Cultura DevOps

Da “entrega por silos” ao fluxo contínuo com qualidade, segurança e aprendizado

Aula / Workshop

Plan • Build • Run • Learn

Rodrigo Barbosa • Material para aula (≈ 4h) • Versão 2026

Agenda da aula

- Por que DevOps surgiu: contexto histórico e “dores” clássicas
- Comparativo: antes (silos) vs depois (fluxo e responsabilidade compartilhada)
- Cultura + capacidades: o que muda de verdade (não é só ferramenta)
- O ciclo DevOps: Plan • Develop • Test • CI/CD • Monitoring • Feedback
- Métricas (DORA) e melhoria contínua
- Tendências atuais: DevSecOps, GitOps, Platform Engineering e AI no delivery
- Exercícios e roteiro de implementação em etapas

O que DevOps NÃO é

- Não é um cargo ("o DevOps") — é um modo de trabalho
- Não é um conjunto fixo de ferramentas
- Não é apenas CI/CD
- Não elimina processos — muda como eles são desenhados (fluxo + feedback)
- Não é “velocidade a qualquer custo”: qualidade, estabilidade e segurança importam

De onde veio

Vias que fomentaram a cultura DevOps

As “vias” que levaram ao DevOps (visão prática)

- Agile: entregas iterativas, colaboração e ciclos curtos
- Lean: reduzir desperdício, otimizar fluxo e limitar WIP
- SRE/Operações modernas: confiabilidade como feature; automação e runbooks
- Cloud e IaC: infra versionada, replicável, provisionamento rápido
- Cultura open source: colaboração, revisão, automação e transparência
- Incidentes e custos de falhas: necessidade de reduzir MTTR e retrabalho

Antes vs Depois (o comparativo que importa)

Antes (silos)

- Dev “joga por cima do muro”
- Deploys raros e traumáticos
- Ambientes inconsistentes ("na minha máquina funciona")
- Mudanças manuais; scripts não versionados
- Feedback tardio (bugs chegam via produção)
- Métricas locais (cada área otimiza o próprio KPI)

Depois (fluxo)

- Responsabilidade compartilhada (build + run)
- Automação e padronização do delivery
- Infra como código + ambientes reproduzíveis
- Testes e segurança no fluxo (shift-left)
- Observabilidade e feedback rápido
- Métricas de sistema (lead time, CFR, MTTR, DF)

O “grande insight”: otimizar o sistema, não o time isolado

- Entrega é uma cadeia: Planejar → Construir → Validar → Entregar → Operar → Aprender
- Gargalo muda com frequência (ambiente, teste, aprovação, deploy, incidentes)
- Automação sem cultura vira “fábrica de problemas”
- Cultura sem engenharia vira “boa intenção” sem escala
- A meta: reduzir tempo de feedback + reduzir risco por mudança

Cultura DevOps

Princípios, comportamentos e capacidades

Princípios práticos da cultura DevOps

- Colaboração: time multifuncional, objetivos compartilhados
- Ownership: quem constrói também se importa com operação e qualidade
- Automação: reduzir trabalho manual repetitivo (e erros humanos)
- Padronização: caminhos “padrão ouro” para delivery, observabilidade e segurança
- Aprendizado contínuo: incidentes viram melhoria do sistema (blameless)
- Transparência: métricas e visibilidade para decidir e priorizar

Três fluxos (modelo didático)

- Flow: entregar valor com lead time baixo e previsível
- Feedback: detectar problemas cedo (testes, observabilidade, usuários)
- Learning: experimentação, melhoria contínua e resiliência organizacional

Antipadrões comuns (e como reconhecer)

- “DevOps = time de ferramentas” (vira novo silo)
- Pipelines “verdes” sem testes relevantes (falso positivo)
- Observabilidade tardia (logs sem correlação, sem SLOs)
- Mudanças sem controle de risco (sem canary, sem rollback)
- Segurança como etapa final (DevSecOps só no nome)
- Excesso de aprovações manuais (gargalos invisíveis)

O Ciclo DevOps

Plan → Develop → Test → CI/CD → Monitoring → Feedback

Plan

Priorizar valor, reduzir incerteza, desenhar o trabalho

Planejamento em DevOps: o que muda?

