Gmail **Fillipe Guerra <fillipe.backup@gmail.com>**

IA_Autonoma_Parte16

1 mensagem

Fillipe Guerra <fillipe.backup@gmail.com> 28 de outubro de 2025 às 11:41
Para: Fillipe Augusto Gomes Guerra <fillipe182@hotmail.com>, Fillipe Guerra <fillipe.backup@gmail.com>

Segue o **lote** com: **(A) Playbooks de desastre (backup/restore de Postgres + modelos LoRA/adapters)**, **(B) Autoscale (Kubernetes via HPA/KEDA)** e **(C) Alertas (Prometheus Alertmanager + exemplo Slack)**. Tudo pronto pra colar. Se algum arquivo já existir, **substitui** (são supersets compatíveis).

A) Playbooks de Desastre — Backup & Restore

A.1 Estrutura de backup

```
ops/
└── backup/
    ├── backup.sh
    ├── restore.sh
    ├── prune.sh
    ├── .env.backup.example
    └── README.md
```

A.2 `.env.backup.example`

```
# Postgres
PGHOST=localhost
PGPORT=5432
PGUSER=aion
PGPASSWORD=aion
PGDATABASE=aion

# Destino local
BACKUP_DIR=./ops/backup/artifacts

# Retenção
RETENTION_DAYS=14

# (Opcional) S3
S3_BUCKET=s3://aion-backups/prod
AWS_ACCESS_KEY_ID=SEU_ACCESS
AWS_SECRET_ACCESS_KEY=SEU_SECRET
AWS_DEFAULT_REGION=sa-east-1
```

A.3 `backup.sh`

```
#!/usr/bin/env bash
set -euo pipefail
source "$(dirname "$0")/.env.backup" 2>/dev/null || true

TS="$(date +%Y%m%d-%H%M%S)"
mkdir -p "${BACKUP_DIR:-./ops/backup/artifacts}"

echo "[AION] Dumpando Postgres..."
pg_dump -h "${PGHOST:-localhost}" -p "${PGPORT:-5432}" -U "${PGUSER:-aion}" -F c -d "${PGDATABASE:-
aion}" \
  -f "${BACKUP_DIR}/pg_${TS}.dump"

echo "[AION] Empacotando adapters LoRA..."
```

```
tar czf "${BACKUP_DIR}/adapters_${TS}.tar.gz" -C ./trainer/lora out || echo "[WARN] Nada em
trainer/lora/out"

echo "[AION] Empacotando KB (opcional: docs & vetores)..."
tar czf "${BACKUP_DIR}/kb_${TS}.tar.gz" -C server/aion/data kb || echo "[WARN] Nada em
server/aion/data/kb"

if [ -n "${S3_BUCKET:-}" ]; then
  echo "[AION] Subindo para S3..."
  aws s3 cp "${BACKUP_DIR}/pg_${TS}.dump" "${S3_BUCKET}/pg_${TS}.dump"
  aws s3 cp "${BACKUP_DIR}/adapters_${TS}.tar.gz" "${S3_BUCKET}/adapters_${TS}.tar.gz"
  aws s3 cp "${BACKUP_DIR}/kb_${TS}.tar.gz" "${S3_BUCKET}/kb_${TS}.tar.gz"
fi

echo "[AION] Backup concluído: ${TS}"
```

A.4 `restore.sh`

```
#!/usr/bin/env bash
set -euo pipefail
source "$(dirname "$0")/.env.backup" 2>/dev/null || true

DUMP_FILE="${1:-}"
ADAPTERS_TGZ="${2:-}"
KB_TGZ="${3:-}"

[ -z "$DUMP_FILE" ] && echo "Uso: restore.sh <pg_dump_file> [adapters.tgz] [kb.tgz]" && exit 1

echo "[AION] Restaurando Postgres de ${DUMP_FILE}..."
pg_restore -h "${PGHOST:-localhost}" -p "${PGPORT:-5432}" -U "${PGUSER:-aion}" -d "${PGDATABASE:-
aion}" --clean --if-exists "$DUMP_FILE"

if [ -n "${ADAPTERS_TGZ:-}" ] && [ -f "$ADAPTERS_TGZ" ]; then
  echo "[AION] Restaurando adapters..."
  mkdir -p trainer/lora/out
  tar xzf "$ADAPTERS_TGZ" -C ./trainer/lora
fi

if [ -n "${KB_TGZ:-}" ] && [ -f "$KB_TGZ" ]; then
  echo "[AION] Restaurando KB..."
  mkdir -p server/aion/data
  tar xzf "$KB_TGZ" -C server/aion/data
fi

echo "[AION] Restore concluído."
```

A.5 `prune.sh`

```
#!/usr/bin/env bash
set -euo pipefail
source "$(dirname "$0")/.env.backup" 2>/dev/null || true

find "${BACKUP_DIR:-./ops/backup/artifacts}" -type f -mtime +"${RETENTION_DAYS:-14}" -print -delete ||
true

if [ -n "${S3_BUCKET:-}" ]; then
  echo "[AION] (Opcional) Rotinas de lifecycle no S3 devem ser configuradas no bucket."
fi
```

