# Vídeo 2 – Práticas do DevOps

## Parte 1 – Problema: Processos manuais e lentos

Pontos-chave:

* - Provisionamento manual, testes tardios e deploys arriscados.
* - Alto custo operacional e tempo de ciclo elevado.
* - Erros humanos frequentes e ambientes inconsistentes.

Diagrama: `diagramas/v2\_p1\_gargalos\_sem\_praticas.puml`

### Texto de fala:

Quando tudo é manual, o time gasta energia com tarefas repetitivas, criando filas e atrasos. Testes no fim do ciclo revelam problemas tarde demais, tornando correções caras e estressantes.

Ambientes diferentes entre si geram o famoso 'na minha máquina funciona'. É preciso padronizar e automatizar para ganhar previsibilidade e ritmo.

## Parte 2 – Discussão: Por que práticas DevOps são essenciais

Pontos-chave:

* - Mercado exige cadência alta com qualidade.
* - Padronização reduz variabilidade e erros.
* - Colaboração elimina handoffs e gargalos.

Diagrama: `diagramas/v2\_p2\_por\_que\_praticas.puml`

### Texto de fala:

Práticas DevOps alinham pessoas, processo e tecnologia para sustentar entregas frequentes sem sacrificar a confiabilidade.

Ao padronizar e automatizar, removemos esperas desnecessárias e criamos um fluxo onde cada etapa agrega valor e entrega feedback.

## Parte 3 – Solução: Infraestrutura como Código (IaC)

Pontos-chave:

* - Ambientes reprodutíveis e versionados.
* - Ferramentas: Terraform, Ansible.
* - Revisões de mudança como code review.

Diagrama: `diagramas/v2\_p3\_iac.puml`

### Texto de fala:

IaC transforma servidores e redes em código revisável, testável e versionado. Isso reduz drift e facilita rollback.

Com Terraform e Ansible, criamos, alteramos e destruímos ambientes com segurança e rastreabilidade.

## Parte 4 – Solução: Arquitetura de Microsserviços

Pontos-chave:

* - Desacoplamento e deploy independente.
* - Escalabilidade seletiva por domínio.
* - Resiliência: falha isolada a um serviço.

Diagrama: `diagramas/v2\_p4\_microservicos.puml`

### Texto de fala:

Microsserviços permitem evoluir partes do sistema sem travar o todo, reduzindo o blast radius de mudanças.

Com limites bem definidos e contratos claros, cada serviço escala conforme sua demanda e seu ciclo de vida.

## Parte 5 – Solução: Integração Contínua (CI) e Entrega Contínua (CD)

Pontos-chave:

* - Testes automáticos a cada commit e pacote sempre pronto.
* - Ferramentas: Jenkins, GitHub Actions, GitLab CI.
* - Políticas: qualidade mínima e gates de segurança.

Diagrama: `diagramas/v2\_p5\_cicd.puml`

### Texto de fala:

CI valida cada mudança cedo e com frequência, evitando bola de neve de bugs. CD mantém o software sempre implantável.

Isso reduz lead time e cria confiança para lançar com mais cadência, apoiado por gates automáticos de qualidade e segurança.

## Parte 6 – Solução: Monitoramento e Logs

Pontos-chave:

* - Observabilidade: métricas, logs e traces.
* - Alertas proativos e SLOs.
* - Ferramentas: Datadog, Splunk, Prometheus.

Diagrama: `diagramas/v2\_p6\_monitoramento.puml`

### Texto de fala:

Sem observabilidade, estamos voando às cegas. Métricas e logs bem instrumentados revelam gargalos e evitam incidentes.

Dashboards e alertas baseados em SLOs trazem foco ao que realmente importa para o usuário final.

## Parte 7 – Solução: Comunicação e Colaboração

Pontos-chave:

* - Transparência e alinhamento contínuo.
* - Ferramentas de colaboração e ritual ágil.
* - Cultura de aprendizado e pós‑mortem sem culpa.

Diagrama: `diagramas/v2\_p7\_comunicacao.puml`

### Texto de fala:

Ferramentas como Jira/Boards, chat e docs vivos encurtam distâncias e reduzem handoffs.

Pós‑mortems sem culpa transformam incidentes em aprendizado, fortalecendo o time e o produto.