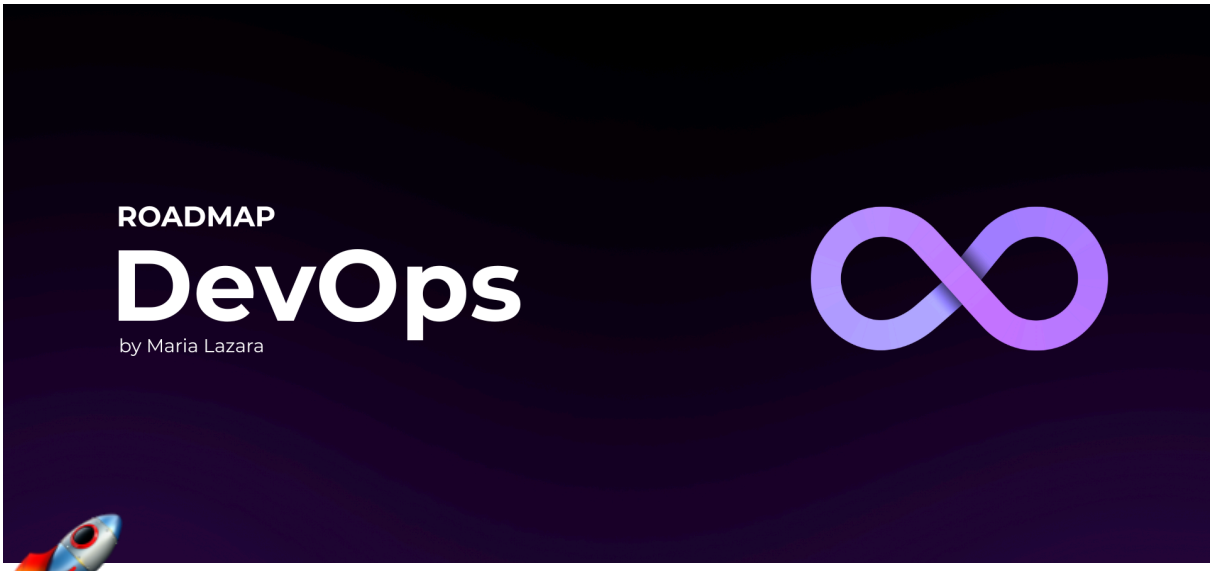

Roadmap DevOps para Iniciantes

By Maria Lazara - Guia Completo para Migração de Carreira - Do Zero ao DevOps

Oi! Eu sou a **Maria Lazara**. Fiz a transição de carreira para DevOps começando totalmente do zero, e hoje atuo com nuvem, automação e infraestrutura como código. Criei este roadmap para te ajudar a trilhar esse caminho com mais **clareza, confiança e leveza**. Você não está sozinha(o)!

Se quiser trocar ideias, tirar dúvidas ou simplesmente conversar sobre sua jornada, aqui estão as formas de me encontrar:

- 🖥️ **YouTube**
- 💼 **LinkedIn**
- 💬 **Grupo de Networking no WhatsApp:** Participe da minha comunidade

No grupo, compartilhamos **dicas, oportunidades e aprendizados**, criando um espaço de apoio e crescimento para quem está começando na área.

Sinta-se à vontade para me chamar. Estou aqui pra caminhar junto com você nessa jornada! 🚀

Introdução

Se você nunca trabalhou com tecnologia e está pensando em migrar de carreira para DevOps, este guia foi feito especialmente para você! Vou explicar tudo de forma simples e direta, sem termos técnicos complicados.

> **DevOps não é uma profissão de entrada em TI**. É como querer ser chef de cozinha sem nunca ter cozinhado um ovo. Você precisa primeiro entender como funciona o desenvolvimento de software para depois automatizar esses processos.

📘 Como usar este roadmap

Aqui, você vai encontrar **fases organizadas por nível de dificuldade**, com o que estudar, como estudar e dicas práticas em cada etapa.

Como funciona:

- **Cada fase representa um estágio da sua evolução** como futura pessoa DevOps.
- Começamos pelo básico (como funciona a internet) e avançamos até ferramentas de automação, cloud, segurança e monitoramento.
- Não é preciso ter pressa! Vá no seu ritmo e respeite seu tempo.
- Você pode seguir o cronograma sugerido ou adaptar conforme sua rotina.

