Aula 24 – Padrões de Deploy no Kubernetes

No dinâmico universo do desenvolvimento de aplicações web, a capacidade de entregar novas funcionalidades e correções de forma rápida e segura é um diferencial competitivo. Imagine um cenário onde cada atualização do seu sistema exige que ele saia do ar por alguns minutos ou até horas. Para uma aplicação crítica, como um e-commerce em Black Friday ou um sistema bancário, isso é simplesmente inaceitável. É aqui que os padrões de deploy entram em cena, transformando o processo de atualização de um risco em uma rotina eficiente.

Esta aula foi cuidadosamente elaborada para desmistificar os principais padrões de deploy utilizados no Kubernetes, a plataforma que se tornou o coração das arquiteturas distribuídas modernas. Você aprenderá não apenas o "como", mas o "porquê" de cada estratégia, conectando o conhecimento teórico com a realidade do mercado de trabalho. Nosso objetivo é que, ao final, você seja capaz de analisar diferentes cenários e escolher o padrão de deploy mais adequado, garantindo que suas aplicações permaneçam sempre disponíveis, mesmo durante as atualizações mais complexas.

Ao longo das próximas páginas, exploraremos o Rolling Update, o Blue-Green Deployment e o Canary Release, entendendo suas nuances, vantagens e desvantagens. Veremos como cada um se encaixa na busca por deploys seguros e sem interrupções, um pilar fundamental para a resiliência e a agilidade que as arquiteturas de microserviços e serverless exigem. Prepare-se para mergulhar em estratégias que definem a vanguarda do desenvolvimento web moderno, capacitando-o a construir e manter sistemas robustos e de alta performance.

O Desafio da Entrega Contínua em Ambientes Distribuídos

No cenário atual de arquiteturas distribuídas, onde microserviços e funções serverless reinam, a complexidade de gerenciar atualizações cresceu exponencialmente. Longe vão os dias em que um "deploy" significava simplesmente substituir um arquivo em um servidor. Hoje, estamos falando de dezenas, centenas ou até milhares de componentes interconectados, cada um com seu próprio ciclo de vida. O grande desafio é como introduzir uma nova versão de um desses componentes sem causar interrupções no serviço, sem degradar a experiência do usuário e, idealmente, com a capacidade de reverter rapidamente caso algo dê errado.

Pense em um grande centro de distribuição de mercadorias, operando 24 horas por dia. Se você precisasse parar todas as esteiras e empilhadeiras para atualizar o software de gerenciamento, o prejuízo seria imenso. Da mesma forma, em uma aplicação web moderna, cada segundo de inatividade pode significar perda de receita, de reputação e de confiança do cliente. É essa necessidade premente de "zero downtime" e de entregas ágeis que impulsionou o desenvolvimento e a adoção de padrões de deploy sofisticados, especialmente em plataformas como o Kubernetes, que foi projetado para orquestrar esses ambientes complexos de forma eficiente.

Rolling Update: A Substituição Gradual e Segura

Substituição Gradual

Instâncias antigas são substituídas por novas de forma progressiva, uma a uma ou em pequenos lotes

Serviço Contínuo

Sempre há instâncias da versão anterior rodando enquanto as novas estão sendo inicializadas

Rollback Seguro

Se uma nova instância falhar, o processo pode ser pausado ou revertido sem impacto

O Rolling Update, ou atualização progressiva, é o padrão de deploy mais comum e, muitas vezes, o padrão default em orquestradores como o Kubernetes. Sua premissa é simples, mas poderosa: em vez de derrubar todas as instâncias de uma aplicação para subir a nova versão, ele substitui as instâncias antigas por novas de forma gradual, uma a uma ou em pequenos lotes. Isso garante que sempre haja instâncias da versão anterior rodando enquanto as novas estão sendo inicializadas, mantendo o serviço disponível.

> 💡 **Analogia:** Imagine que você precisa trocar os pneus de um carro de corrida durante uma prova, mas sem parar o carro. Impossível, certo? No mundo digital, o Rolling Update faz algo parecido. Ele permite que você "troque os pneus" (as instâncias da sua aplicação) enquanto o "carro" (seu serviço) continua em movimento.

