Minicurso: REST APIs com FastAPI, Microservices, Docker e Kubernetes

Introdução Completa em 60 Minutos

Professor Dr. Bento Rafael Siqueira

UFLA - Universidade Federal de Lavras

2 de outubro de 2025

Agenda

1. Introdução
2. REST APIs
3. FastAPI
4. Microservices
5. Docker
6. Kubernetes
7. Demonstração Prática
8. Próximos Passos

O que vamos aprender hoje?

Conceitos Fundamentais:

- REST APIs
- FastAPI Framework
- Microservices
- Containerização
- Orquestração

Tecnologias:

- Python 3.8+
- FastAPI
- Docker
- Kubernetes
- Docker Compose

Objetivo

Ao final desta aula, você será capaz de criar, containerizar e orquestrar uma API REST completa!

O que é uma REST API?

REST - Representational State Transfer

- Arquitetura de software para sistemas distribuídos
- Usa HTTP como protocolo de comunicação
- Baseada em recursos (Resources) identificados por URLs
- Operações através de métodos HTTP (GET, POST, PUT, DELETE)

Características Principais

- **Stateless**: Cada requisição é independente
- **Client-Server**: Separação clara de responsabilidades
- **Cacheable**: Respostas podem ser armazenadas em cache
- **Uniform Interface**: Interface consistente

Métodos HTTP e seus significados

Método	Operação	Descrição
GET	Leitura	Recuperar dados
POST	Criação	Criar novo recurso
PUT	Atualização	Atualizar recurso completo
PATCH	Atualização	Atualizar parte do recurso
DELETE	Exclusão	Remover recurso

Exemplo de URLs REST

- `GET /api/users` - Listar todos os usuários
- `GET /api/users/123` - Obter usuário específico
- `POST /api/users` - Criar novo usuário
- `PUT /api/users/123` - Atualizar usuário
- `DELETE /api/users/123` - Deletar usuário

Por que FastAPI?

Vantagens:

- **Performance**: Uma das APIs mais rápidas do Python
- **Documentação Automática**: Swagger/OpenAPI
- **Type Hints**: Validação automática
- **Async/Await**: Suporte nativo
- **Fácil de usar**: Sintaxe simples

Comparação de Performance

- FastAPI: 60,000 req/s
- Flask: 20,000 req/s
- Django: 15,000 req/s

Primeira API com FastAPI

```python
from fastapi import FastAPI
from pydantic import BaseModel
from typing import List

app = FastAPI(title="Minha API", version="1.0.0")

# Modelo de dados
class User(BaseModel):
    id: int
    name: str
    email: str

# Dados em memoria (para demonstracao)
users = [
    User(id=1, name="Joao", email="joao@email.com"),
    User(id=2, name="Maria", email="maria@email.com")
]

@app.get("/")
async def root():
    return {"message": "Hello World!"}

@app.get("/users", response_model=List[User])
async def get_users():
    return users

@app.get("/users/{user_id}", response_model=User)
async def get_user(user_id: int):
    for user in users:
        if user.id == user_id:
```

Adicionando mais funcionalidades

```python
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
from typing import List, Optional

# Modelo para criacao de usuario
class UserCreate(BaseModel):
    name: str
    email: str

@app.post("/users", response_model=User)
async def create_user(user: UserCreate):
    new_id = max([u.id for u in users]) + 1
    new_user = User(id=new_id, name=user.name, email=user.email)
    users.append(new_user)
    return new_user

@app.put("/users/{user_id}", response_model=User)
async def update_user(user_id: int, user: UserCreate):
    for i, existing_user in enumerate(users):
        if existing_user.id == user_id:
            users[i] = User(id=user_id, name=user.name, email=user.email)
            return users[i]
    raise HTTPException(status_code=404, detail="User not found")

@app.delete("/users/{user_id}")
async def delete_user(user_id: int):
    for i, user in enumerate(users):
        if user.id == user_id:
            del users[i]
            return {"message": "User deleted"}
```

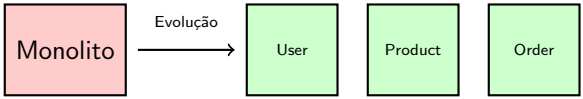
Arquitetura de Microservices

Monolito vs Microservices

- **Monolito**: Uma aplicação única
- **Microservices**: Múltiplas aplicações pequenas

Vantagens dos Microservices:

- Escalabilidade independente
- Tecnologias diferentes
- Deploy independente
- Falhas isoladas



Exemplo de Microservices

Service	Porta	Responsabilidade
User Service	8001	Gerenciar usuários
Product Service	8002	Gerenciar produtos
Order Service	8003	Gerenciar pedidos
Payment Service	8004	Processar pagamentos
API Gateway	8000	Roteamento e autenticação

Comunicação entre Services

- **Síncrona**: HTTP/REST, gRPC
- **Assíncrona**: Message Queues (RabbitMQ, Kafka)
- **Service Discovery**: Consul, Eureka

O que é Docker?