> Pense nisso como uma trilha de montanha: você vai subir degrau por degrau. Cada passo importa — e nenhum

> conhecimento é “bobo” ou inútil.

Ao longo do caminho, priorize a **prática**, conecte-se com comunidades e monte um portfólio com tudo o que aprender. No fim das contas, o que mais importa não é decorar termos técnicos, mas entender como as peças se conectam e **ganhar confiança na prática**.

Agora sim: bora começar do início! 👇

O que é DevOps?

Imagine que uma equipe está construindo um aplicativo. De um lado, os **desenvolvedores** escrevem o código e criam novas funcionalidades. Do outro, o **time de operações** cuida da infraestrutura, da segurança, dos servidores — e garante que tudo esteja funcionando bem.

Antigamente, esses dois times trabalhavam separados. O time de desenvolvimento entregava o código, e o de operações tentava “dar um jeito” de colocar no ar. Isso gerava **muitos problemas**: atrasos, erros, retrabalho.

O DevOps surgiu justamente pra resolver isso. É uma cultura (com ferramentas e práticas) que une desenvolvimento e operações, promovendo **colaboração, automação e entregas mais rápidas e confiáveis**.

> 💡 DevOps é como construir uma ponte sólida entre quem cria o software e quem coloca ele pra rodar, garantindo que tudo aconteça de forma integrada, segura e contínua.

FASE 1: PREPARAÇÃO

Essa fase é **essencial** para quem **está começando totalmente do zero**. Aqui, você vai construir a base que vai te ajudar a entender o que vem depois.

Mesmo que você não vire programador(a), é importante conhecer os fundamentos do mundo da tecnologia.

1. ENTENDA O MUNDO DO SOFTWARE

O que você vai aprender:

- Como sites e aplicativos são criados
- Como equipes de programadores trabalham juntas
- O que acontece quando você clica em um botão no seu celular

Conceitos importantes:

- **Frontend:** é a parte visual com a qual o usuário interage (como botões, menus, telas).
- **Backend:** é o que acontece por trás (como processar um pedido ou buscar informações no banco de dados).
- **APIs:** são como “pontes” que permitem que diferentes sistemas se comuniquem.
- **Ciclo de vida do software:** como um projeto nasce, é desenvolvido, testado e entregue.

Como estudar:

- Assista vídeos no YouTube com títulos como “Como funciona a internet” ou “Como sites são criados”.
- Faça um curso gratuito de HTML/CSS apenas para entender a estrutura de uma página web.
- Leia sobre **metodologias ágeis**, como Scrum e Kanban, que mostram como os times se organizam para entregar software em etapas curtas e eficientes.

🔍 **Checklist de ferramentas e conceitos:**

☐ **HTML** – estrutura das páginas web

☐ **CSS** – aparência/estilo das páginas

☐ **JavaScript (básico)** – lógica no lado do usuário (frontend)

☐ **GitHub** – onde os códigos são hospedados

☐ **API (REST/JSON)** – comunicação entre sistemas

☐ **Metodologias Ágeis** – Scrum, Kanban

☐ **Ferramentas de time ágil** – Trello, Jira

☐ **Frontend vs Backend** – diferença entre o que o usuário vê e o que roda por trás

☐ **CI/CD (visão geral)** – automação do ciclo de entrega de software

---

2. APRENDA O BÁSICO DE LÓGICA DE PROGRAMAÇÃO

Por que é importante:

Você não vai ser ser programador, mas vai **automatizar tarefas com scripts** — que são pequenos trechos de código. Por isso, precisa **entender a lógica** por trás do funcionamento de um programa.

O que estudar:

- **Variáveis:** onde guardamos valores (como nome, número, senha).
- **Condicionais (if/else):** permitem tomar decisões (“se isso acontecer, faça aquilo...”).
- **Loops (while/for):** repetem ações até que uma condição seja atendida.
- **Funções:** blocos de código que executam tarefas específicas.

Linguagem recomendada:

- Comece com **Python**, por ser simples e muito usado em automações.

Como praticar:

- Faça exercícios simples nos sites HackerRank ou Codecademy.

