENTRAR AL SERVIDOR.

MONITORIZACIÓN

- LOGGING ESTRUCTURADO: IMPLEMENTACIÓN DE LOGS ESTANDARIZADOS EN STDOUT/STDERR EN AMBOS MICROSERVICIOS.
- CENTRALIZACIÓN: KUBERNETES AGREGAN LOS LOGS DE TODOS LOS PODS (RUNNING/TERMINATED) PERMITIENDO AUDITORÍA EN TIEMPO REAL.
- TRAZABILIDAD: CADA PETICIÓN GENERA UN ID Y MENSAJE DE ESTADO (HTTP 200/500) RASTREABLE MEDIANTE KUBECTL LOGS.

VERIFICAR Y PROBAR

```
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s> kubectl apply -f main-deployment.yaml
● deployment.apps/logic-deployment created
  service/flame-logic-service created
  deployment.apps/frontend-deployment created
  service/flame-frontend-service created
○ PS C:\Users\Flame\Documents\GitHub\Computacion tolerante fallas\Provecto Final\k8s> █
```

VERIFICAR QUE LOS PODS ESTÉN CORRIENDO: KUBECTL GET PODS

VERIFICAR Y PROBAR

VERIFICAR QUE LOS PODS ESTÉN CORRIENDO: KUBECTL GET PODS

```
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s> kubectl get pods
● NAME                      READY   STATUS    RESTARTS   AGE
  frontend-deployment-68565f99bf-tbc42   1/1     Running   0          60s
  logic-deployment-656594bf66-bpdtt      1/1     Running   0          60s
  logic-deployment-656594bf66-gkk5j      1/1     Running   0          60s
○ PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s>
```



Sistema de Microservicios Resiliente (Node.js + Python)

ACCEDER A LA WEB

ENTRA A [HTTP://LOCALHOST](http://localhost) EN TU NAVEGADOR.
DEBERÁS VER LA INTERFAZ DEL CONCIERTO.

Concierto: Katia & Flame - Acoustic Tour 2025

Estado del sistema: **ONLINE**

COMPRAR BOLETO VIP

¡Compra Exitosa!

ID Ticket: TKT-1001

Mensaje: ¡Tu lugar para Katia & Flame está reservado!

Procesado por: Procesado en contenedor Python: logic-deployment-656594bf66-2vb7w

"**COMPRAR BOLETO**". SI TODO FUNCIONA, VERÁS EL ID DEL
TICKET Y QUÉ CONTENEDOR LO PROCESÓ.

[Volver al inicio](#)

INGENIERÍA DEL CAOS: PRUEBA DE FUEGO

EL EXPERIMENTO

EL EXPERIMENTO:
ELIMINACIÓN MANUAL DE UN POD CRÍTICO
(KUBECTL DELETE POD).

RESULTADO

EL REPLICASET DETECTÓ LA FALLA Y
CREÓ UNA NUEVA INSTANCIA EN
POCOS SEGUNDOS SEGUNDOS.

Ingeniería del Caos: Prueba de Fuego

Chaos Engineering: Trial by Fire

El Experimento:

- Eliminación manual de un pod crítico (`kubectl delete pod`).



Resultado:

- El ReplicaSet detectó la falla y creó una nueva instancia en < 5 segundos.



INGENIERIA DEL CAOS

CHAOS ENGINEERING

1. EL ATAQUE: ELIMINAMOS MANUALMENTE UN POD DEL SERVICIO DE LÓGICA PARA SIMULAR UN "CRASH" O FALLO FATAL. COMANDO: KUBECTL DELETE POD LOGIC-DEPLOYMENT-XXXXX

```
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s>
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s> kubectl delete pod logic-deployment-656594bf66-bpdtt
pod "logic-deployment-656594bf66-bpdtt" deleted from default namespace
[]
```

2. LA RECUPERACIÓN (SELF-HEALING): KUBERNETES DETECTA QUE FALTA UNA RÉPLICA Y CREA UNA NUEVA INMEDIATAMENTE. EN LA SIGUIENTE CAPTURA SE OBSERVA:

```
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s>
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s> kubectl delete pod logic-deployment-656594bf66-bpdtt
● pod "logic-deployment-656594bf66-bpdtt" deleted from default namespace
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s>
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s>
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s> kubectl get pods
● NAME           READY   STATUS    RESTARTS   AGE
  frontend-deployment-68565f99bf-tbc42   1/1     Running   0          6m31s
  logic-deployment-656594bf66-gkk5j      1/1     Running   0          6m31s
  logic-deployment-656594bf66-phqdg      1/1     Running   0          39s
PS C:\Users\Flame\Documents\GitHub\Computacion_tolerante_fallas\Proyecto_Final\k8s> []
```

PODS ANTIGUOS (6 MINUTOS DE VIDA).

* NUEVO POD (39 SEGUNDOS DE VIDA) CREADO AUTOMÁTICAMENTE PARA REEMPLAZAR AL ELIMINADO.

STACK TECNOLÓGICO Y CI/CD

- Docker
 - Contenerización independiente (node:alpine y python:slim).
- Kubernetes (K8s)
 - Orquestación, Self-Healing y Balanceo de Carga.
- Automatización (CI/CD)
 - Script deploy.ps1 automatiza Build -> Tag -> Deploy.



PIPLINE DE AUTOMATIZACION (CI/CD)

- Script de Despliegue (deploy.ps1): Automatización del ciclo de vida completo.
- Etapas del Pipeline:
- Build: Construcción de imágenes Docker limpias (docker build).
- Tag: Etiquetado de versiones (v1).
- Deploy: Aplicación de manifiestos en el clúster (kubectl apply).
- Resultado: Reducción del tiempo de despliegue de minutos a segundos y eliminación de error humano.

```
docker build -t flame-frontend:v1
./service-frontend
docker build -t flame-logic:v1
./service-logic
kubectl apply -f ./k8s/main-
deployment.yaml
```

MUCHAS GRACIAS

DANIEL GAITAN CHAVEZ

219294005

2025B