



# Migrando de Kubernetes Ingress para Gateway API na prática

Do modelo clássico ao padrão moderno de tráfego L7 • Por Márcio "Zamp" Zampiron

# O que você vai aprender hoje



## Diferenças fundamentais

Entenda como Ingress e Gateway API se comparam, seus pontos fortes e limitações



## Arquitetura da Gateway API

Explore GatewayClass, Gateway e HTTPRoute e como eles se conectam



## Migração prática

Aprenda o passo a passo para migrar seus recursos de Ingress para Gateway API



## Hands-on com manifests

Execute exemplos reais usando YAMLs prontos do repositório Git público

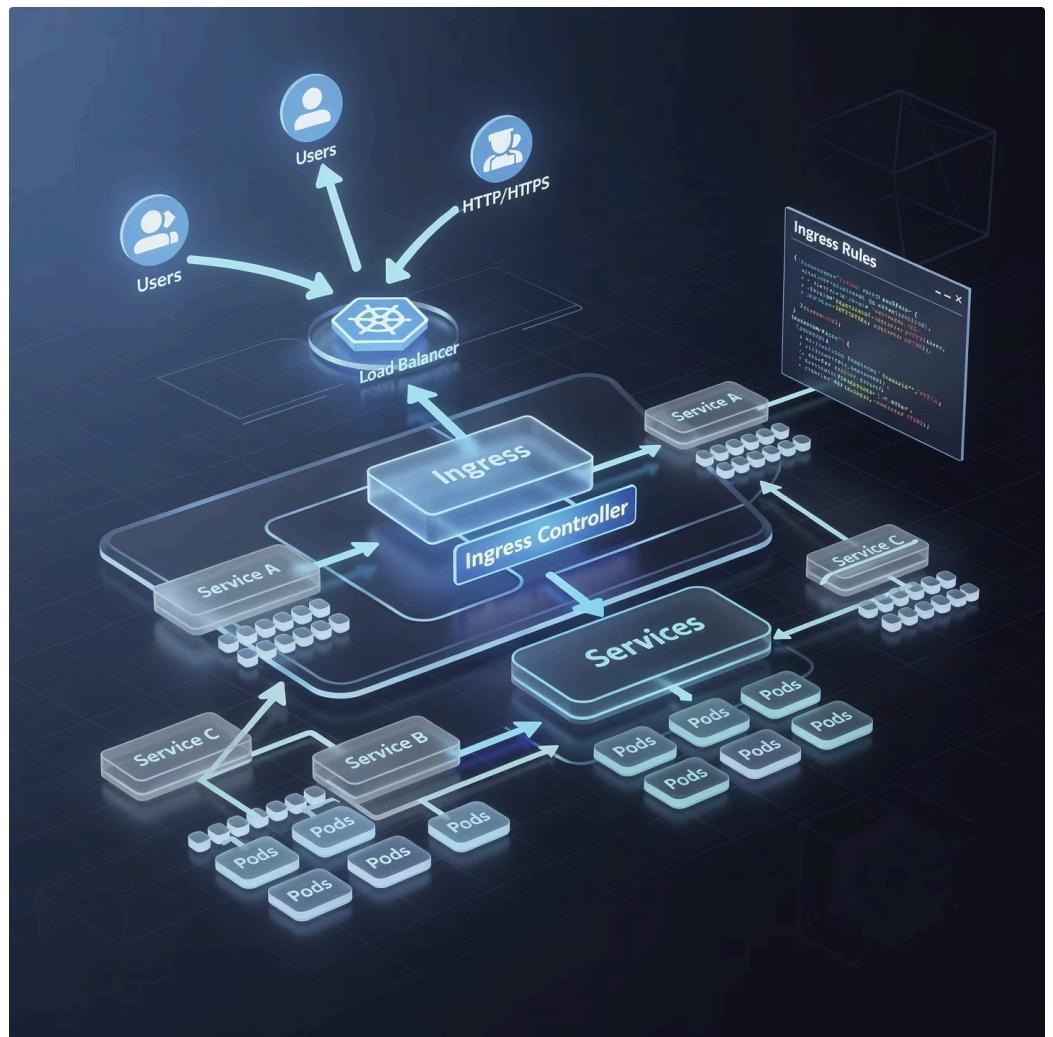
# Revisão: O que é Ingress?