- Planejar com foco em fluxo: lotes pequenos, entregas frequentes
- Definir “Definition of Ready/Done” com operação e segurança
- Arquitetura evolutiva: decisões reversíveis quando possível
- Qualidade e risco entram cedo (observabilidade, SLOs, threat modeling)
- Trabalho visível: backlog, WIP, blockers e dependências claras

Artefatos úteis (rápidos) para reduzir retrabalho

Produto/Negócio

- Roadmap por objetivos (OKRs)
- Histórias pequenas + critérios de aceitação
- Mapeamento de jornada do usuário
- SLIs/SLOs orientados ao usuário

Engenharia/Operação

- ADR (Architecture Decision Record) para decisões chave
- Threat modeling leve (STRIDE) para fluxos críticos
- Plano de release (feature flags, rollout gradual)
- Runbook mínimo + estratégia de rollback

Atividade: transformar uma feature em entrega segura

- Escolha uma feature simples (ex: cadastro de usuário / pagamento / webhook).
- Defina critérios de aceitação + 1 métrica de sucesso (SLI).
- Liste riscos técnicos (deploy, dados, segurança, performance).
- Proponha 1 estratégia de rollout (canary, blue/green, feature flag).

Entrega esperada: Uma mini-especificação (1 slide/1 página) com SLI e plano de release

Develop

Código pronto para produção desde o início

Práticas de desenvolvimento para fluxo contínuo

- Trunk-based development (quando possível) + branches curtas
- Code review com foco em risco e clareza (não burocracia)
- Qualidade automatizada: lint, format, análise estática
- Dependências rastreáveis (SBOM quando aplicável)
- Infra como código ao lado do app (revisado e versionado)

Padrões de design que ajudam DevOps

Arquitetura/Serviços

- Config por ambiente (12-factor)
- Idempotência em operações críticas
- Health checks e readiness/liveness
- Backward compatibility (APIs e schema)

Dados

- Migrações versionadas (sem “alter manual”)
- Expansão/contração de schema (zero-downtime)
- Seeds e dados de teste reproduzíveis
- Observabilidade de queries e latência

GitOps (conceito rápido)

- Git como fonte de verdade para configuração e infra
- Mudanças via PR + revisão + auditoria
- Reconciliação automática (estado desejado vs real)
- Rollback “por commit” (mais previsível)
- Funciona bem com Kubernetes, IaC e plataformas internas

Test

Testar cedo, testar sempre, reduzir custo do defeito

Pirâmide de testes (e o que dá errado na prática)

- Unitários: rápidos, isolados, alta cobertura de lógica
- Integração: contratos, DB, filas, serviços externos (com controle)
- E2E: poucos, focados no caminho crítico (mais lentos)
- Teste não é só funcional: performance, segurança, resiliência
- Flaky tests destroem confiança — trate como incidente

Testes e qualidade: “shift-left” e “shift-right”

Shift-left (antes do deploy)

- SAST, lint, secrets scanning
- Testes unitários e integração
- Validação de IaC (policy-as-code)
- Build reproduzível + artefatos versionados

Shift-right (após o deploy)

- Canary + análise automática
- Synthetics e testes em produção
- Feature flags para experimentos
- Chaos engineering (com maturidade)
- Feedback de usuário + observabilidade

Atividade: desenhar uma estratégia de testes por risco

- Escolha um sistema (monólito, microserviço, API).
- Liste 3 riscos (ex: pagamentos duplicados, vazamento, indisponibilidade).
- Associe cada risco a testes (unit/integration/e2e/perf/sec).
- Defina o que deve ser “bloqueador” no pipeline.

