Inteligência Artificial 2

Aula 29: Hackathon do Projeto Final (Cloud e Deploy)

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Inteligência Artificial 2

Sumário

Onde Paramos?

Ontem vocês construíram o **rosto do produto**:

- Interface HTML/React consumindo a API via `fetch()`.
- CORS resolvido, indicador de carregamento, balões de chat.
- Teste de usabilidade: 10 cliques $\rightarrow$ sem crashes.

A Missão de Hoje

"Na minha máquina funciona" $\neq$ produto real.
Hoje o produto vai para a **nuvem** com URL pública.

Objetivos do Dia

Ao final desta sessão, a equipe deve ter:

1. **Deploy** funcional na nuvem (Render, Railway, Vercel ou similar).
2. **Variáveis de ambiente** configuradas no painel da plataforma.
3. **URL pública** acessível por qualquer navegador.
4. **Pitch** esboçado: Dor $\rightarrow$ Solução $\rightarrow$ Demo $\rightarrow$ Engenharia.
5. **Ensaio** da demo ao vivo na URL pública.

O Fim do Localhost

- Vocês construíram IAs que analisam dados corporativos, extraem *insights* e possuem uma interface reativa.
- Porém, todo o projeto está ancorado no *hardware* do laboratório.
- **A Realidade Comercial:** Investidores, professores e clientes não instalam Python e Git para testar o seu produto. A validação do *Software* depende de um *Link* (URL Pública).
- O passo final que diferencia o nível acadêmico do nível profissional é o **Deploy na Nuvem**.

O Paradoxo do "Na Minha Máquina Funciona"

Sistemas de Inteligência Artificial sofrem fortemente com variações de ambiente:

- A máquina local possui bibliotecas de C++ específicas instaladas globalmente (Necessárias para compilar `hnswlib` do ChromaDB, por exemplo).
- O seu Windows trata caminhos de arquivo como `C:\pasta`, enquanto o servidor Linux (Nuvem) usa `/pasta`. O seu *Text Loader* pode falhar silenciosamente no Deploy.
- O fuso horário (*Timezone*) do servidor geralmente é UTC, fazendo suas rotas de histórico de mensagens SQL ficarem com horas defasadas.

Virtualização Mínima: Containers

Para garantir que o código funcione no Servidor da Amazon ou Google exatamente como roda no seu Laptop, utilizamos tecnologias de *Container*.

- Um **Container (Docker)** é uma "bolha" isolada que empacota o seu Código, a Linguagem (Python 3.10), as Dependências (`requirements.txt`) e as variáveis do Sistema Operacional Ubuntu de forma ultraleve (sem interface gráfica).
- Garante a reprodutibilidade universal: *Se o container compilou na sua máquina, ele funcionará em qualquer nuvem do planeta.*

Anatomia de um Dockerfile (Backend de IA)

As plataformas modernas de *Deploy* (Render, Railway) criam isso por debaixo dos panos, mas é vital entender a sequência de execução:

```
# 1. Baixar o Sistema Operacional com Python embute
FROM python:3.11-slim

# 2. Definir pasta de trabalho
WORKDIR /app

# 3. Copiar a lista e instalar LangChain, FastAPI, etc.
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 4. Copiar o codigo da IA
COPY . .

# 5. Ligar a porta da API Web
CMD [‘‘uvicorn’’, ‘‘main:app’’, ‘‘--host’’, ‘‘0.0.0.0’’, ‘‘--port’’,
```

PaaS: Platform as a Service

A maneira mais rápida para Startups pequenas (Hackathons) fazerem deploy do *Backend* é utilizando provedores *PaaS* como **Render**, **Railway** ou **Heroku**. **Como funciona?**

1. Você vincula a plataforma ao seu Repositório **GitHub**.
2. Ao fazer o comando `git push`, o servidor percebe a mudança (*Webhook*).
3. A plataforma clona seu código, monta o Docker, resolve os conflitos e levanta o servidor HTTP automaticamente sem que você configure IPs ou Firewalls de Rede.

Edge/Serverless: O Padrão do Frontend

Para a **Interface de Usuário** (o código React, HTML/JS, Next.js), utilizamos provedores de Edge Computing como **Vercel** ou **Netlify**.

- **Serverless:** O servidor não fica "ligado" ocioso gastando dinheiro. Apenas quando o cliente acessa o site, um micro-servidor é acordado em milissegundos para entregar o HTML.
- **Global Edge:** Seu site de IA é distribuído em nós ao redor do planeta. O usuário do Japão acessa o seu Frontend a partir do servidor fisicamente no Japão (Latência mínima).

O Problema do Cold Start em Inteligência Artificial

Pode-se rodar o Backend de IA inteiro em Serverless (Vercel Functions)? Geralmente, **NÃO**.