## Conceito básico

Ingress é um recurso do Kubernetes que gerencia acesso externo aos serviços no cluster, tipicamente HTTP/HTTPS. Ele funciona como um **ponto de entrada unificado** para o tráfego L7.

## Componentes principais

- **Recurso Ingress:** define regras de roteamento (hosts, paths, backends)
- **Ingress Controller:** implementa as regras (NGINX, Traefik, HAProxy, etc.)
- **Backend Service:** destino final do tráfego (seus Pods)



- **Fluxo básico:** Cliente → LoadBalancer → Ingress Controller → Service → Pods



# Limitações do modelo Ingress

## API limitada e pouco expressiva

A especificação do Ingress é minimalista demais para cenários complexos de roteamento, deixando muita funcionalidade fora do padrão

## Dependência de annotations

Comportamentos avançados (rate limiting, rewrite, autenticação) exigem **annotations específicas** do controlador, quebrando portabilidade

## Inconsistências entre implementações

Cada Ingress Controller interpreta configurações de forma diferente, dificultando migração entre fornecedores

## Gestão compartilhada difícil

Times de infraestrutura e aplicação não conseguem separar responsabilidades de forma clara no mesmo recurso

# Gateway API: A evolução natural

A **Gateway API** é a próxima geração de APIs para gerenciar tráfego de entrada no Kubernetes. Desenvolvida pela comunidade Kubernetes SIG Network, ela oferece uma abordagem **role-oriented** mais poderosa e expressiva.

## Principais vantagens

- **Separação de responsabilidades:** infra pode gerenciar Gateways, devs gerenciam rotas
- **Expressividade nativa:** recursos ricos sem depender de annotations
- **Portabilidade:** implementações consistentes entre controladores
- **Extensibilidade:** suporte a TCPRoute, UDPRoute, GRPCRoute, filtros customizados
- **Multi-tenant:** isolamento claro entre namespaces e times



# Conceitos fundamentais da Gateway API

01

## GatewayClass

Define o **"tipo"** de gateway disponível no cluster. É gerenciado pelo time de infraestrutura e associado a um controlador específico (exemplo: nginx, istio, traefik).

02

## Gateway

Instância concreta de um ponto de entrada. Define **listeners** (portas, protocolos, hostnames, TLS). É o recurso de infraestrutura que aceita tráfego externo.