Essa abordagem minimiza o risco de interrupção, pois se uma nova instância falhar ao iniciar ou apresentar problemas, o processo pode ser pausado ou revertido, e as instâncias antigas ainda estarão operacionais. É como ter uma rede de segurança: você tenta a mudança, mas se algo der errado, o sistema volta para um estado estável sem que o usuário final perceba.

Rolling Update na Prática com Kubernetes

No Kubernetes, o Rolling Update é gerenciado por um recurso chamado Deployment. Quando você atualiza a imagem de um contêiner em um Deployment, o Kubernetes não derruba tudo de uma vez. Em vez disso, ele cria um novo ReplicaSet para a nova versão e, gradualmente, aumenta o número de réplicas no novo ReplicaSet enquanto diminui o número de réplicas no ReplicaSet antigo. Isso acontece de forma orquestrada, garantindo que o número total de réplicas disponíveis para o serviço permaneça constante ou dentro de limites aceitáveis.

Parâmetros de Controle

- **maxUnavailable:** Quantas réplicas podem estar indisponíveis durante o update
- **maxSurge:** Quantas réplicas podem ser criadas acima do número desejado durante o update

Exemplo de Fluxo

Com 5 réplicas, o Kubernetes:

1. Cria 1 nova réplica (v2.0.0)
2. Aguarda ela ficar saudável
3. Termina 1 réplica antiga (v1.0.0)
4. Repete até todas estarem atualizadas

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: minha-aplicacao
spec:
  replicas: 5
  selector:
    matchLabels:
      app: minha-aplicacao
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25% # Permite que 25% das réplicas estejam indisponíveis
      maxSurge: 25% # Permite que 25% das réplicas sejam criadas acima do total
  template:
    metadata:
      labels:
        app: minha-aplicacao
    spec:
      containers:
      - name: web
        image: minha-aplicacao:v1.0.0 # Versão atual
        ports:
        - containerPort: 80
```

🚀 **Para atualizar:** Basta mudar a imagem no YAML para `v2.0.0` e aplicar: `kubectl apply -f deployment.yaml`. O Kubernetes fará o resto.

Blue-Green Deployment: A Troca Instantânea de Ambientes

Enquanto o Rolling Update é gradual, o Blue-Green Deployment adota uma abordagem mais radical, mas igualmente eficaz para garantir zero downtime. A ideia central é manter dois ambientes de produção idênticos: um ambiente "azul" (Blue), que está atualmente ativo e recebendo todo o tráfego, e um ambiente "verde" (Green), onde a nova versão da aplicação é implantada e testada. Uma vez que o ambiente verde esteja totalmente validado, o tráfego é instantaneamente desviado do ambiente azul para o verde.

01

Ambiente Blue Ativo

Versão atual recebendo 100% do tráfego

02

Deploy no Green

Nova versão implantada e testada em paralelo

03

Validação Completa

Testes de integração e aceitação no ambiente Green

04

Troca Instantânea

Tráfego redirecionado do Blue para o Green

A principal vantagem do Blue-Green é a capacidade de realizar um rollback instantâneo. Se, após a troca, algum problema crítico for detectado no ambiente verde, basta redirecionar o tráfego de volta para o ambiente azul, que ainda está rodando a versão anterior e comprovadamente estável. Isso oferece uma camada de segurança e confiança muito alta para deploys em sistemas críticos, onde qualquer falha pode ter consequências graves.

Implementando Blue-Green com Kubernetes e Serviços

A implementação de um Blue-Green Deployment no Kubernetes geralmente envolve a criação de dois Deployments separados, um para a versão "Blue" e outro para a versão "Green". Ambos os Deployments são expostos por um único Service (ou Ingress), que atua como o ponto de entrada para o tráfego externo. Inicialmente, o Service aponta para o Deployment "Blue".

Fase 1: Blue Ativo

Service configurado com selector apontando para `version: blue`

- Todo tráfego vai para Blue
- Green está sendo preparado
- Testes em andamento no Green

Fase 2: Troca para Green

Service atualizado com selector para `version: green`

- Tráfego redirecionado instantaneamente
- Blue mantido como backup
- Rollback disponível a qualquer momento