Containerização

- Empacota aplicação e suas dependências
- Ambiente isolado e portável
- Consistência entre desenvolvimento e produção
- Escalabilidade horizontal

Vantagens:

- "Funciona na minha máquina"
- Deploy rápido
- Isolamento
- Recursos otimizados

Conceitos:

- **Image**: Template
- **Container**: Instância
- **Dockerfile**: Receita
- **Registry**: Repositório

Dockerfile para FastAPI

```
# Usar imagem base do Python
FROM python:3.9-slim

# Definir diretorio de trabalho
WORKDIR /app

# Copiar arquivos de dependencias
COPY requirements.txt .

# Instalar dependencias
RUN pip install --no-cache-dir -r requirements.txt

# Copiar codigo da aplicacao
COPY . .

# Expor porta
EXPOSE 8000

# Comando para executar a aplicacao
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

requirements.txt

```
fastapi==0.104.1
uvicorn[standard]==0.24.0
pydantic==2.5.0
```

Docker Compose para Microservices

```yaml
version: '3.8'

services:
  user-service:
    build: ./user-service
    ports:
      - "8001:8000"
    environment:
      - DATABASE_URL=postgresql://user:pass@db:5432/users
    depends_on:
      - db

  product-service:
    build: ./product-service
    ports:
      - "8002:8000"
    environment:
      - DATABASE_URL=postgresql://user:pass@db:5432/products
    depends_on:
      - db

  api-gateway:
    build: ./api-gateway
    ports:
      - "8000:8000"
    depends_on:
      - user-service
      - product-service

  db:
```

O que é Kubernetes?

Orquestração de Containers

- Plataforma open-source para orquestração
- Automatiza deploy, scaling e gerenciamento
- Desenvolvido pelo Google
- Padrão da indústria

Recursos Principais:

- **Pods**: Unidade mínima
- **Services**: Descoberta e load balancing
- **Deployments**: Gerenciamento de replicas
- **Ingress**: Roteamento HTTP

Benefícios:

- Auto-scaling
- Auto-healing
- Rolling updates
- Service discovery
- Load balancing

Deployment Kubernetes

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: user-service
  labels:
    app: user-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: user-service
  template:
    metadata:
      labels:
        app: user-service
    spec:
      containers:
      - name: user-service
        image: user-service:latest
        ports:
        - containerPort: 8000
        env:
        - name: DATABASE_URL
          value: "postgresql://user:pass@db:5432/users"
---
apiVersion: v1
kind: Service
metadata:
  name: user-service
spec:
```

Ingress para API Gateway

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: api-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: api.example.com
    http:
      paths:
      - path: /users
        pathType: Prefix
        backend:
          service:
            name: user-service
            port:
              number: 80
      - path: /products
        pathType: Prefix
        backend:
          service:
            name: product-service
            port:
              number: 80
```

Demonstração: API Completa

1. **Criar API FastAPI** (10 min)
   - Estrutura básica
   - Endpoints CRUD
   - Validação de dados
2. **Containerizar com Docker** (10 min)
   - Dockerfile
   - Build da imagem
   - Executar container
3. **Microservices com Docker Compose** (15 min)
   - Múltiplos serviços
   - Comunicação entre serviços
   - Banco de dados
4. **Deploy no Kubernetes** (15 min)
   - Configuração YAML
   - Deploy dos serviços
   - Teste da aplicação
5. **Testes e Monitoramento** (10 min)
   - Testes de carga

O que aprender depois?

Conceitos Avançados:

- Service Mesh (Istio)
- Message Queues
- Event Sourcing
- CQRS Pattern
- Circuit Breaker
- Distributed Tracing

Ferramentas:

- Prometheus + Grafana
- ELK Stack
- Jaeger
- Helm
- ArgoCD
- Terraform

Recursos para Estudo

- Documentação FastAPI
- Documentação Kubernetes
- Documentação Docker
- Microservices Patterns

Obrigado!

Perguntas?

Contato: bento.siqueira@ufla.br
Repositório: github.com/californi/disciplinas-ufla