03

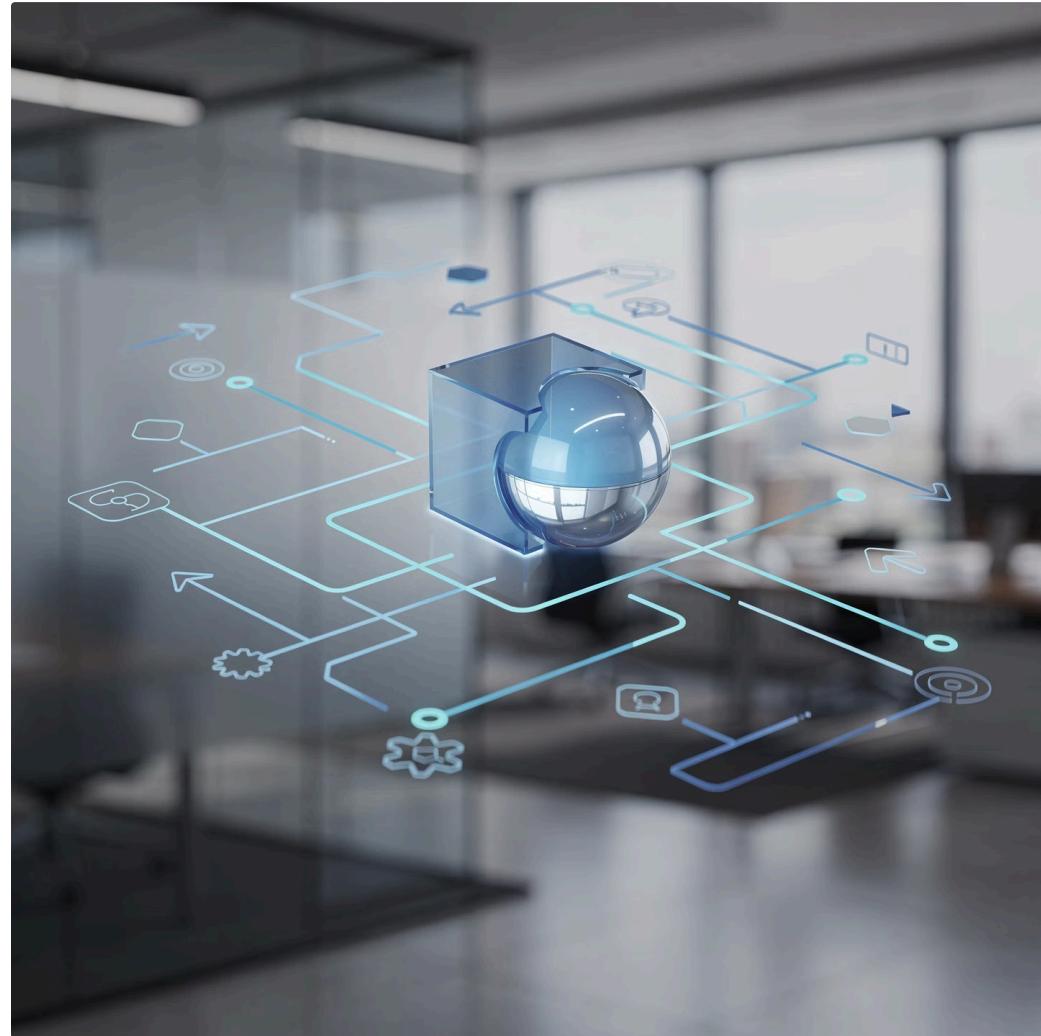
## HTTPRoute

Regras de roteamento HTTP. Define **quais requisições** (hosts, paths, headers, métodos) vão para **quais backends** (Services). Gerenciado por times de aplicação.

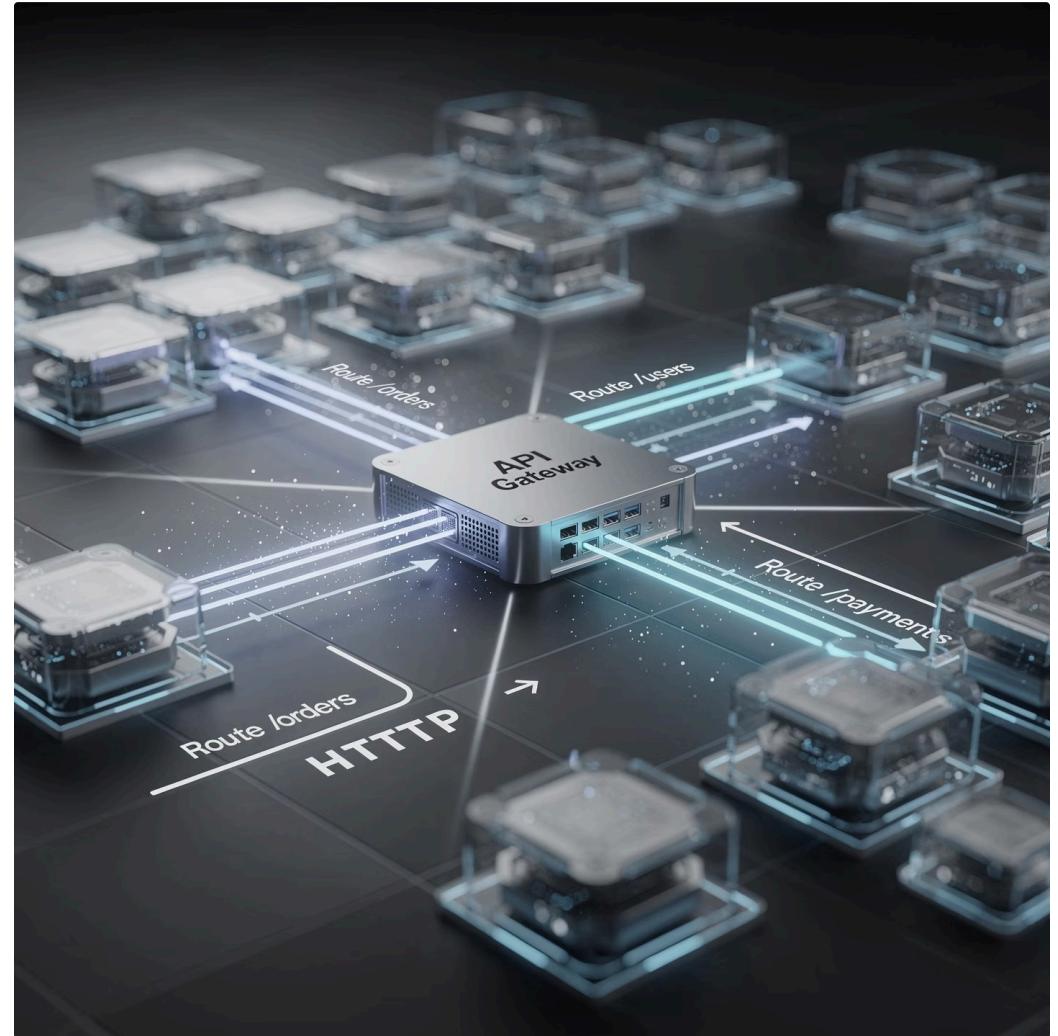
- ☐ **Outros tipos de rota:** TCPRoute (L4), GRPCRoute (gRPC nativo), TLSRoute (SNI routing). Não veremos nesta aula, mas são parte do ecossistema.