🔍 **Checklist de ferramentas e conceitos:**

☐ **Python** – linguagem fácil e usada em automações

- [ ] **Variáveis, condições, loops, funções** – fundamentos da lógica
- [ ] **IDEs simples** – Replit, VS Code
- [ ] **Exercícios interativos** – HackerRank, Codecademy, Exercism
- [ ] **Conceito de scripts** – automatizar tarefas repetitivas
- [ ] **Boas práticas básicas** – indentação, nomeação clara de variáveis
- [ ] **Debug básico** – identificar e corrigir erros simples
- [ ] **Entrada e saída de dados** – `input()`, `print()`

---

3. ENTENDA COMO FUNCIONA UM COMPUTADOR

Por que isso importa:

Antes de trabalhar com servidores, scripts e infraestrutura, é fundamental entender como computadores funcionam, como se conectam e como organizam seus dados.

Conceitos importantes:

- **Servidor:** um tipo de computador que fica ligado o tempo todo e responde a requisições (como quando você acessa um site).
- **Arquivos e diretórios:** toda a organização do sistema é feita por pastas (diretórios) e arquivos — saber navegar por eles é essencial.
- **Sistema operacional:** o programa principal que controla o computador (ex: Windows, Linux).
- **Rede:** como um computador "conversa" com outro. Conceitos como IP, internet, roteadores e DNS fazem parte disso.
- **Cliente x Servidor:** o computador do usuário (cliente) faz um pedido e o servidor responde.

Como estudar:

- Busque no YouTube: "como funciona um servidor", "o que é um sistema operacional" ou "como computadores se conectam".
- Leia guias simples ou veja infográficos que explicam o que acontece quando você digita um site no navegador.

🔍 **Checklist de ferramentas e conceitos:**

- ☐ **Sistema Operacional (Linux/Windows)** – o que é e para que serve
- ☐ **Terminal / Prompt de Comando** – comandos básicos
- ☐ **Arquivos e diretórios** – navegação, criação, estrutura
- ☐ **Servidores** – o que são e para que servem
- ☐ **Conceito de rede** – IP, roteador, internet, DNS
- ☐ **Cliente vs Servidor** – como se comunicam
- ☐ **Processos e memória** – noções básicas do que roda por trás

---

🐧 FASE 2: FUNDAMENTOS TÉCNICOS

1. APRENDA LINUX

O que é Linux:
Linux é um sistema operacional open source, muito utilizado em servidores, supercomputadores, dispositivos IoT e até em desktops. Ao contrário do Windows, Linux é baseado em comandos via terminal (linha de comando), o que traz muita flexibilidade e controle para administrar sistemas.

> 👉🏻 Aprender Linux é **fundamental** para qualquer profissional de TI, especialmente DevOps, pois a maioria dos ambientes de produção roda Linux.

O que estudar:

- **Comandos básicos do terminal:** navegação entre diretórios, manipulação de arquivos e pastas, visualização de conteúdo.
- **Permissões e propriedade de arquivos:** entender os níveis de permissão (leitura, escrita, execução) para usuário, grupo e outros.

- **Conexão remota via SSH:** aprender a conectar-se com servidore, configurar chaves públicas e privadas para autenticação segura.
- **Gerenciamento de processos:** usar comandos para monitorar e controlar programas rodando no sistema.
- **Pacotes e gerenciamento de software:** usar gerenciadores para instalar e atualizar softwares.
- **Scripting básico:** aprender shell scripts para automatizar tarefas comuns.

🔍 **Checklist de ferramentas e conceitos:**

- [ ] **Distribuição Ubuntu (ou Debian)** – ideal para começar
- [ ] **Terminal / Shell** – acesso via linha de comando
- [ ] **Comandos básicos:**
  - `ls`, `cd`, `pwd` (navegar entre pastas)
  - `mkdir`, `rm`, `touch`, `cp`, `mv` (gerenciar arquivos)
  - `chmod`, `chown` (permissões de arquivos)
- [ ] **Editor de texto no terminal:** `nano` ou `vim`
- [ ] **Gerenciamento de pacotes:** `apt` (instalar programas)
- [ ] **Processos e serviços:** `ps`, `top`, `htop`, `kill`
- [ ] **Acesso remoto via SSH:** `ssh`, `scp`
- [ ] **Sistema de arquivos:** diretórios como `/home`, `/etc`, `/var`
- [ ] **VirtualBox + ISO do Ubuntu** – para criar sua máquina virtual e praticar

---

2. APRENDA GIT

O que é Git:
Git é um sistema de controle de versões distribuído que registra todas as alterações feitas em arquivos ao longo do tempo. Ele permite que desenvolvedores trabalhem simultaneamente em um mesmo projeto,

rastreando mudanças, facilitando a colaboração e possibilitando reverter para versões anteriores se necessário.

