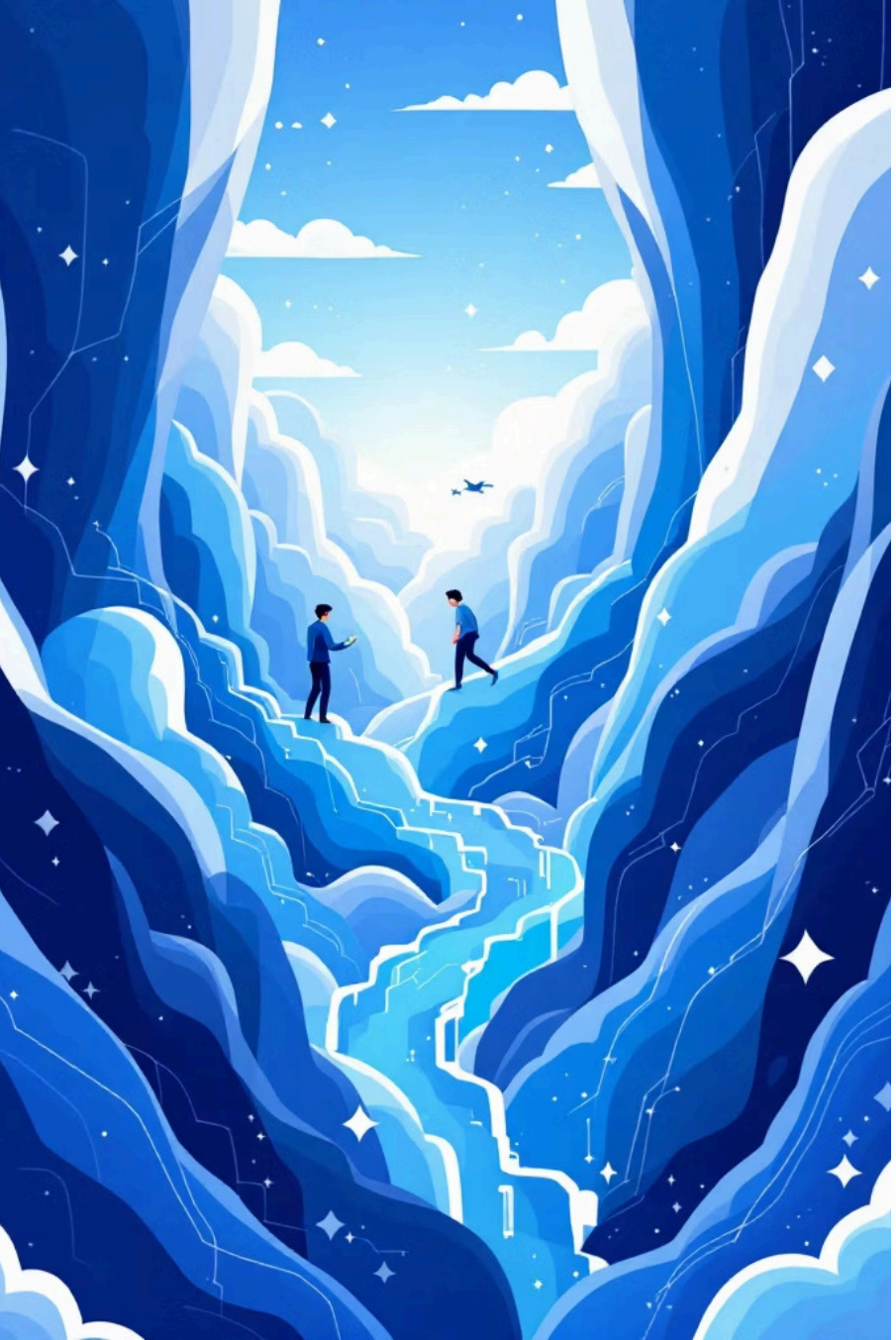


DevOps e Observabilidade: Guia Completo

Uma jornada técnica através dos conceitos fundamentais de DevOps, containerização, orquestração com Kubernetes e observabilidade de aplicações em ambientes de produção modernos.

By Prof. Cloves Alves da Rocha (**https://linktr.ee/clovesrocha**). Graduado em Gestão de Tecnologia da Informação (2014/GTI-FG) Mestre em Ciência da Computação (2017/CIn-UFPE), Professor Universitário (Graduação e Pós), Cientista de Dados (2022/Edx ***Harvard***) certificado na escola de mestres (Seminário Teológico Carisma), **Florida University of Science and Theology (FUST) Florida University of Science and Theology (FUST),** PhD Candidate in Theology, professor mestre responsável da Escola Bíblica Dominical (E.B.D. EFATÁ) da Comunidade Batista EFATÁ (@c.b.efata).