Entrega esperada: Uma matriz risco → testes → gate no CI

CI/CD

Automação do delivery com segurança e previsibilidade

CI bem-feito: objetivo e componentes

- Integração frequente (commit pequeno) e build automático
- Testes automatizados e relatórios rápidos
- Artefatos imutáveis (build uma vez, promova entre ambientes)
- Versionamento semântico / tags; rastreabilidade
- Pipeline como código (revisável)

CD bem-feito: reduzir risco por mudança

- Deploy automatizado com etapas padronizadas
- Estratégias: rolling, blue/green, canary, shadow
- Feature flags para desacoplar deploy de release
- Rollback rápido e testado (não improvisado)
- Aprovação humana onde faz sentido (mudanças de alto risco)

Segurança no pipeline (DevSecOps em fluxo)

Automatize

- SAST/DAST e scanners de dependências
- Secrets scanning (pre-commit + CI)
- Assinatura/verificação de artefatos
- Policy-as-code (ex: OPA) para IaC
- Gates por criticidade + exceções rastreadas

Padronize

- Pipelines “golden path”
- Templates e bibliotecas internas
- Threat modeling leve para mudanças grandes
- Rotina de patches e SLOs de vulnerabilidade
- Cultura: segurança como responsabilidade compartilhada

Atividade: montar um pipeline mínimo “prod-ready”

- Defina as etapas do pipeline (build → test → scan → package → deploy).
- Marque o que é obrigatório (gate) e o que é sinal (report).
- Escolha uma estratégia de deploy (canary/blue-green/rolling).
- Desenhe o rollback e o critério de “abort” do rollout.

Entrega esperada: Um diagrama do pipeline + critérios de gate/rollback

Monitoring

Observabilidade e confiabilidade para operar com velocidade

Monitoring vs Observability (sem dogma)

- Monitoring: acompanhar métricas/limites conhecidos (CPU, erro, latência)
- Observability: entender o “porquê” com dados (logs, métricas, traces)
- Pergunta central: “o usuário está bem?” (SLIs)
- Alertas acionáveis: menos ruído, mais contexto
- Dashboards para operação e para produto (com objetivos diferentes)

SLOs, Error Budgets e práticas SRE (resumo)

Como definir

- Escolha SLIs orientados ao usuário (latência, erro, disponibilidade)
- Defina SLOs realistas e negociados
- Use error budget para guiar decisões (velocidade vs estabilidade)

Como operar

- On-call e runbooks com responsabilidade clara
- Postmortem sem culpa (blameless) + ações no backlog
- Game days e simulações com maturidade
- Automação de resposta (quando possível)

Quatro sinais clássicos (úteis para triagem)

- Latência: p95/p99 (não só média)
- Tráfego: req/s, throughput e filas
- Erros: taxa de erro por rota/serviço
- Saturação: CPU, memória, I/O, conexões, pools

Atividade: desenhar observabilidade para uma API crítica

- Escolha uma rota crítica (ex: POST /payments).
- Defina 2 SLIs (latência e erro) e 1 SLO.
- Liste logs essenciais (ids, traceId, status, tempo, erro).
- Defina 2 alertas acionáveis + 1 dashboard mínimo.

Entrega esperada: SLIs/SLOs + alertas + dashboard mínimo

Feedback

Aprender rápido e evoluir o sistema

Feedback loops que aceleram (sem quebrar)

- Do usuário: métricas de produto, suporte, NPS, comportamento
- Do sistema: observabilidade, incidentes, capacity
- Do código: testes, code review, qualidade estática
- Do processo: lead time, retrabalho, gargalos, WIP
- Experimentação: A/B, feature flags, análises pós-release

Postmortem eficaz (blameless) — estrutura mínima

- Linha do tempo: o que aconteceu (fatos)
- Impacto: quem foi afetado e como
- Detecção: como percebemos (e quanto tempo levou)
- Resposta: ações tomadas e por quê
- Causa(s): técnicas e sistêmicas (processo, comunicação, gaps)
- Ações: preventivas, detectivas e corretivas (com dono e prazo)

Métricas (DORA)

Medir para melhorar: desempenho de entrega

Métricas DORA: 4 chaves + 5ª métrica (2024)

Métrica	O que mede	Por que importa
Deployment Frequency	Frequência de deploys em produção	Indica capacidade de entregar valor continuamente
Lead Time for Changes	Tempo do commit até produção	Mostra velocidade real do fluxo (e gargalos)
Change Failure Rate	% de deploys que causam falha/incidente	Conecta velocidade com qualidade/estabilidade
Time to Restore Service	Tempo para recuperar de falhas (MTTR)	Mostra resiliência e qualidade de resposta a incidentes
Deployment Rework Rate	% de deploys não planejados após incidentes	Evidencia retrabalho e reforça a visão de instabilidade

Como usar métricas sem “gaming”

- Métricas são para melhorar o sistema, não para punir pessoas
- Use tendências (trend) — não “ranking” entre times
- Combine métricas de velocidade e estabilidade
- Conecte com objetivos de negócio (tempo de resposta ao mercado)
- Faça revisões periódicas: o que vamos mudar no processo?