```
# Service apontando inicialmente para a versão 'blue'
apiVersion: v1
kind: Service
metadata:
  name: minha-aplicacao-service
spec:
  selector:
    app: minha-aplicacao
    version: blue # Aponta para o deployment 'blue'
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  type: LoadBalancer
```

> 🔄 **Para alternar:** Atualize o selector do Service para `version: green`. Essa é a "chave" que vira o tráfego de forma instantânea.

Quando uma nova versão precisa ser implantada, um novo Deployment "Green" é criado com a nova imagem da aplicação. Após a validação completa do ambiente "Green" (testes de integração, testes de aceitação, etc.), o seletor do Service é atualizado para apontar para o Deployment "Green". Essa mudança no seletor é quase instantânea, redirecionando todo o tráfego para a nova versão. O ambiente "Blue" pode ser mantido por um tempo como backup ou ser desativado para economizar recursos.

Canary Release: O Teste Controlado com Usuários Reais

O Canary Release, ou lançamento canário, é uma estratégia que combina a segurança de um deploy gradual com a capacidade de testar novas funcionalidades em um ambiente de produção real, mas com um risco controlado. Inspirado na prática de usar canários em minas de carvão para detectar gases tóxicos, este padrão envolve o lançamento da nova versão da aplicação para uma pequena porcentagem de usuários reais. Se a nova versão se mostrar estável e performática para esse grupo "canário", ela é gradualmente liberada para mais usuários, até atingir 100%.

1% dos Usuários

Teste inicial com grupo pequeno

5% → 10% → 25%

Aumento gradual baseado em métricas

100% dos Usuários

Rollout completo após validação

> **Analogia:** Imagine que você está lançando um novo recurso em uma rede social popular. Em vez de liberar para todos os bilhões de usuários de uma vez, você decide testar com apenas 1% deles. Se esse 1% não reportar bugs ou problemas de performance, você aumenta para 5%, depois 10%, e assim por diante.

A grande vantagem do Canary Release é a mitigação de riscos. Ao expor a nova versão a um grupo limitado de usuários, qualquer problema que surja afetará apenas uma pequena fração da sua base de usuários, permitindo que você reaja rapidamente, revertendo a mudança ou corrigindo o problema antes que ele se espalhe. É uma forma inteligente de "testar em produção" com segurança.

Mecanismos do Canary Release no Kubernetes

A implementação de um Canary Release no Kubernetes geralmente requer ferramentas mais avançadas do que apenas Deployments e Services básicos, embora seja possível com eles. A forma mais eficaz envolve o uso de um Service Mesh (como Istio, Linkerd) ou um Ingress Controller avançado que suporte roteamento baseado em peso ou em regras.

Ferramentas Necessárias

- **Service Mesh:** Istio, Linkerd
- **Ingress Controller:** NGINX, Traefik
- **Monitoramento:** Prometheus, Grafana
- **Observabilidade:** Jaeger, Datadog

Capacidades de Roteamento

- Distribuição por peso (5% vs 95%)
- Roteamento por cabeçalhos HTTP
- Roteamento por cookies de sessão
- Segmentação por região geográfica