# Ingress vs Gateway API: Comparação visual

Modelo Ingress (clássico)



Gateway API (moderno)



- Um único recurso Ingress
- Configuração misturada (infra + app)
- Annotations para features avançadas
- Pouca granularidade de permissões

- Recursos separados (Gateway + Routes)
- Responsabilidades divididas por papel
- Configuração nativa e expressiva
- RBAC granular por recurso e namespace

---

**Analogia:** Ingress é como um porteiros com uma lista simples de nomes. Gateway API é um porteiros com painel de controle completo, câmeras e múltiplos critérios de validação.



# Arquitetura do nosso Lab

Vamos trabalhar no namespace `ingress-gateway-lab` com uma aplicação web simples (`demo-web`). Nosso objetivo é expor essa aplicação de duas formas paralelas para comparar as abordagens.



## Deployment + Service

App `demo-web` (`nginxdemos/hello`) exposto via Service ClusterIP na porta 80



## Exposição via Ingress

Ingress tradicional roteando `demo.example.com` para o Service



## Exposição via Gateway API

Gateway + HTTPRoute roteando o mesmo host para o mesmo Service

- ☐ **Repositório Git:** Todos os manifests YAML estarão disponíveis no repositório público. Você vai clonar e aplicar os arquivos durante o hands-on.

# YAML do Ingress (resumo)

O arquivo `02-ingress.yaml` define um recurso Ingress tradicional.

Veja os principais campos:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: demo-web-ingress
  namespace: ingress-gateway-lab
spec:
  rules:
    - host: demo.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: demo-web
                port:
                  number: 80
```

## Como funciona

- O **Ingress Controller** lê esse recurso e configura o proxy reverso interno
- Requisições para `demo.example.com` são roteadas ao Service `demo-web`
- Comportamentos avançados (TLS, rewrites, autenticação) vão via **annotations** específicas do controlador

💡 O YAML completo está no repositório Git. Use `kubectl apply -f manifests/02-ingress.yaml` para criar o recurso.

# YAML de Gateway + HTTPRoute (resumo)

O arquivo `03-gateway-httproute.yaml` contém três recursos: `GatewayClass`, `Gateway` e `HTTPRoute`. Vamos ver os trechos principais de cada um.

## GatewayClass

```
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: example-gateway-class
spec:
  controllerName: example.com/gateway
```

Define o **tipo** de gateway, associado ao controlador que vai implementá-lo.

## Gateway

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: demo-gateway
  namespace: ingress-gateway-lab
spec:
  gatewayClassName: example-gateway-class
  listeners:
    - name: http
      protocol: HTTP
      port: 80
      hostname: demo.example.com
```

Instância do gateway, escutando HTTP na porta 80 para o hostname especificado.

