

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

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

: Validação automática

Async/Await: Suporte nativo

Fácil de usar: Sintaxe simples

bcom

```
1 from fastapi import FastAPI
2 from pydantic import BaseModel
3 from typing import List
4
5 app = FastAPI(title="Minha API", version="1.0.0")
6
7 # Modelo de dados
8 class User(BaseModel):
9     id: int
10    name: str
11    email: str
12
13 # Dados em memoria (para demonstracao)
14 users = [
15     User(id=1, name="Joao", email="joao@email.com"),
16     User(id=2, name="Maria", email="maria@email.com")
17 ]
18
19 @app.get("/")
20 async def root():
21     return {"message": "Hello World!"}
22
23 @app.get("/users", response_model=List[User])
```

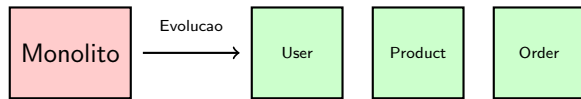
```
24 async def get_users():
25     return users
26
27 @app.get("/users/{user_id}", response_model=User)
28 async def get_user(user_id: int):
29     for user in users:
30         if user.id == user_id:
31             return user
32     return {"error": "User not found"}
```

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 usuarios
Product Service	8002	Gerenciar produtos
Order Service	8003	Gerenciar pedidos
Payment Service	8004	Processar pagamentos
API Gateway	8000	Roteamento e autenticao

Comunicacao entre Services

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

O que é Docker?

Containerização

- Empacota aplicação e suas dependências
- Ambiente isolado e portátil
- 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



```
1 # Usar imagem base do Python
2 FROM python:3.9-slim
3
4 # Definir diretório de trabalho
5 WORKDIR /app
6
7 # Copiar arquivos de dependências
8 COPY requirements.txt .
9
10 # Instalar dependências
11 RUN pip install --no-cache-dir -r requirements.txt
12
13 # Copiar código da aplicação
14 COPY . .
15
16 # Expor porta
17 EXPOSE 8000
18
19 # Comando para executar a aplicação
20 CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

requirements.txt

```
1 fastapi==0.104.1
2 uvicorn[standard]==0.24.0
3 pydantic==2.5.0
```

Docker Compose para Microservices

```
1 version: '3.8'
2
3 services:
4   user-service:
5     build: ./user-service
6     ports:
7       - "8001:8000"
8     environment:
9       - DATABASE_URL=postgresql://user:pass@db:5432/users
10    depends_on:
11      - db
12
13   product-service:
14     build: ./product-service
15     ports:
16       - "8002:8000"
17     environment:
18       - DATABASE_URL=postgresql://user:pass@db:5432/products
19    depends_on:
20      - db
21
22   api-gateway:
23     build: ./api-gateway
24     ports:
25       - "8000:8000"
26     depends_on:
27       - user-service
28       - product-service
29
30   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

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: user-service
5   labels:
6     app: user-service
7 spec:
8   replicas: 3
9   selector:
10    matchLabels:
11      app: user-service
12   template:
13     metadata:
14       labels:
15         app: user-service
16     spec:
17       containers:
18       - name: user-service
19         image: user-service:latest
20         ports:
21         - containerPort: 8000
22         env:
23         - name: DATABASE_URL
24           value: "postgresql://user:pass@db:5432/users"
25 ---
26 apiVersion: v1
27 kind: Service
28 metadata:
29   name: user-service
30 spec:
```

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

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