ChurnInsight — Deploy no OCI (Opção 1)

VM + Docker Compose + OCIR (UI estática no Spring Boot)

Projeto	Hackathon NoCountry — ChurnInsight
Arquitetura	Opção 1 — VM + Docker Compose + OCIR (UI estática no Spring Boot)
Data	09/01/2026

Sumário

1. Visão executiva

Este documento descreve uma arquitetura de implantação de baixo custo no Oracle Cloud Infrastructure (OCI) para o projeto ChurnInsight, utilizando uma única Compute Instance (VM) executando Docker Compose. As imagens de aplicação são armazenadas no Oracle Container Registry (OCIR). A interface do usuário é servida como UI estática a partir do Spring Boot (java-api).

Objetivos desta opção (MVP / demo)

- Minimizar custo e complexidade operacional (uma única VM, sem OKE e sem Load Balancer).
- Isolar componentes internos (FastAPI e Postgres) sem exposição pública.
- Garantir um fluxo ponta a ponta: Browser -> Spring Boot (UI + API) -> FastAPI (inferência) -> Postgres (persistência).
- Padronizar build e entrega via imagens no OCIR (docker push/pull).

2. Visão geral da arquitetura

A arquitetura proposta organiza o deploy em três camadas: (i) estação de desenvolvimento para build e publicação das imagens, (ii) OCIR como repositório de imagens e (iii) OCI Compute (VM) executando a stack com Docker Compose em rede interna. O diagrama a seguir representa a disposição lógica e os limites de exposição pública.

Diagrama de arquitetura

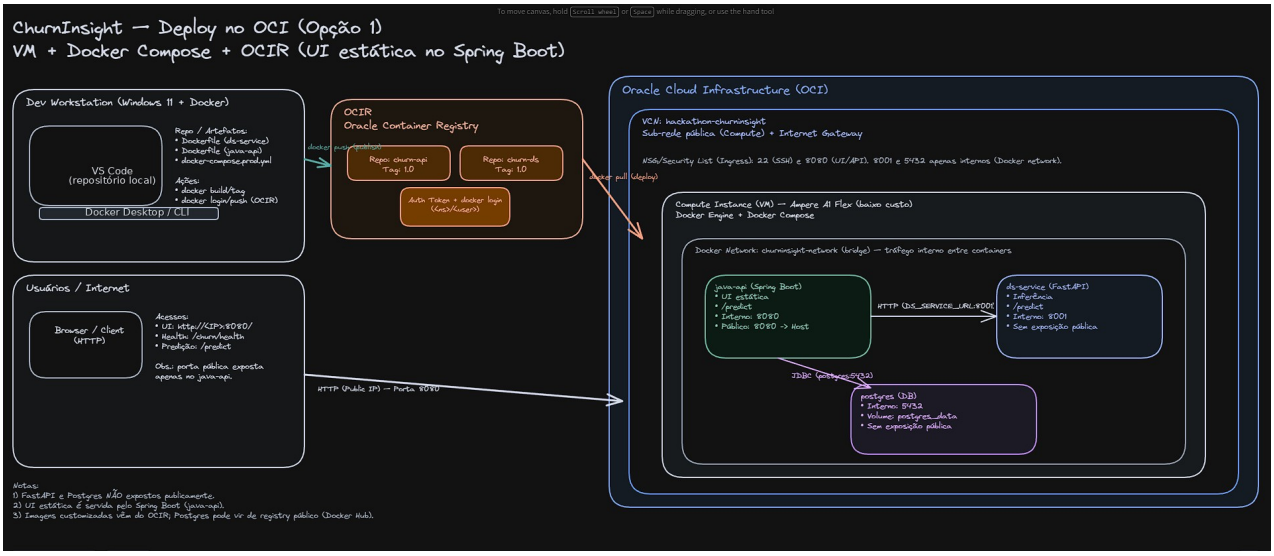

Figura 1 — churninsight_oci_opcao1 (Opção 1: VM + Docker Compose + OCIR)

3. Componentes e responsabilidades

A pilha em produção (VM) é composta por três containers, cada um com papel bem definido:

3.1 Matriz de serviços

Serviço	Função	Portas	Observações
java-api (Spring Boot)	UI estática + API gateway	8080 (público)	Chama ds-service via DS_SERVICE_URL; acessa Postgres via JDBC
ds-service (FastAPI)	Inferência do modelo (XGBoost serializado)	8001 (interno)	Não exposto publicamente; atende apenas chamadas do java-api
postgres (DB)	Persistência (dados e/ou auditoria)	5432 (interno)	Volume persistente postgres_data; sem exposição pública

4. Fluxo de requisição e contratos entre serviços

O fluxo de chamada é determinístico e evita dependências externas desnecessárias:

- O usuário acessa a UI estática no endpoint HTTP público do java-api: http://<IP_PUBLICO>:8080/.
- Ao acionar uma predição, o java-api recebe a requisição (ex.: POST /predict) e valida o payload.
- O java-api encaminha a requisição ao ds-service via HTTP interno (Docker network): DS_SERVICE_URL=http://ds-service:8001.
- O ds-service executa a inferência carregando o modelo a partir de MODEL_PATH (filesystem do container) e retorna a resposta.
- Opcionalmente, o java-api registra metadados e/ou resultados no Postgres via JDBC (postgres:5432).

4.1 Endpoints recomendados

Componente	Endpoint	Uso
java-api	GET /	Entrega da UI estática
java-api	POST /predict	Recebe payload e chama o ds-service
java-api	GET /churn/health	Healthcheck (validação e monitoramento)
ds-service	POST /predict	Inferência interna
ds-service	GET /health	Healthcheck interno (recomendado)

5. Rede e segurança

A exposição pública é intencionalmente mínima. Apenas a porta do java-api é publicada no host (VM). FastAPI e Postgres permanecem acessíveis apenas na rede interna do Docker.

5.1 Regras de entrada (Ingress) sugeridas — NSG/Security List

Porta	Origem	Destino	Finalidade

22/TCP	Seu IP (recomendado)	VM	Acesso SSH administrativo
8080/TCP	Internet	VM	UI + API (java-api)
8001/TCP	N/A (não expor)	VM	Acesso interno apenas via Docker network
5432/TCP	N/A (não expor)	VM	Acesso interno apenas via Docker network

Recomendação: restringir 22/TCP por CIDR (seu IP) e manter 8080/TCP aberta apenas durante a fase de demo. Em ambientes mais maduros, recomenda-se adicionar reverse proxy (Nginx) e expor apenas 80/443 com TLS.

6. Pipeline de build e publicação (Dev -> OCIR -> VM)

O ciclo de entrega é baseado em imagens Docker versionadas. A estação de desenvolvimento faz build e push para o OCIR. A VM autentica no OCIR e faz pull das mesmas imagens para subir a stack com Docker Compose.

6.1 Artefatos publicados no OCIR

Repositório	Imagem	Tag recomendada	Origem no repositório
churn-api	<region>.ocir.io/<ns>/churn-api	1.0 (ou semver)	back-end/churn (Dockerfile do Spring Boot)
churn-ds	<region>.ocir.io/<ns>/churn-ds	1.0 (ou semver)	ds_service (Dockerfile do FastAPI)

6.2 Exemplo de docker-compose.prod.yml (produção)

```
version: "3.9"
services:
  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: churninsight_db
      POSTGRES_USER: churnuser
      POSTGRES_PASSWORD: churnpass123
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - churninsight-network

  ds-service:
    image: <region>.ocir.io/<ns>/churn-ds:1.0
    environment:
      HOST: 0.0.0.0
      PORT: 8001
      MODEL_PATH: /app/models/modelo_churn_final.pkl
    expose:
      - "8001"
```

```
    networks:
      - churninsight-network
    restart: unless-stopped

  java-api:
    image: <region>.ocir.io/<ns>/churn-api:1.0
    environment:
      DATABASE_HOST: postgres:5432
      DATABASE_NAME: churninsight_db
      DATABASE_USER: churnuser
      DATABASE_PASSWORD: churnpass123
      DS_SERVICE_URL: http://ds-service:8001
    ports:
      - "8080:8080"
    depends_on:
      - postgres
      - ds-service
    networks:
      - churninsight-network
    restart: unless-stopped

volumes:
  postgres_data:

networks:
  churninsight-network:
    driver: bridge
```

7. Procedimento de implantação na VM (passo a passo)

7.1 Preparar o OCIR

- Identificar tenancy namespace e região do OCIR.
- Criar repositórios churn-api e churn-ds.
- Gerar Auth Token para docker login.