UNIDADE I

DevOps: Cultura, Conceitos e Fundamentos

DevOps representa uma mudança cultural profunda que quebra as barreiras tradicionais entre equipes de desenvolvimento e operações. Esta filosofia enfatiza a colaboração contínua, automação extensiva e feedback rápido ao longo de todo o ciclo de vida do software.

A cultura DevOps se fundamenta em princípios essenciais que transformam a maneira como organizações entregam software. O primeiro pilar é a **colaboração intensiva** entre desenvolvedores, operadores e outras partes interessadas, eliminando silos organizacionais. O segundo é a **automação** de processos repetitivos, desde testes até deployment, reduzindo erros humanos e acelerando entregas.

Outros princípios incluem **medição contínua** de métricas de desempenho e qualidade, **compartilhamento de responsabilidades** entre equipes, e **aprendizado constante** através de experimentação e análise de falhas. Esta abordagem está intimamente relacionada às Metodologias Ágeis, compartilhando valores como entregas incrementais, adaptabilidade e foco no cliente.

UNIDADE I

Pipeline CI/CD: Da Integração ao Deployment

O pipeline de Continuous Integration, Continuous Delivery e Continuous Deployment representa a espinha dorsal da automação DevOps, permitindo que código seja integrado, testado e implantado de forma consistente e confiável.

Continuous Integration (CI)

1

Desenvolvedores integram código ao repositório várias vezes ao dia. Cada commit dispara builds automatizados e execução de testes unitários, de integração e análise estática. O objetivo é detectar problemas rapidamente, quando são mais baratos de corrigir. Ferramentas como GitHub Actions executam pipelines que compilam, testam e validam cada mudança, garantindo que o código principal permaneça sempre em estado deployável.

Continuous Delivery (CD)

2

Extensão do CI que automatiza a preparação de releases para produção. Todo código que passa pelos testes é automaticamente deployado em ambientes de staging/homologação que espelham produção. Artefatos são versionados e armazenados em registries. A decisão de deploy para produção permanece manual, mas o processo está completamente automatizado e pode ser executado a qualquer momento com um clique.

Continuous Deployment (CD)

3

O passo final onde cada mudança que passa nos testes automatizados é deployada automaticamente em produção, sem intervenção humana. Requer confiança extrema na suite de testes, monitoramento robusto e capacidade de rollback rápido. Práticas como feature flags, canary releases e blue-green deployments mitigam riscos. Permite entregas múltiplas por dia com tempo mínimo entre desenvolvimento e valor entregue ao usuário.

UNIDADE I

GitHub Actions: Automação Nativa do GitHub

Por Que GitHub Actions?

Integração nativa com repositórios GitHub, eliminando necessidade de serviços externos. YAML declarativo e simples. Marketplace com milhares de actions reutilizáveis. Runners hospedados ou self-hosted. Pricing generoso para projetos open source.

Estrutura de um Workflow

Workflows são definidos em arquivos YAML no diretório `.github/workflows/`. Cada workflow contém um ou mais **jobs** que executam em paralelo ou sequencialmente. Jobs consistem em **steps** que executam actions ou comandos shell.

```
name: CI Pipeline
on: [push, pull_request]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Setup Node
        uses: actions/setup-node@v3
      - run: npm install
      - run: npm test
```

Triggers incluem eventos Git (push, PR), schedules (cron), webhooks externos ou dispatch manual. Secrets são gerenciados de forma segura e injetados como variáveis de ambiente durante execução.

UNIDADE II

Máquinas Virtuais vs Containers: Arquiteturas Comparadas

Máquinas Virtuais

Arquitetura: Cada VM inclui um sistema operacional completo, binários, bibliotecas e a aplicação. Hypervisor virtualiza hardware físico.

Isolamento: Isolamento forte a nível de hardware virtualizado. VMs são completamente independentes.

Overhead: Alto consumo de recursos. Cada VM requer GB de memória e disco. Boot lento (minutos).

Casos de Uso: Ambientes que requerem SOs diferentes, isolamento máximo, ou aplicações legacy monolíticas.

Containers

Arquitetura: Compartilham kernel do SO host. Incluem apenas aplicação e dependências. Container runtime gerencia isolamento.

Isolamento: Isolamento a nível de processo usando namespaces e cgroups do kernel Linux. Mais leve que VMs.

Overhead: Mínimo. Containers usam MB de disco e memória. Start instantâneo (segundos ou milissegundos).

Casos de Uso: Microserviços, aplicações cloud-native, ambientes de desenvolvimento consistentes, CI/CD pipelines.