# HTTPRoute: roteamento explícito

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: demo-web-httproute
  namespace: ingress-gateway-lab
spec:
  parentRefs:
    - name: demo-gateway
  hostnames:
    - demo.example.com
  rules:
    - matches:
        - path:
            type: PathPrefix
            value: /
  backendRefs:
    - name: demo-web
      port: 80
```

## Entendendo o HTTPRoute

- **parentRefs:** vincula essa rota ao Gateway `demo-gateway`
- **hostnames:** define quais hosts essa rota atende
- **rules:** regras de match (path, método, headers) e backends de destino
- **backendRefs:** lista de Services para onde o tráfego vai (com pesos opcionais para A/B testing)

Onde antes tínhamos **um Ingress**, agora temos **Gateway + HTTPRoute**. O roteamento fica mais explícito e separado da configuração de infraestrutura.

# Passo a passo do Lab

## Deploy da aplicação

Aplique 01-app-deploy-service.yaml para criar o namespace, Deployment e Service do demo-web. Verifique com `kubectl get pods -n ingress-gateway-lab`.

## Teste com Ingress

Aplique 02-ingress.yaml e teste o acesso usando `curl -H "Host: demo.example.com" http://<INGRESS-IP>`. Confirme que está funcionando.

## Deploy do Gateway API

Aplique 03-gateway-httproute.yaml (GatewayClass, Gateway, HTTPRoute). Aguarde o Gateway ficar pronto com `kubectl get gateway -n ingress-gateway-lab`.

## Teste com Gateway

Teste o acesso via Gateway API usando `curl -H "Host: demo.example.com" http://<GATEWAY-IP>`. Valide que ambas as rotas funcionam.

## Planeje a remoção do Ingress

Após validar que o Gateway funciona corretamente, você pode remover o Ingress antigo com `kubectl delete ingress demo-web-ingress -n ingress-gateway-lab`.

# Boas práticas ao migrar

1

## Comece pelo menos crítico

Escolha serviços de desenvolvimento ou staging para migrar primeiro. Valide o comportamento antes de tocar em produção crítica.

2

## Entenda seu controlador

Cada implementação de Gateway API tem features e limitações específicas. Leia a documentação do controlador que você vai usar (NGINX Gateway Fabric, Istio, Envoy Gateway, etc.).

3

## Não copie annotations cegamente

Use os campos nativos da Gateway API sempre que possível. Evite replicar o padrão antigo de annotations para configurações avançadas.

4

## Monitore e observe

Tenha logs, métricas e traces do seu controlador de Gateway. Use ferramentas como Prometheus, Grafana e Jaeger para depurar problemas de roteamento.

5

## Mantenha paralelo durante a transição

Rode Ingress e Gateway lado a lado por um tempo. Valide tráfego, latência e erros antes de fazer o cutover completo.

# Benefícios adicionais da Gateway API



## Recursos avançados nativos

- **Traffic splitting:** distribua tráfego entre múltiplos backends (A/B, canary)
- **Header manipulation:** adicione, remova ou modifique headers sem annotations
- **Request mirroring:** duplique tráfego para backends de teste
- **Redirects e rewrites:** controle nativo de redirecionamentos e reescrita de paths
- **Timeout e retry policies:** configure timeouts e retries de forma declarativa

Tudo isso sem depender de configurações proprietárias de cada controlador.



# Separação de responsabilidades na prática

## Time de Infraestrutura

**Gerencia:** GatewayClass e Gateways

- Define controladores disponíveis
- Configura listeners, TLS, políticas de segurança
- Controla recursos de infraestrutura (IPs, load balancers)

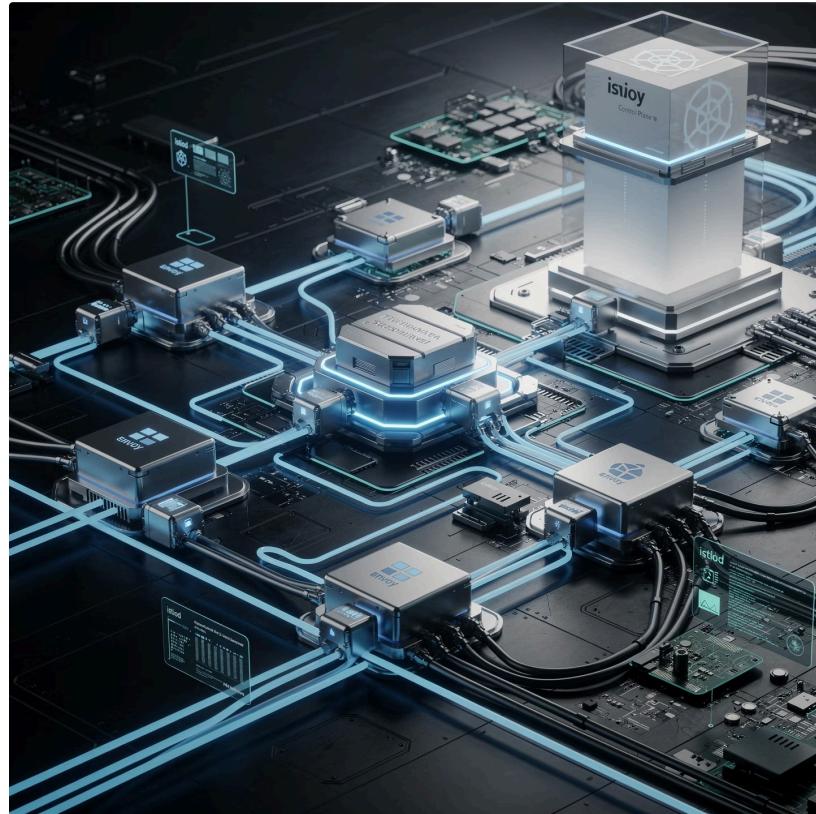