7.2 Build e push das imagens (Dev)

- Autenticar no OCIR (docker login).
- Fazer build/tag das imagens.
- Publicar com docker push.

7.3 Provisionar a VM no OCI

- Shape recomendado: Ampere A1 Flex (1 OCPU, 6 GB).
- Criar VCN/subnet pública e NSG com 22 e 8080.
- Não expor 5432 e 8001 publicamente.

7.4 Instalar Docker + Compose na VM

- Instalar Docker Engine (Oracle Linux/Ubuntu).
- Habilitar e iniciar Docker.
- Validar Docker Compose e permissões do usuário.

7.5 Subir a stack em produção

- Realizar docker login no OCIR na VM.
- Copiar docker-compose.prod.yml (e opcional .env) para a VM.
- Executar docker compose pull e docker compose up -d.
- Validar UI e healthchecks.

7.6 Comandos de referência (build/push/deploy)

```
# Login no OCIR (use Auth Token)
docker login <region>.ocir.io -u "<namespace>/<user>" -p "<auth_token>"

# Build/Push (no Dev)
docker build -t <region>.ocir.io/<ns>/churn-ds:1.0 ./ds_service
docker build -t <region>.ocir.io/<ns>/churn-api:1.0 ./back-end/churn
docker push <region>.ocir.io/<ns>/churn-ds:1.0
docker push <region>.ocir.io/<ns>/churn-api:1.0

# Deploy (na VM)
docker compose -f docker-compose.prod.yml pull
docker compose -f docker-compose.prod.yml up -d
docker compose -f docker-compose.prod.yml logs -f --tail=200
```

8. Operação, observabilidade e controle de custos

Por se tratar de uma arquitetura de MVP em VM única, a operação deve priorizar simplicidade e contenção de custos.

8.1 Checklist operacional

Tópico	Recomendação prática
Logs	Usar docker compose logs; opcionalmente configurar OCI Logging/Agent para forward.
Healthchecks	Garantir /churn/health no java-api e /health no ds-service; usar para smoke tests.
Alertas	Definir Budget e alertas (por exemplo, 50% e 80% do crédito).
Custos	Desligar a VM quando não estiver usando; manter boot volume no mínimo necessário.
Backups	Se houver dados relevantes, considerar backup do volume do Postgres (dump periódico).

9. Limitações e próximos incrementos

Esta opção é adequada para demonstrações e MVPs, porém não oferece alta disponibilidade. As principais limitações e melhorias incrementais recomendadas são:

- Ponto único de falha: a VM concentra todos os serviços (sem HA).
- Escala manual: aumento de carga exige ajuste de shape/recursos ou migração para Container Instances/OKE.
- Segurança/TLS: em produção real, adicionar reverse proxy e TLS (80/443), além de hardening de SSH e atualização de patches.
- Observabilidade avançada: métricas, traces e centralização de logs (OCI Logging + dashboards).

Apêndice A — Variáveis de ambiente (referência)

A.1 Variáveis por serviço

Serviço	Variável	Exemplo	Descrição
java-api	DS_SERVICE_URL	http://ds-service:8001	URL interna do FastAPI no Docker network
java-api	DATABASE_HOST	postgres:5432	Host/porta do Postgres no Docker network
java-api	DATABASE_NAME	churninsight_db	Nome do banco
java-api	DATABASE_USER	churnuser	Usuário do banco
java-api	DATABASE_PASSWORD	*****	Senha do banco
ds-service	MODEL_PATH	/app/models/ modelo_churn_final.pkl	Caminho do modelo serializado dentro do container
ds-service	HOST	0.0.0.0	Bind do servidor (container)
ds-service	PORT	8001	Porta interna do FastAPI