O que estudar:

- **Commit:** registrar as mudanças feitas no código de forma organizada e com mensagens claras.
- **Branches:** trabalhar em diferentes linhas de desenvolvimento paralelas, para testar funcionalidades sem afetar a versão principal.
- **Merge e pull requests:** juntar mudanças de diferentes branches, resolver conflitos e colaborar em equipe.
- **Repositórios remotos:** usar plataformas como GitHub, GitLab ou Bitbucket para armazenar o código na nuvem, facilitar o trabalho em equipe e integração contínua.
- **Histórico e logs:** comandos para entender o que foi alterado e quando.
- **Reset e revert:** desfazer mudanças ou voltar para versões anteriores com segurança.

🔍 **Checklist de ferramentas e conceitos:**

- [ ] **Git** – ferramenta de versionamento
- [ ] **GitHub** – plataforma de hospedagem de repositórios
- [ ] **GitLab / Bitbucket** – alternativas ao GitHub
- [ ] **Principais comandos:**
  - `git init`, `git clone` (iniciar/copiar projeto)
  - `git add`, `git commit` (salvar mudanças)
  - `git push`, `git pull` (enviar e buscar mudanças)
  - `git branch`, `git checkout`, `git merge` (trabalhar com versões diferentes)
- [ ] **Arquivo** `.gitignore` – evitar que arquivos desnecessários sejam versionados
- [ ] **GitHub Desktop / GitKraken** – interfaces visuais para Git (opcional)
- [ ] **README.md** – documentação básica do projeto
- [ ] **Licenças e boas práticas de commit**

3. REDES E SEGURANÇA BÁSICA

O que aprender:

- **Funcionamento da internet:**
  - **IP:** endereço único que identifica dispositivos na rede.
  - **DNS:** sistema que traduz nomes de domínios (ex: google.com) em endereços IP.
  - **Protocolos HTTP e HTTPS:** como a informação trafega entre cliente e servidor, e a importância da criptografia para segurança (HTTPS usa TLS).
- **Firewall:** ferramenta que controla o tráfego de rede, permitindo ou bloqueando conexões com base em regras, funcionando como uma "portaria" que protege a rede contra acessos não autorizados.
- **Certificados SSL/TLS:** certificados digitais que garantem a autenticidade de um site e criptografam a comunicação entre usuário e servidor (visualizado pelo cadeado no navegador).
- **Load Balancer:** componente que distribui o tráfego de rede entre vários servidores para garantir alta disponibilidade e melhor desempenho do sistema, evitando sobrecarga em um único servidor.
- **Noções básicas de TCP/IP:** entender as camadas de rede, portas, protocolos mais usados (TCP, UDP).
- **Conceitos de VPN e proxies:** para acesso seguro e privado em redes públicas.

🔍 **Checklist de ferramentas e conceitos:**

☐ **IP (IPv4, IPv6)** – identificação de computadores na rede

☐ **DNS (Domain Name System)** – traduz nomes para IPs (ex: google.com → IP)

☐ **HTTP/HTTPS** – como navegadores trocam dados com servidores

☐ **Portas e protocolos** – ex: 80 (HTTP), 443 (HTTPS), 22 (SSH)

☐ **Firewall:**

- Conceito de filtragem de entrada/saída

- Ferramentas básicas como `ufw` (Uncomplicated Firewall)

☐ **Certificados SSL/TLS:**

- Criptografia de dados
- Ex: Let's Encrypt para emissão gratuita

☐ **Ping, traceroute, nslookup, dig** – comandos para testes de rede

☐ **Load Balancers:**

- Conceito de balanceamento de carga (ex: Nginx, HAProxy)
- Tipos: Round Robin, Least Connections

☐ **Modelo OSI (visão geral)** – camadas de rede para entender a comunicação

---

🐋 FASE 3: CONTAINERIZAÇÃO

1. APRENDA DOCKER

O que é Docker:

Docker é uma ferramenta que cria “caixinhas” para suas aplicações, garantindo que elas rodem iguais em qualquer computador.