A.6 `README.md` (resumo rápido)

```
# Backup/Restore AION

- Rodar backup local:
  ./ops/backup/backup.sh

- Enviar a S3 (preencha .env.backup):
```

```
  ./ops/backup/backup.sh

- Prune local (retenção em dias):
  ./ops/backup/prune.sh

- Restaurar:
  ./ops/backup/restore.sh ./ops/backup/artifacts/pg_YYYYMMDD-HHMMSS.dump \
                          ./ops/backup/artifacts/adapters_YYYYMMDD-HHMMSS.tar.gz \
                          ./ops/backup/artifacts/kb_YYYYMMDD-HHMMSS.tar.gz
```

Agende no compose (cron do host) ou no próprio app usando os crons que já criamos. Ex.: rodar `backup.sh` diariamente 02:30.

B) Autoscale — Kubernetes (HPA + KEDA)

Para **produção escalável**, recomendo K8s. Abaixo, manifests básicos para **app (Node)** e **inferência (FastAPI)** com **HPA** e **KEDA** (fila/cron). Use **Prometheus Adapter** se quiser escalar por métricas customizadas.

B.1 Namespace & ConfigMaps

`k8s/00-namespace.yaml`

```
apiVersion: v1
kind: Namespace
metadata:
  name: aion
```

`k8s/01-config.yaml`

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aion-config
  namespace: aion
data:
  AION_LOCAL_LLM_URL: "http://inference:8008"
```

B.2 App Node (Deployment + Service + HPA)

`k8s/10-app.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app
  namespace: aion
spec:
  replicas: 2
  selector: { matchLabels: { app: app } }
  template:
    metadata: { labels: { app: app } }
    spec:
      containers:
        - name: app
          image: your-registry/aion-app:latest
          ports: [{ containerPort: 3000 }]
          envFrom:
            - configMapRef: { name: aion-config }
          env:
            - name: NODE_ENV
              value: "production"
            - name: DATABASE_URL
              valueFrom:
                secretKeyRef:
                  name: aion-secrets
```

```
                  key: DATABASE_URL
            resources:
              requests: { cpu: "200m", memory: "512Mi" }
              limits: { cpu: "1", memory: "1Gi" }
            livenessProbe:
              httpGet: { path: /health, port: 3000 }
              initialDelaySeconds: 15
              periodSeconds: 10
            readinessProbe:
              httpGet: { path: /health, port: 3000 }
              initialDelaySeconds: 5
              periodSeconds: 10
---
apiVersion: v1
kind: Service
metadata:
  name: app
  namespace: aion
spec:
  selector: { app: app }
  ports:
    - name: http
      port: 3000
      targetPort: 3000
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: app-hpa
  namespace: aion
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app
  minReplicas: 2
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 60
```

B.3 Inferência (Deployment + Service + HPA com GPU opcional)

```
k8s/20-inference.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: inference
  namespace: aion
spec:
  replicas: 1
  selector: { matchLabels: { app: inference } }
  template:
    metadata: { labels: { app: inference } }
    spec:
      containers:
        - name: inference
          image: your-registry/aion-inference:gpu   # ou :cpu
          ports: [{ containerPort: 8008 }]
          envFrom:
            - configMapRef: { name: aion-config }
          env:
            - name: AION_BASE_MODEL
              value: mistralai/Mistral-7B-Instruct-v0.3
```

```
            - name: AION_LOAD_IN_4BIT
              value: "true"
          resources:
            requests: { cpu: "500m", memory: "6Gi" }
            limits:   { cpu: "2",    memory: "12Gi" }
          # Para GPU (NVIDIA):
          # resources:
          #   limits:
          #     nvidia.com/gpu: 1
          livenessProbe:
            httpGet: { path: /health, port: 8008 }
            initialDelaySeconds: 30
            periodSeconds: 15
          readinessProbe:
            httpGet: { path: /health, port: 8008 }
            initialDelaySeconds: 15
            periodSeconds: 10
---
apiVersion: v1
kind: Service
metadata:
  name: inference
  namespace: aion
spec:
  selector: { app: inference }
  ports:
    - name: http
      port: 8008
      targetPort: 8008
---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: inference-hpa
  namespace: aion
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: inference
  minReplicas: 1
  maxReplicas: 4
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

B.4 (Opcional) KEDA para jobs de treino (cron/queue)

k8s/30-keda-train.yaml

```
apiVersion: keda.sh/v1alpha1
kind: ScaledJob
metadata:
  name: aion-train-lora
  namespace: aion
spec:
  jobTargetRef:
    template:
      spec:
        template:
          spec:
            containers:
              - name: trainer
                image: your-registry/aion-trainer:latest
                command: ["bash","-lc","pnpm ts-node server/training/cli.start-lora.ts"]
```

