📸 Evidencias del Proyecto: Arquitectura de Microservicios Distribuidos en AWS

Este documento contiene evidencias visuales y técnicas del despliegue e integración de los microservicios, la infraestructura en AWS (con Terraform), y la comunicación basada en eventos mediante Kafka.

---

## 📁 1. Repositorios

- [x] Repositorio central con `README.md` y estructura general.

- [x] Repositorios individuales por microservicio:

| Servicio | Repositorio |

|----------------------|------------------------------------|

| time-service | `https://github.com/tu-user/time-service` |

| timezone-service | `https://github.com/tu-user/timezone-service` |

| minutes-service | `https://github.com/tu-user/minutes-service` |

| aggregator-service | `https://github.com/tu-user/aggregator-service` |

| formatter-service | `https://github.com/tu-user/formatter-service` |

📷 \*Capturas:\*

- Estructura de carpetas del repositorio principal

- Dockerfile y `main.go` o `app.py` por microservicio

---

## ☁️ 2. Terraform

- [x] Archivo `main.tf` para EC2, VPC, ALB y ASG.

- [x] Variables configuradas (`variables.tf`)

- [x] Ejecución de `terraform plan` y `terraform apply`

📷 \*Capturas:\*

- Código de Terraform desplegando instancias

- Salida del comando `terraform apply`

- Recursos creados en AWS Console: EC2, ALB, ASG

---

## 🚀 3. Despliegue de Microservicios en EC2

- [x] 2 instancias EC2 por servicio

- [x] Docker instalado y contenedores en ejecución

📷 \*Capturas:\*

- Lista de EC2 en AWS

- SSH a una instancia con `docker ps`

- Logs mostrando ejecución del servicio

---

## ⚙️ 4. Bases de Datos

- [x] MongoDB ejecutándose (EC2 o Atlas)

- [x] MySQL funcionando (RDS o EC2)

📷 \*Capturas:\*

- Conexión desde microservicio a MongoDB (ej: registro insertado)

- Tabla MySQL con datos reales

---

## 🔁 5. Comunicación entre Microservicios

- [x] `aggregator-service` consulta a `time`, `timezone`, `minutes`

- [x] `formatter-service` recibe datos del `aggregator`

📷 \*Capturas:\*

A screenshot of a computer

AI-generated content may be incorrect.

- Petición `curl` o Postman a `/aggregate` mostrando respuesta completa

- Logs de cada servicio registrando la comunicación

---

## 📬 6. Kafka y Eventos

- [x] Kafka funcionando (MSK o EC2)

- [x] Cada microservicio produce eventos

📷 \*Capturas:\*

- Topic `events.time.created` con mensajes en Kafka UI

- Evento JSON registrado por `aggregator-service`

- Diagrama de flujo de eventos (opcional)

---

## ☁️ 7. Elastic Load Balancer (ALB)

- [x] ALB configurado con routing por path

- [x] Target groups asociados a ASG

📷 \*Capturas:\*

- Lista de listeners en ALB

- Comprobación con DNS del ALB: `http://<ALB-DNS>/time`

---

## 🔁 8. Auto Scaling Groups

- [x] ASG por cada microservicio (min: 2 instancias)

- [x] Reglas de escalado configuradas

📷 \*Capturas:\*

- Configuración del ASG (min/max/desencadenante)

- Evidencia de escalado (antes y después)

---

## 📊 9. Observabilidad

- [x] Logs en CloudWatch de cada instancia

- [ ] (Opcional) Monitoreo con Prometheus o Grafana

- [x] Kafka UI operando

📷 \*Capturas:\*

- Logs en CloudWatch mostrando peticiones o errores

- Dashboard en Kafka UI con topics activos

---

## 📦 10. Docker y CI/CD (opcional)

- [x] Imágenes Docker subidas a Docker Hub o ECR

- [x] Scripts de despliegue automatizado

📷 \*Capturas:\*

- `docker push` al repositorio

- Pipeline ejecutándose (GitHub Actions, CodePipeline)

---

## ✅ Conclusión

La infraestructura distribuida cumple con los siguientes requisitos:

- 5 microservicios con persistencia doble (MongoDB + MySQL)

- Comunicación basada en eventos con Kafka

- Alta disponibilidad (EC2 x2 + ASG + ELB)

- Infraestructura reproducible con Terraform