Aprender Docker é importante para DevOps porque facilita o desenvolvimento, testes e deploys, tornando tudo mais rápido e confiável.

Por que é importante:

- Sua aplicação roda igual em qualquer lugar
- Facilita testes e deployments
- Resolve o problema "funciona na minha máquina"

O que estudar:

- Como criar containers
- Como executar aplicações em containers
- Docker Compose (para rodar várias aplicações juntas)
- Conceitos de imagens e registries
- Inspecionar containers ativos
- Networking Docker

- Persistir dados com Docker Volumes

🔍 **Checklist de ferramentas e conceitos:**

☐ **Docker Engine** — instalar e entender o daemon que cria e gerencia containers

☐ **Docker CLI** — comandos básicos: `docker run`, `docker ps`, `docker stop`, `docker rm`, etc.

☐ **Docker Images** — entender o que são, como criar (Dockerfile), usar imagens oficiais e customizadas

☐ **Dockerfile** — sintaxe básica, comandos mais comuns (`FROM`, `RUN`, `COPY`, `CMD`, `ENTRYPOINT`)

☐ **Docker Containers** — criar, executar, inspecionar, parar, reiniciar containers

☐ **Docker Compose** — arquivo `docker-compose.yml`, orquestrar múltiplos containers, comandos básicos `up`, `down`, `logs`

☐ **Docker Registry** — trabalhar com Docker Hub, repositórios privados, push/pull de imagens

☐ **Volumes Docker** — persistência de dados com volumes e bind mounts

☐ **Networking Docker** — redes bridge, host, overlay, conectar containers, expor portas

☐ **Docker Logs e Monitoramento** — comandos para inspecionar logs e status dos containers

☐ **Docker Swarm** (básico) — orquestração simples dentro do Docker (opcional para começar)

---

2. APRENDA KUBERNETES

O que é Kubernetes:
Se Docker é como um container, Kubernetes é como um porto que gerencia milhares de containers automaticamente.

O que estudar:

- Conceitos básicos: Pods, Services, Deployments
- Como escalar aplicações automaticamente
- Como gerenciar configurações e segredos
- kubectl (ferramenta de linha de comando)

🔍 **Checklist de ferramentas e conceitos:**

☐ **kubectl** — comandos básicos para interagir com o cluster (`get`, `describe`, `apply`, `delete`)

☐ **Pods** — unidade básica de execução no Kubernetes

☐ **Deployments** — gerenciar versões, atualizações e replicação de Pods

☐ **Services** — exposição e balanceamento de carga interno (ClusterIP, NodePort, LoadBalancer)

☐ **Namespaces** — organizar recursos em ambientes separados

☐ **ConfigMaps e Secrets** — gerenciar configurações e dados sensíveis

☐ **Volumes e PersistentVolumes (PV) / PersistentVolumeClaims (PVC)** — persistência de dados em clusters

☐ **Horizontal Pod Autoscaler (HPA)** — escalar Pods automaticamente baseado em métricas

☐ **Ingress** — gerenciar acesso HTTP externo e roteamento para serviços

☐ **StatefulSets e DaemonSets** — gerenciar workloads específicos (banco de dados, serviços em cada nó)

☐ **Helm** — gerenciador de pacotes para Kubernetes (charts)

☐ **Minikube / Kind** — ferramentas para rodar cluster local para aprendizado

---

☁️ FASE 4: CLOUD COMPUTING

1. ESCOLHA UMA PLATAFORMA CLOUD

O que é:

Plataformas Cloud oferecem recursos de computação, armazenamento e rede sob demanda pela internet. AWS é a mais usada, mas Google Cloud e Azure também são populares.