```
              envFrom:
                - configMapRef: { name: aion-config }
            restartPolicy: Never
  triggers:
    - type: cron
      metadata:
        timezone: America/Sao_Paulo
        start: "0 2 * * *"   # 02:00 todos os dias
        end: "0 3 * * *"
        desiredReplicas: "1"
```

Assim, o **treino LoRA** pode ser escalado sob demanda, sem travar o app.

C) Alertas — Prometheus Alertmanager (Slack)

C.1 Alertmanager (compose)

Adicione no `docker-compose.yml`:

```
  alertmanager:
    image: prom/alertmanager:v0.27.0
    volumes:
      - ./ops/alertmanager/alertmanager.yml:/etc/alertmanager/alertmanager.yml:ro
    ports: ["9093:9093"]
    restart: unless-stopped
    depends_on: [prometheus]
```

C.2 Prometheus — regras de alerta

Crie `ops/prometheus/alerts.yml` e **inclua** no `prometheus.yml`:

`ops/prometheus/prometheus.yml` (trecho extra no final):

```
rule_files:
  - /etc/prometheus/alerts.yml
```

`ops/prometheus/alerts.yml`

```
groups:
- name: aion-alerts
  rules:
  - alert: HighFallbackRate
    expr: sum(rate(aion_fallback_total[5m])) / sum(rate(aion_answers_total[5m])) > 0.25
    for: 10m
    labels: { severity: page }
    annotations:
      summary: "Fallback elevado"
      description: "Taxa de fallback > 25% por > 10m"

  - alert: LowNDCGEst
    expr: avg_over_time(aion_answers_total[30m]) > 0 and (sum(rate(aion_fallback_total[30m])) /
sum(rate(aion_answers_total[30m])) > 0.2)
    for: 30m
    labels: { severity: warn }
    annotations:
      summary: "Provável queda de nDCG (proxy)"
      description: "Fallback consistente sugere perda de relevância."

  - alert: AppDown
    expr: up{job="aion-app"} == 0
    for: 1m
    labels: { severity: page }
    annotations:
      summary: "App fora do ar"
      description: "Instância do app não responde para Prometheus."
```

```
    - alert: InferenceDown
      expr: up{job="aion-inference"} == 0
      for: 1m
      labels: { severity: page }
      annotations:
        summary: "Inferência fora do ar"
        description: "Microserviço LLM local indisponível."
```

C.3 Alertmanager — Slack webhook

`ops/alertmanager/alertmanager.yml`

```
global:
  resolve_timeout: 5m

route:
  receiver: "slack"
  routes:
    - matchers:
        - severity="page"
      receiver: "slack"

receivers:
  - name: "slack"
    slack_configs:
      - api_url: "https://hooks.slack.com/services/SEU/WEBHOOK/ID"
        channel: "#aion-alertas"
        send_resolved: true
        title: "{{ .Status | toUpper }}: {{ .CommonAnnotations.summary }}"
        text: |
          {{ range .Alerts }}
          *Alert:* {{ .Labels.alertname }}
          *Severity:* {{ .Labels.severity }}
          *Descr:* {{ .Annotations.description }}
          *StartsAt:* {{ .StartsAt }}
          {{ end }}
```

Troque o webhook/canal. Se preferir, use **Grafana Alerting** com UI.

D) Mini Runbook de Incidente (resumo prático)

1. **Sintoma**: queda de respostas / aumento de fallback → veja **Grafana** (AION Overview).
2. **Checar saúde**:
   - `GET /health` do app e do microserviço (`/health` porta 8008).
   - `docker compose ps` ou `kubectl get pods -n aion`.
3. **Ver custos** (telemetria Fase 5) e **FR**; se alto:
   - Executar **"Iniciar LoRA"** no painel ou `POST /api/ai/train/lora/start`.
   - Confirmar dataset via **export JSONL**.
4. **Degradação da KB**:
   - Acessar **/admin/learning** → rodar batch discovery → **curadoria** → aprovar para KB.
   - Rodar `ann.rebuild()` se o painel já não acionar automaticamente após ingest.
5. **Rollbacks**:
   - Restaurar **adapters** (último `adapters_*.tar.gz`) e, se necessário, **pg_dump**.
   - Reaplicar `.env`/secrets.
6. **Pós-mortem**:

- Telemetria (nDCG/MRR/CTR/CR) vs. janela do incidente.
- Ajustar `τ_fallback`, `τ_ndcg`, e **cooldown** do watcher.

E) Como colocar tudo pra rodar (resumo)

1. **Backups**: preencha `ops/backup/.env.backup`, rode `backup.sh` e agende (cron).
2. **Alertas**: adicione Alertmanager + `alerts.yml`, reinicie Prometheus/Alertmanager.
3. **K8s (se for usar)**: aplique `k8s/*.yaml` → `kubectl apply -f k8s -n aion`.
4. **Compose**: `docker compose up -d --build`.
5. **Grafana**: confirme data source e dashboards carregados.
6. **Teste SSE** e **fallback** como já te mostrei.