```
# Exemplo conceitual de regra de tráfego com Service Mesh (e.g., Istio VirtualService)
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: minha-aplicacao
spec:
  hosts:
  - minha-aplicacao.exemplo.com
  http:
  - route:
    - destination:
        host: minha-aplicacao-v1
      weight: 95 # 95% do tráfego para a versão estável
    - destination:
        host: minha-aplicacao-v2
      weight: 5 # 5% do tráfego para a versão canário
```

> 📊 **Monitoramento Crítico:** Acompanhe métricas como taxa de erros, latência e uso de recursos para a versão v2. Com base nos dados, você pode aumentar gradualmente o `weight` para 100% ou reverter para 0%.

Com um Service Mesh, você pode configurar regras de tráfego que direcionam uma porcentagem específica do tráfego para a nova versão (o "canário") e o restante para a versão estável. Por exemplo, 5% do tráfego para a minha-aplicacao:v2.0.0 e 95% para a minha-aplicacao:v1.0.0. Você pode até mesmo rotear tráfego com base em atributos específicos, como cabeçalhos HTTP ou cookies, permitindo que apenas usuários internos ou testadores beta vejam a nova versão.

Comparando os Padrões de Deploy

Agora que exploramos individualmente cada padrão, é fundamental entender suas diferenças e quando cada um brilha. A escolha do padrão ideal depende de diversos fatores, como a criticidade da aplicação, a tolerância a riscos, a infraestrutura disponível e a complexidade das mudanças. Não existe uma solução única que sirva para todos os casos; o segredo está em saber qual ferramenta usar para cada martelo.

Rolling Update

Como construir uma ponte nova enquanto a antiga ainda está em uso, substituindo as tábuas uma a uma

Blue-Green

Como construir uma ponte completamente nova ao lado da antiga e desviar o tráfego quando pronta

Canary Release

Como testar uma pequena passarela nova com algumas pessoas antes de abrir a ponte principal

Quadro Comparativo

Padrão	Âmbito/Aplicação	Base/Origem	Exemplo
Rolling Update	Atualizações de rotina, correções de bugs	Substituição gradual de instâncias	Atualizar uma API de microserviço de v1 para v2 sem interrupção
Blue-Green	Deploys críticos, grandes mudanças, rollback rápido	Dois ambientes idênticos, troca de tráfego	Lançamento de uma nova versão principal de um sistema bancário
Canary Release	Lançamento de novas funcionalidades, testes A/B	Liberação gradual para subconjunto de usuários	Testar um novo algoritmo de recomendação com 1% dos usuários de um e-commerce

Escolhendo a Estratégia Certa para o Seu Cenário

A decisão sobre qual padrão de deploy utilizar não é trivial e deve ser guiada por uma análise cuidadosa das necessidades do seu projeto e da sua organização. Não se trata apenas de qual é o mais "avançado", mas sim de qual se alinha melhor com seus objetivos de negócio, sua tolerância a riscos e sua capacidade de infraestrutura. Um deploy Blue-Green, por exemplo, oferece um rollback quase instantâneo, mas exige o dobro de recursos de infraestrutura em determinado momento.

Criticidade da Aplicação

Para sistemas de saúde ou financeiros que não podem ter um segundo de inatividade, Blue-Green ou Canary são preferíveis. Para aplicações internas com menor impacto, Rolling Update pode ser adequado.

Velocidade de Feedback

Se você quer testar uma nova funcionalidade com usuários reais antes de um lançamento completo, o Canary é a escolha óbvia para obter feedback rápido e controlado.

Complexidade das Mudanças

Se a nova versão envolve grandes alterações no banco de dados ou em APIs críticas, o Blue-Green oferece a segurança de testar o ambiente completo antes de expor aos usuários.

Recursos de Infraestrutura