- O *Backend* pesado (FastAPI/LangChain) carrega modelos de *Embeddings* locais que pesam centenas de Megabytes na RAM inicial.
- Se o servidor estiver "dormindo" (*Serverless*) e for acordado, ele precisará puxar as bibliotecas (TensorFlow/PyTorch) para a memória (Processo chamado de **Cold Start**).
- Isso fará com que a primeira pergunta do dia leve de 15 a 30 segundos, causando falha (*Timeout*) no navegador. **Mantenha APIs de IA em servidores contínuos (Render).**

O Sistema de Arquivos Efêmero (Ephemeral Filesystem)

Se a sua equipe construiu o RAG utilizando o **ChromaDB** apontando para uma pasta local (`./chroma_db`), o Deploy tem 90% de chance de falhar no segundo dia. **Por quê?**

- Plataformas *PaaS* tratam os servidores como descartáveis. A cada *Deploy* (cada push no Github) ou reinicialização diária, o disco rígido é **apagado** e reconstruído "zerado".
- Seu banco de vetores que continha os PDFs da empresa desaparecerá, e a IA sofrerá uma amnésia irreversível em produção.

A Migração Compulsória para o Cloud DB

A arquitetura moderna obriga que os dados (*State*) saiam do disco do Backend e vão para um serviço autônomo (Isolamento de Estado).

Banco Relacional (Usuários/Chat)

- Abandonem o `SQLite` local.
- Criem uma conta gratuita na *Supabase*, *NeonDB* ou *Render PostgreSQL*.
- Atualizem a String de Conexão no Python/Java.

Banco Vetorial (Conhecimento)

- Abandonem o `Chroma Local`.
- Migrem a coleção para APIs Serverless: *Pinecone* ou *Qdrant Cloud*.
- Re-executem o script de Ingestão para repopular o banco na nuvem.

Gestão de Segredos: O Maior Risco de IA

As chaves (`OPENAI_API_KEY`) são dinheiro puro. Elas cobram por uso de *Token*.

- **A Falha Amadora:** Commitar (enviar via Git) o arquivo `.env` ou colar a chave de texto "sk-..." diretamente no código fonte. Em minutos, bots varrem o GitHub público e esgotam sua fatura na OpenAI.
- **A Solução em Nuvem (Environment Variables):** No painel do Render ou Vercel, existe a aba *"Variables/Secrets"*. Você copia e cola a chave lá. No seu código Python, você mantém a chamada genérica: `os.getenv(''OPENAI_API_KEY'')`. O Sistema Operacional da nuvem injeta a chave blindada na memória RAM, sem gravá-la em disco.

Proteção contra Ataques de Custos (Rate Limiting)

Uma vez que sua URL de Frontend está pública, qualquer um pode acessá-la.

- E se um bot começar a enviar mil requisições por minuto para a sua API, gerando textos enormes? A sua conta da OpenAI irá à falência.
- Para evitar ataques de *Denial of Wallet* (Negação de Carteira), é obrigatório implementar **Rate Limiting** no Backend (Ex: max 5 requests/minuto por IP).
- Startups avançadas usam *Cloudflare* na frente do domínio para bloquear tráfego malicioso antes que chegue à API.

Infraestrutura como Código (IaC)

Clicar em botões na Vercel ou Render é ótimo para Hackathons. Para grandes empresas, é inaceitável.

- Se o funcionário que criou a conta for demitido, como recriar os mesmos 50 servidores com as mesmas chaves de API?
- **IaC (Terraform / Pulumi):** A infraestrutura é escrita em código (arquivos HCL).
- Você roda `terraform apply` e ele fala com a AWS/Google Cloud para criar os Bancos de Dados e os Servidores automaticamente. O servidor é tratado como "gado", não como "animal de estimação".

Serverless GPU: Rodando Modelos Abertos (Llama 3)

E se vocês não quiserem pagar a API da OpenAI, mas sim rodar o Llama 3 70B?

- O Llama 70B exige cerca de 40GB de VRAM. Uma placa A100 custa milhares de dólares.
- **Solução Serverless GPU (RunPod / Modal / Replicate):**
- Você aluga a GPU *apenas* pelos segundos que ela gasta inferindo os *Tokens*. O provedor liga a Placa de Vídeo, roda seu contêiner Docker com o Llama, devolve a reposta e desliga a GPU, cobrando centavos por invocação.

Auto-Scaling e Kubernetes (K8s)

O que acontece com a sua API de IA se o aplicativo viralizar e passar de 10 para 100.000 usuários em uma hora?

- **Kubernetes:** Um orquestrador de contêineres. Se a CPU do Backend FastAPI chegar a 80%, o Kubernetes clona seu *Docker* em um novo servidor (*Pod*) instantaneamente.
- Um *Load Balancer* roteará metade das perguntas para o Pod A e metade para o Pod B.
- **Cuidado:** Se a API não for *Stateless* (não armazenar o Histórico de Chat no disco local, mas sim no SQL Cloud), o escalonamento automático irá falhar e misturar as conversas dos usuários.

Disaster Recovery em Bancos Vetoriais

Toda nuvem quebra eventualmente. O *Pinecone* pode sair do ar por horas.