O que estudar:

- **IAM (Identity and Access Management):** controle de usuários, permissões e segurança
- **EC2:** servidores virtuais configuráveis e escaláveis
- **S3:** armazenamento de arquivos, backups e dados estáticos
- **VPC (Virtual Private Cloud):** redes virtuais isoladas para recursos cloud
- **Load Balancers:** distribuir tráfego entre múltiplas instâncias para alta disponibilidade

🔍 **Checklist de ferramentas e conceitos:**

☐ **AWS IAM** – gerenciamento de acesso e permissões

☐ **Amazon EC2** – instâncias de servidores virtuais

☐ **Amazon S3** – buckets de armazenamento de objetos

☐ **Amazon VPC** – redes virtuais e configuração de subnets, security groups

☐ **Elastic Load Balancer (ELB)** – balanceamento de carga para aplicações

☐ **AWS CLI** – linha de comando para gerenciar recursos AWS

☐ **AWS Console** – interface web para administração

☐ **Conhecimento básico em:** Route 53 (DNS), CloudWatch (monitoramento), CloudTrail (logs)

---

2. APRENDA INFRAESTRUTURA COMO CÓDIGO - TERRAFORM

O que é:
Terraform é uma ferramenta que permite criar e gerenciar infraestrutura usando código, garantindo automação, versionamento e facilidade de reprodução.

Ferramenta principal: Terraform

- Como provisionar infraestrutura
- Como gerenciar múltiplos ambientes (desenvolvimento, teste, produção)
- Versionamento de infraestrutura

O que estudar:

- **Provisionamento:** criar recursos cloud declarativamente (ex: EC2, S3, VPC)
- **Estado (state):** arquivo que armazena o status atual da infraestrutura
- **Variáveis e outputs:** parametrizar configurações e extrair informações
- **Módulos:** reutilizar código para facilitar a manutenção e escalabilidade
- **Workspaces:** gerenciar múltiplos ambientes (dev, test, prod)
- **Versionamento:** usar Git para controlar versões dos scripts Terraform

🔍 **Checklist de ferramentas e conceitos:**

- ☐ **Terraform CLI** – comandos básicos: `init`, `plan`, `apply`, `destroy`
- ☐ **Providers** – integração com AWS, Azure, GCP, etc.
- ☐ **Resources** – definição dos recursos de infraestrutura
- ☐ **State files** – local e remoto (ex: backend S3)
- ☐ **Módulos Terraform** – criação e utilização de módulos reutilizáveis
- ☐ **Variables e Outputs** – parametrização e extração de dados
- ☐ **Workspaces** – isolamento de ambientes distintos
- ☐ **Integração com CI/CD** (opcional) – automatizar deploys de infraestrutura
- ☐ **Versionamento Git** – gerenciar código Terraform em repositórios

---

🔄 FASE 5: CI/CD E AUTOMAÇÃO

1. ENTENDA CI/CD

O que é:
É como uma linha de produção automática para software. Quando um

programador termina uma funcionalidade, ela automaticamente passa por testes e vai para produção.

Processo típico:

1. Desenvolvedor escreve código
2. Sistema executa testes automaticamente
3. Se tudo OK, cria um package da aplicação
4. Deploys automaticamente para produção

O que estudar:

- Fluxo básico: build → test → package → deploy
- Tipos de CI/CD: Continuous Integration, Continuous Delivery e Continuous Deployment
- Ferramentas populares (Jenkins, GitLab CI, CircleCI, GitHub Actions)
- Testes automatizados: unitários, integração e e2e
- Rollbacks e monitoramento pós-deploy

🔍 **Checklist de conceitos:**

☐ **Pipeline de CI/CD** – automação de build, testes e deploy

☐ **Versionamento de código** – integração com repositórios Git

☐ **Automação de testes** – integração com pipelines

☐ **Deploy automático** – para ambientes de homologação e produção

☐ **Monitoramento e alertas** – identificar falhas rapidamente

---

2. APRENDA JENKINS

O que é:

Ferramenta de automação que executa jobs e pipelines para construir, testar e fazer deploy de aplicações.