Blue-Green requer o dobro de recursos temporariamente. Rolling Update e Canary são mais econômicos, mas oferecem diferentes níveis de controle e segurança.

Considere a criticidade da sua aplicação. Para um sistema de saúde que não pode ter um segundo de inatividade, um Blue-Green ou Canary pode ser preferível, apesar do custo. Para uma aplicação interna com menor impacto, um Rolling Update pode ser perfeitamente adequado e mais econômico. Pense também na velocidade de feedback que você precisa. Se você quer testar uma nova funcionalidade com usuários reais antes de um lançamento completo, o Canary é a escolha óbvia.

Outro fator importante é a complexidade das mudanças. Se a nova versão envolve grandes alterações no banco de dados ou em APIs críticas, o Blue-Green pode oferecer a segurança de testar o ambiente completo antes de expor aos usuários. Lembre-se que a escolha do padrão é uma decisão estratégica que impacta diretamente a confiabilidade, a agilidade e a reputação da sua aplicação.

Garantindo Zero Downtime e Rollbacks Seguros

Pilares Fundamentais

Observabilidade

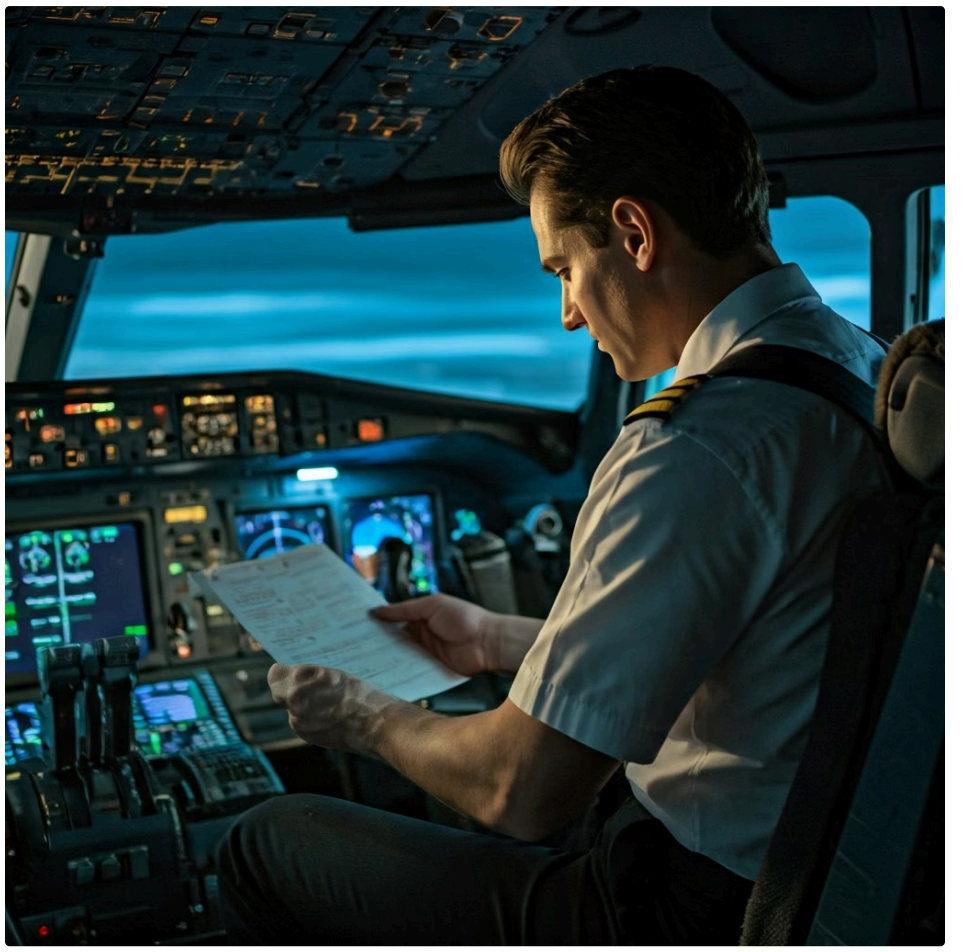
Monitorar métricas de performance, logs de erro e rastreamento de requisições em tempo real

Automação

Pipelines de CI/CD garantem deploys padronizados, testados e consistentes

Plano de Rollback

Processo bem definido, testado e automatizado para reverter mudanças rapidamente



> **Analogia do Piloto:** Um piloto não apenas sabe como decolar e voar, mas também tem um plano detalhado para emergências e sabe como pousar com segurança em qualquer situação.

Independentemente do padrão de deploy escolhido, a busca por zero downtime e a capacidade de realizar rollbacks seguros são objetivos primordiais. Para alcançar isso, é fundamental ir além da simples escolha do padrão e investir em práticas e ferramentas complementares que fortalecem todo o ciclo de vida do deploy. A observabilidade, por exemplo, é a sua linha de frente. Monitorar métricas de performance, logs de erro e rastreamento de requisições em tempo real permite detectar problemas na nova versão antes que afetem um grande número de usuários.

A automação é outro pilar inegociável. Pipelines de CI/CD (Integração Contínua/Entrega Contínua) garantem que cada deploy seja padronizado, testado e executado de forma consistente, reduzindo a chance de erros manuais. Além disso, ter um plano de rollback bem definido e testado é tão importante quanto o próprio deploy. Saber exatamente como reverter para a versão anterior em caso de falha, e ter essa reversão automatizada, é a garantia final de segurança.

O Papel da CI/CD e GitOps na Automação de Deploys

A automação é a espinha dorsal de qualquer estratégia de deploy moderna. É aqui que as práticas de CI/CD (Continuous Integration/Continuous Delivery) e, mais recentemente, o GitOps, se tornam indispensáveis. Um pipeline de CI/CD automatiza desde a compilação do código e a execução de testes unitários e de integração (CI) até a construção de imagens de contêiner e a implantação em ambientes de teste e produção (CD). Isso garante que cada mudança de código passe por um processo rigoroso e consistente antes de chegar aos usuários.

Commit no Git

Desenvolvedor faz push do código

CI Pipeline

Testes, build e criação de imagem

GitOps Sync

Ferramenta monitora repositório

Deploy Kubernetes

Estado sincronizado automaticamente

GitOps: A Evolução da Automação

O GitOps eleva essa automação a um novo patamar. Ele propõe que o Git seja a única fonte de verdade para a infraestrutura e as aplicações. Em vez de aplicar comandos diretamente no cluster Kubernetes, você declara o estado desejado do seu ambiente em arquivos YAML no Git. Ferramentas de GitOps (como Argo CD ou Flux CD) monitoram o repositório Git e, sempre que há uma mudança, elas sincronizam automaticamente o estado do cluster para corresponder ao que está declarado no Git.