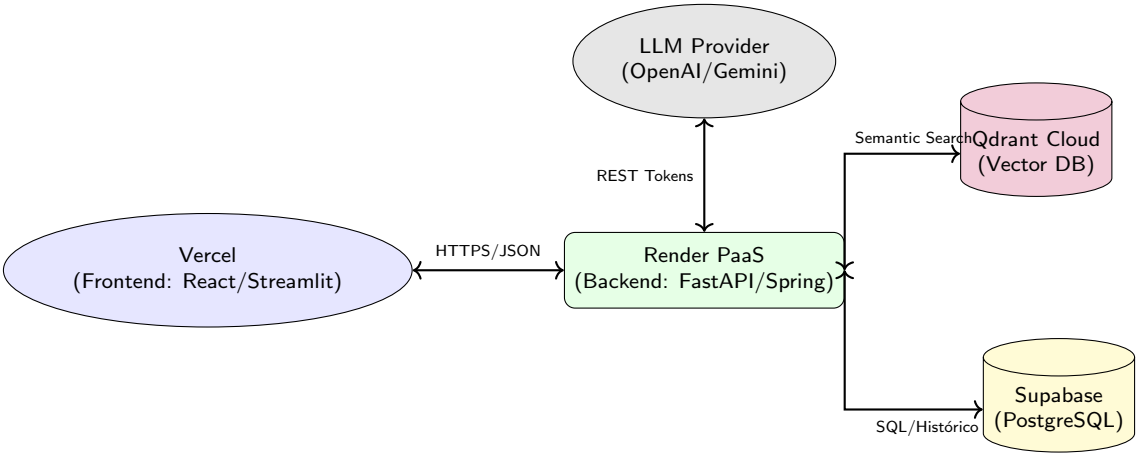
- **O Risco:** Sem o banco de vetores (Contexto), sua IA inteira fica cega. O Sistema cai.
- **Disaster Recovery (DR):** Empresas maduras mantêm cópias arquivadas (*Snapshots*) do banco vetorial no Amazon S3.
- Além disso, o script de Ingestão original (PDF $\rightarrow$ Embeddings) deve estar guardado no Git, permitindo reconstruir o banco do zero num provedor concorrente (ex: Qdrant) em questão de minutos, caso o primário decrete falência.

Integração e Entrega Contínua (CI/CD)

A automação da entrega (CD) muda o fluxo de trabalho da Startup:

1. O Dev altera o Prompt de Sistema localmente e testa.
2. O Dev comita a alteração (`git push`).
3. O GitHub alerta o servidor *PaaS*.
4. O Servidor compila o código novo. Se houver erro de sintaxe, ele **cancela** a substituição e mantém a versão velha no ar.
5. Se for bem sucedido, o servidor vira a chave. A URL agora roda a versão nova, sem o usuário sentir queda do serviço (*Zero-Downtime Deploy*).

Diagrama da Arquitetura Distribuída Final



Observabilidade (Logs de Nuvem)

"A aplicação retorna 500 Internal Server Error na nuvem, mas no meu PC funciona." O diagnóstico não é feito por adivinhação.

- Abram a aba **Logs** ou **Console** do provedor (Render/Vercel).
- Lá aparecerá o terminal do Linux operando. Você verá o *Traceback* real (ex: "KeyError: 'OPENAI_API_KEY"' ou "Pinecone Connection Refused").
- Consertem no código local, façam novo Push e aguardem os novos Logs.

A Dinâmica do Demoday

A próxima aula consistirá nas apresentações das Startups (*Pitchs*). O que os avaliadores olharão?

1. **A Dor:** O problema real que tentam resolver (O domínio do negócio).
2. **O Produto:** A plataforma acessível via URL (Testada ao vivo sem rodar na máquina do professor).
3. **A Arquitetura:** Quais as tecnologias utilizadas no RAG? Qual o LLM base e o Modelo de Embedding?
4. **Proteção (Guardrails):** A IA consegue ser corrompida a falar de assuntos proibidos durante a apresentação?

Laboratório 29: A Nuvem Pública

Meta Crítica do Dia:

1. Criem as contas nos provedores PaaS (*Render, Railway, Vercel*).
2. Migrem os bancos locais (Chroma/SQLite) para alternativas em nuvem (Pinecone/Supabase).
3. Subam os repositórios Git, injetem as senhas (*Environment Variables*) no painel e desencadeiem o *Deploy*.
4. Monitorem a aba *Logs*. Vejam os erros do Docker em tempo real.
5. Finalizem com a URL pública operando pelo celular dos membros da equipe via 4G.

Dúvidas?

Email: alessio@cefetmg.br
Aula: Segunda e Quarta-Feira, 7h as 8:40h (Sala 39)
WhatsApp: Dúvidas rápidas

Lembrete

Confira sua Issue no GitLab!

Próxima aula: Ensaio do Demoday

100% da Aula 29 Concluída!