A escolha entre VMs e containers não é binária. Arquiteturas modernas frequentemente combinam ambas: containers executando dentro de VMs para combinar flexibilidade e isolamento.

UNIDADE II

Docker: Construção e Publicação de Imagens

Dockerfile: Receita para Imagens

Um Dockerfile define instruções step-by-step para construir uma imagem. Cada instrução cria uma layer no sistema de arquivos em camadas do Docker, permitindo cache eficiente e reuso.

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```

FROM: Imagem base. Preferir imagens oficiais e variantes alpine (menores).

WORKDIR: Define diretório de trabalho dentro do container.

COPY: Copia arquivos do host para a imagem.

RUN: Executa comandos durante build (instalação de dependências).

CMD/ENTRYPOINT: Define comando padrão ao iniciar container.

Build e Publicação

Build de imagens usa o Docker daemon e pode ser otimizado com multi-stage builds para imagens menores:

```
# Build
docker build -t myapp:1.0 .

# Tag para registry
docker tag myapp:1.0 \
  registry.io/user/myapp:1.0

# Push para registry
docker push registry.io/user/myapp:1.0
```

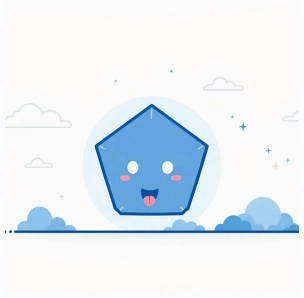
Container Registries armazenam e distribuem imagens. Opções incluem Docker Hub (público), Amazon ECR, Google Container Registry, Azure Container Registry, e Harbor (self-hosted). Registries suportam scanning de vulnerabilidades, signing de imagens, e controle de acesso granular.

Boas práticas incluem usar tags semânticas (não apenas `latest`), minimizar layers, remover caches de build, e executar containers como usuário não-root.

UNIDADE III

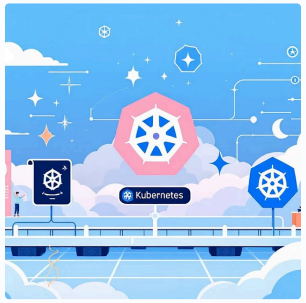
Kubernetes: Orquestração de Containers em Escala

Kubernetes (K8s) é a plataforma líder para orquestração de containers, automatizando deployment, scaling e gerenciamento de aplicações containerizadas em clusters de máquinas.



Pods

Menor unidade deployável. Agrupa um ou mais containers que compartilham rede e storage. Containers no mesmo pod comunicam via localhost e compartilham volumes.



Deployments

Gerencia ReplicaSets e Pods. Define estado desejado (quantas réplicas, qual imagem). K8s reconcilia automaticamente estado atual com desejado. Suporta rolling updates e rollbacks.

Services

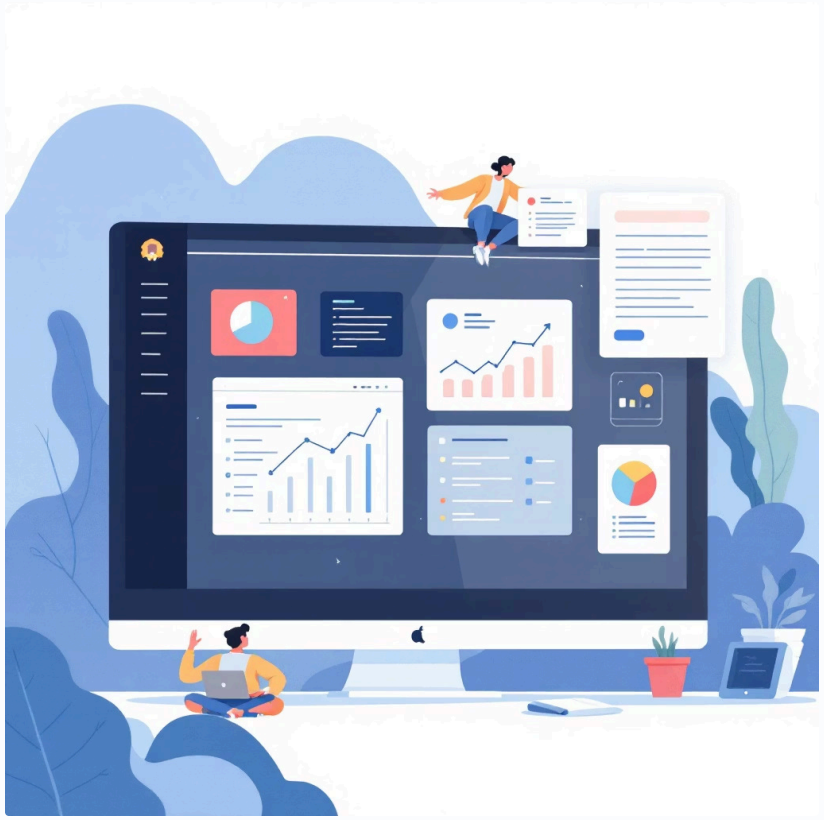
Abstração que define acesso lógico a conjunto de Pods. Tipos: ClusterIP (interno), NodePort (expõe porta no node), LoadBalancer (provisiona LB externo). Fornece service discovery e load balancing.