Tendências atuais

O que está em alta no delivery moderno

Platform Engineering (por que apareceu?)

- Times de produto querem foco em entrega — não em “montar infra” sempre do zero
- Plataformas internas criam “golden paths” (padrões) e reduzem fricção
- Self-service: provisionar e operar com segurança e guardrails
- Métricas de plataforma: adoção, tempo de provisionamento, satisfação do dev
- Cuidado: plataforma ruim vira burocracia; plataforma boa acelera

AI no fluxo de entrega (pragmático)

- Ajuda em tarefas: boilerplate, testes, documentação, triagem de incidentes
- Risco: acelerar “produção de mudanças” sem aumentar qualidade do sistema
- Melhor uso: reforçar padrões (templates), revisão, detecção e automação
- Governança: dados sensíveis, revisão humana e rastreabilidade

DevSecOps e supply chain de software

- Shift-left: segurança integrada desde o planejamento
- Supply chain: dependências, builds, artefatos e deploys rastreáveis
- SBOM, assinatura de artefatos e políticas automatizadas ganham força
- Segurança de runtime: hardening, least privilege, secret management
- Resposta a incidentes: playbooks, correções rápidas, lições aprendidas

Como implementar

Roteiro incremental (sem big-bang)

Roteiro em 6 passos (incremental)

- 1) Tornar fluxo visível: mapear value stream e gargalos
- 2) Padronizar o básico: repo template, lint, testes mínimos
- 3) CI confiável: build/test/scan e artefatos imutáveis
- 4) CD seguro: rollout gradual + rollback + feature flags
- 5) Observabilidade: SLIs/SLOs, logs/métricas/traces e alertas acionáveis
- 6) Loop de melhoria: postmortems, métricas DORA e revisão contínua

O que priorizar primeiro (alto impacto, baixo risco)

Quick wins (1–4 semanas)

- Pipeline mínimo com testes
- Lint/format e checks no PR
- Infra como código (ambiente dev/stage)
- Logs estruturados + correlação básica
- Runbook mínimo para incidentes

Próximos níveis (1–3 meses)

- Canary + automação de rollback
- SLOs e error budgets
- Policy-as-code e templates padrão-ouro
- GitOps (se fizer sentido)
- Plataforma interna / portal de dev

Atividade final: plano DevOps para um time real

- Escolha um cenário (ex: fintech com API de pagamentos).
- Descreva “antes” (dores e gargalos) e defina 2 objetivos.
- Escolha 3 iniciativas (CI, CD, observabilidade, segurança, plataforma).
- Defina métricas de sucesso (DORA + 1 métrica de produto).
- Monte um roadmap de 30/60/90 dias.

Entrega esperada: Roadmap 30/60/90 com métricas e trade-offs

Checklist de maturidade (rápido)

- Conseguimos colocar uma mudança pequena em produção em horas/dias?
- Temos rollback simples e praticado?
- O time sabe se o usuário está bem (SLIs) em minutos?
- Incidentes geram ações concretas e priorizadas?
- Pipeline é confiável (não “quebra toda hora”)?
- A segurança está integrada no fluxo (não é só auditoria)?

Referências (principais)

- DORA — Accelerate State of DevOps Report (edição 2024) e guias de métricas
- CD Foundation — iniciativas e ecossistema de entrega contínua
- Práticas SRE (SLIs/SLOs, error budgets, postmortems) — Google SRE book (conceitos)
- DevSecOps — segurança integrada ao SDLC (conceitos e boas práticas)
- Platform Engineering — padrões, self-service e “golden paths”