Benefícios do GitOps

- **Auditabilidade completa:** Cada mudança é um commit no Git
- **Versionamento:** Infraestrutura como código versionado
- **Recuperação simplificada:** Rollback via Git revert
- **Colaboração:** Pull requests para mudanças de infra

Ferramentas Populares

- **Argo CD:** GitOps declarativo para Kubernetes
- **Flux CD:** Operador GitOps nativo do Kubernetes
- **Jenkins X:** CI/CD com GitOps integrado
- **Tekton:** Framework de CI/CD cloud-native

Essa abordagem traz inúmeros benefícios: auditabilidade completa (cada mudança é um commit no Git), versionamento da infraestrutura, recuperação de desastres simplificada e um fluxo de trabalho mais seguro e colaborativo. É como ter um "controle de versão" para todo o seu ambiente de produção, permitindo que você rastreie, reverta e colabore em mudanças de infraestrutura com a mesma facilidade que faz com o código da aplicação.

Tendências e Considerações Avançadas (2025)

O cenário de deploy está em constante evolução, impulsionado por novas tecnologias e a busca incessante por mais agilidade e resiliência. Para 2025, algumas tendências e considerações avançadas se destacam, moldando o futuro das estratégias de deploy.

Progressive Delivery

Combina Canary Release com feature flags para controle granular de funcionalidades por usuário

IA/ML em Deploys

Algoritmos preveem falhas, otimizam escalonamento e automatizam rollbacks baseados em anomalias

Serverless Patterns

Adaptação dos padrões para funções serverless com gateways de API e ferramentas cloud-native

Progressive Delivery

A "Progressive Delivery" é uma delas, que vai além do Canary Release ao combinar a liberação gradual com "feature flags" (chaves de funcionalidade). Isso permite que você não apenas libere uma nova versão para um subconjunto de usuários, mas também ative ou desative funcionalidades específicas para esses usuários, oferecendo um controle ainda mais granular sobre a experimentação.

Inteligência Artificial

Outra área de inovação é a aplicação de Inteligência Artificial e Machine Learning em deploys. Algoritmos podem analisar métricas de performance e logs para prever falhas antes que aconteçam, otimizar o escalonamento durante um Rolling Update ou até mesmo automatizar rollbacks com base em anomalias detectadas. Isso transforma o deploy de um processo reativo em um processo proativo e inteligente.

Superando Desafios Comuns em Deploys Complexos

Mesmo com os padrões e ferramentas mais sofisticados, o deploy de aplicações complexas apresenta desafios inerentes que precisam ser abordados. Um dos maiores é a gestão de mudanças de esquema de banco de dados. Um deploy sem downtime da aplicação pode ser inútil se o banco de dados precisar parar para uma migração.

Mudanças de Esquema de Banco de Dados

Estratégias como migrações "zero-downtime" permitem que versões antigas e novas coexistam com o mesmo esquema ou esquemas compatíveis. Processo de múltiplas etapas onde o esquema é alterado gradualmente.

- Adicionar colunas opcionais primeiro
- Migrar dados em background
- Remover colunas antigas depois

Aplicações Stateful

Gerenciar estado durante atualizações exige cuidado extra. Soluções incluem StatefulSets no Kubernetes, volumes persistentes e estratégias robustas de backup e restauração.

- StatefulSets para ordem de deploy
- Persistent Volumes para dados
- Snapshots antes de mudanças

Segredos e Configurações

Garantir que informações sensíveis sejam injetadas de forma segura e que as configurações corretas sejam aplicadas a cada ambiente é um desafio constante.

- Kubernetes Secrets e ConfigMaps