ConfigMaps & Secrets

Desacoplam configuração de imagens. ConfigMaps armazenam dados não-sensíveis. Secrets armazenam informações sensíveis (base64 encoded). Injetados como variáveis de ambiente ou arquivos.

UNIDADE III

Helm: Package Manager para Kubernetes



Helm simplifica deployment de aplicações complexas no Kubernetes, funcionando como um package manager que agrupa recursos K8s relacionados em pacotes reutilizáveis chamados **Charts**.

Estrutura de um Chart

```
mychart/
  Chart.yaml          # Metadados
  values.yaml         # Valores default
  templates/          # Templates K8s
    deployment.yaml
    service.yaml
    ingress.yaml
  charts/             # Dependências
```

Templates usam sintaxe Go template com placeholders para valores configuráveis. **Values.yaml** define defaults que podem ser sobrescritos via CLI ou arquivos customizados por ambiente.

Helm gerencia releases (instâncias de charts instaladas), permite upgrades atômicos com rollback automático em falhas, e facilita compartilhamento via repositórios de charts (Artifact Hub).

```
# Instalar release
helm install myapp ./mychart \
 --values prod-values.yaml

# Upgrade
helm upgrade myapp ./mychart

# Rollback
helm rollback myapp 1
```

UNIDADE IV

Stack ELK: Gerenciamento e Agregação de Logs

A stack Elastic (anteriormente ELK - Elasticsearch, Logstash, Kibana) com Beats fornece solução completa para coleta, processamento, armazenamento e visualização de logs e métricas em ambientes distribuídos.

1

Beats

Filebeat: Coleta logs de arquivos, parsing de formatos comuns (JSON, syslog). **Metricbeat:** Coleta métricas de sistemas e serviços. **Packetbeat:** Análise de tráfego de rede. **Heartbeat:** Monitoring de uptime. Agents leves instalados em cada servidor/container.

2

Elasticsearch

Motor de busca e analytics distribuído. Armazena logs em índices sharded e replicados. Query DSL poderosa baseada em JSON. Agregações complexas para análises. Escalável horizontalmente. Retention policies automáticas via Index Lifecycle Management (ILM).

3

Kibana

Interface web para visualização e análise. **Discover:** Exploração ad-hoc de logs. **Visualize:** Criação de gráficos e métricas. **Dashboard:** Painéis customizados. **Alerting:** Notificações baseadas em queries. Canvas para relatórios executivos.

Arquitetura típica: Beats → Elasticsearch ← Kibana. Para pipelines complexos de transformação, Logstash pode ser inserido entre Beats e Elasticsearch para parsing avançado, enrichment e roteamento.

UNIDADE IV

APM: Monitoramento de Performance de Aplicações

Application Performance Monitoring vai além de logs, fornecendo visibilidade profunda sobre comportamento e performance de aplicações através de métricas, traces distribuídos e análise de transações.

Métricas de Performance

Latência: Tempo de resposta de requests (p50, p95, p99). **Throughput:** Requisições por segundo. **Error Rate:** Percentual de falhas. **Saturation:** Utilização de recursos (CPU, memória, I/O). Elastic APM agents coletam automaticamente estas métricas de frameworks populares.

Distributed Tracing

Rastreia requisições através de microserviços. Cada trace contém spans representando operações. Identifica gargalos em chamadas entre serviços, queries lentas em bancos de dados, ou chamadas externas problemáticas. Contexto propagado via headers HTTP (W3C Trace Context).

Detecção de Anomalias

Machine learning identifica desvios de comportamento normal. Alertas inteligentes reduzem falsos positivos. Correlação automática entre logs, métricas e traces para root cause analysis. Kibana Machine Learning detecta patterns e prevê problemas.

Configuração envolve instrumentar aplicações com APM agents (disponíveis para Java, .NET, Node.js, Python, Ruby, Go), configurar sampling rates para controlar overhead, e criar dashboards específicos por serviço. A combinação de logs (contexto), métricas (sintomas) e traces (causas) fornece observabilidade completa.