## Time de Aplicação

**Gerencia:** HTTPRoutes e outros tipos de rota

- Define regras de roteamento para suas aplicações
- Configura hosts, paths, headers, backends
- Implementa traffic splitting e A/B testing

Com RBAC apropriado, cada time trabalha apenas nos recursos de sua responsabilidade, sem conflitos ou dependências desnecessárias.

# Gateway API e Service Mesh



A Gateway API não é apenas para **ingress**. Ela também é usada por **service meshes** modernos como Istio e Linkerd para configurar tráfego interno (east-west).

## Vantagens dessa convergência

- **API unificada:** mesmos recursos para ingress e mesh
- **Portabilidade:** mude de mesh ou ingress controller com menos refatoração
- **Ecossistema crescente:** ferramentas de observabilidade, segurança e GitOps integram-se nativamente

A Gateway API está se tornando o padrão para **todo tipo de roteamento L4/L7** no Kubernetes, não apenas ingress.

# Implementações de Gateway API



## NGINX Gateway Fabric

Implementação oficial da NGINX.  
Focada em performance e simplicidade.



## Istio

Service mesh completo com suporte robusto a Gateway API para ingress e tráfego interno.



## Envoy Gateway

Gateway construído sobre o proxy Envoy, parte do projeto oficial da comunidade.



## Traefik

Proxy moderno com suporte nativo a Gateway API e integração com Kubernetes.



## Kong Gateway

API Gateway com rich feature set, plugins extensíveis e suporte a Gateway API.



## Contour

Ingress controller baseado em Envoy com implementação completa de Gateway API.

# Perguntas de revisão (Quiz)

Teste seu conhecimento respondendo às perguntas abaixo. Vamos revisar as respostas juntos!

- Qual a principal diferença entre Ingress e Gateway API?
  - a) Ingress é para HTTP, Gateway API só para TCP
  - b) Ingress é um único recurso; Gateway API separa conceitos em GatewayClass, Gateway e HTTPRoute
  - c) Gateway API substitui o Service
  - d) Ingress não precisa de controlador, Gateway precisa
- Qual o papel do recurso `GatewayClass` na Gateway API?
- No exemplo do lab, qual recurso é responsável por definir as regras de roteamento HTTP (`host`, `path`, `backend`)?
  - a) Gateway
  - b) Service
  - c) HTTPRoute
  - d) Deployment

# Perguntas de revisão (continuação)

- Por que muitas equipes querem sair do modelo baseado apenas em Ingress?
- Durante a migração, qual prática é mais segura?
  - a) Deletar todos os Ingress de uma vez e criar Gateways depois
  - b) Manter Ingress e Gateway em paralelo por um tempo, validando tráfego e fazendo cutover gradual
  - c) Ter dois Gateways para o mesmo host sem controle de DNS
  - d) Usar apenas port-forward sem alterar nada em produção
- No `HTTPRoute`, o que o campo `backendRefs` representa?

 **Vamos discutir:** Que serviço do seu cluster faria mais sentido migrar primeiro? Quais desafios você antecipa?



# Obrigado! Continue a jornada Cloud Native

## Recapitulando

- **Ingress** é o modelo clássico, mas tem limitações de expressividade e portabilidade
- **Gateway API** é a evolução: separação de responsabilidades, recursos nativos ricos, portabilidade
- **GatewayClass, Gateway, HTTPRoute**: três conceitos fundamentais para dominar
- **Lab prático**: migração simples de Ingress para Gateway API disponível no repositório Git

[Acessar Repositório Git](#)

[Acessar linkedin](#)

### Referências

- [api gateway](#)
- [gateway-api.sigs.k8s.io](#)

### Repositório do Lab

Acesse todos os manifests YAML e instruções completas no repositório Git público da aula