- Ferramentas como Vault ou Sealed Secrets
- Separação por namespace/ambiente

🎯 **Cultura Shift-Left:** A superação desses obstáculos exige uma combinação de planejamento cuidadoso, automação robusta, observabilidade contínua e uma cultura de "shift-left" (identificar problemas o mais cedo possível no ciclo de desenvolvimento). É um processo de aprendizado contínuo, onde cada deploy é uma oportunidade para refinar e melhorar as práticas.

Consolidação e Próximos Passos

Chegamos ao fim da nossa jornada pelos padrões de deploy no Kubernetes. Vimos que a escolha da estratégia certa – seja o gradual Rolling Update, a troca instantânea do Blue-Green Deployment ou a experimentação controlada do Canary Release – é fundamental para garantir a disponibilidade, a resiliência e a agilidade das suas aplicações. Cada padrão oferece um conjunto único de vantagens e desvantagens, e a maestria reside em saber qual aplicar em cada situação, sempre com o objetivo de entregar valor ao usuário final sem interrupções.

Em Prática: Recomendações Essenciais

01

Comece com Rolling Update

É o padrão mais simples e seguro para a maioria dos casos de uso cotidianos

02

Blue-Green para Criticidade

Use para deploys de alta criticidade ou grandes mudanças onde rollback instantâneo é essencial

03

Canary para Inovação

Utilize para lançar novas funcionalidades, testar hipóteses ou mitigar riscos em produção

04

Invista em Observabilidade

Logs, métricas e traces para monitorar deploys em tempo real e detectar problemas rapidamente

05

Automatize com CI/CD e GitOps

Garanta consistência, velocidade e segurança em todos os seus deploys

Autoavaliação

1

Qual padrão de deploy é mais adequado para introduzir uma nova funcionalidade a uma pequena porcentagem de usuários antes de um lançamento completo, minimizando riscos?

- a) Rolling Update
- b) Blue-Green Deployment
- c) Canary Release
- d) Shadow Deployment

2

No Kubernetes, qual recurso é comumente utilizado para gerenciar o Rolling Update de uma aplicação?

- a) Service
- b) Pod
- c) Deployment
- d) Ingress

3

Uma das principais vantagens do Blue-Green Deployment é:

- a) A economia de recursos de infraestrutura
- b) A capacidade de realizar um rollback quase instantâneo
- c) A liberação gradual de funcionalidades para usuários
- d) A simplicidade de implementação para aplicações stateful

4

Qual das seguintes práticas é crucial para garantir a segurança e a eficiência de qualquer padrão de deploy?

- a) Desativar o monitoramento durante o deploy para economizar recursos
- b) Realizar deploys manualmente para maior controle
- c) Investir em automação com CI/CD e observabilidade
- d) Evitar testes em produção a todo custo

5

Questão Dissertativa

Explique como o GitOps complementa os padrões de deploy no Kubernetes para melhorar a gestão da infraestrutura e das aplicações.

Gabarito

1. c) Canary Release

2. c) Deployment

3. b) Rollback instantâneo

4. c) Automação CI/CD

Próxima Aula

Aula 25

Métricas de Performance Web (Core Web Vitals)

Mergulharemos nas métricas que medem e otimizam a experiência do usuário, um complemento essencial para garantir que suas aplicações não apenas funcionem, mas também performem de forma excelente após cada deploy.

Recursos Adicionais

Documentação Kubernetes

Documentação oficial do Kubernetes sobre Deployments para aprofundar nos detalhes técnicos e configurações

Service Mesh

Artigos sobre Service Mesh (Istio/Linkerd) para entender como implementar Canary Releases avançados

Livros DevOps e SRE

Para uma visão mais ampla sobre cultura e práticas de entrega contínua e confiabilidade de sistemas

⚠️ **NOTA IMPORTANTE:** As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.