O que estudar:

- Configuração e instalação do Jenkins
- Criar jobs freestyle e pipelines declarativos (Jenkinsfile)
- Integração com repositórios Git
- Plugins essenciais (Git, Docker, Slack, Pipeline, Credentials)
- Gerenciamento seguro de credenciais e variáveis
- Agendamento e triggers (push, pull request, timer)

🔍 **Checklist de ferramentas e conceitos:**

- ☐ **Jenkins Master e Agents** – arquitetura básica
- ☐ **Jenkinsfile** – pipeline como código (declarativo e scripted)
- ☐ **Plugins Git** – integração com SCM
- ☐ **Plugins Docker** – build e deploy de containers
- ☐ **Credenciais** – gerenciar tokens e chaves com segurança
- ☐ **Webhooks** – gatilhos automáticos via Git
- ☐ **Monitoramento de builds** – status e logs

---

3. APRENDA ARGOCD

O que é ArgoCD:
É uma ferramenta que implementa GitOps para Kubernetes. Em termos simples, você descreve como sua aplicação deve estar no Kubernetes em arquivos Git, e o ArgoCD garante que ela permaneça sempre assim.

O que estudar:

- Conceitos de GitOps e deploy declarativo
- Instalação e configuração do ArgoCD no Kubernetes
- Configuração de aplicações (AppProjects, Applications)
- Estratégias de sincronização: manual, automática, auto-prune

- Hooks (pre-sync, post-sync) e health checks customizados
- Integração com pipelines CI (ex: Jenkins, GitLab CI)
- Rollbacks automáticos e observabilidade via UI e CLI

🔍 **Checklist de ferramentas e conceitos:**

- [ ] **ArgoCD Server e Controller** – arquitetura
- [ ] **kubectl** e `argocd` CLI – comandos básicos
- [ ] **Manifestos Kubernetes** – YAML de Deployments, Services, ConfigMaps
- [ ] **Git Repositories** – configuração de repositórios Git com apps
- [ ] **Sync Policy** – definir estratégias de sincronização
- [ ] **Health Checks** – monitorar estado da aplicação
- [ ] **RBAC no ArgoCD** – controle de acesso e permissões
- [ ] **UI do ArgoCD** – acompanhar status e histórico de deploys

---

📊 FASE 6: MONITORAMENTO E OBSERVABILIDADE

1. APRENDA PROMETHEUS E GRAFANA

O que fazem:

- **Prometheus:** coleta e armazena métricas em tempo real sobre o funcionamento de aplicações e infraestrutura.
- **Grafana:** visualiza essas métricas em dashboards personalizados e interativos.

O que estudar:

- Instalação e configuração básica do Prometheus
- Exporters (ex: Node Exporter, kube-state-metrics, cAdvisor)
- Criação de dashboards no Grafana
- Alertas baseados em regras (Alertmanager)
- Principais métricas para acompanhar: CPU, memória, uso de disco, disponibilidade de serviços

- Integração com Kubernetes

🔍 **Checklist de ferramentas e conceitos:**

☐ **Prometheus** – sistema de coleta de métricas

☐ **Alertmanager** – envio de alertas (e-mail, Slack, etc.)

☐ **Exporters** – coleta de métricas (Node Exporter, cAdvisor, etc.)

☐ **Grafana** – dashboards personalizados

☐ **Dashboards predefinidos** – Grafana Labs ou community templates

☐ **Queries PromQL** – linguagem de consulta do Prometheus

☐ **Integração com Kubernetes** – monitorar clusters com Prometheus Operator

☐ **Alertas** – criação e gerenciamento de alert rules no Grafana ou Prometheus

---

2. APRENDA LOGGING (ELK Stack)

O que é:

Logs são registros automáticos de tudo o que acontece em uma aplicação ou servidor. O stack **ELK** ajuda a coletar, processar, armazenar e visualizar esses registros.

Ferramentas:

- **Elasticsearch:** banco de dados de busca para armazenar logs
- **Logstash:** processador de logs
- **Kibana:** interface para visualizar e analisar os logs

O que estudar:

- Instalação e configuração do ELK Stack
- Coleta de logs de aplicações e servidores (via Filebeat, Logstash)
- Indexação e armazenamento no Elasticsearch
- Criação de dashboards e visualizações no Kibana
- Consultas básicas e filtros para análise de logs

- Monitoramento de erros, falhas, tempo de resposta e padrões anormais

🔍 **Checklist de ferramentas e conceitos:**

☐ **Elasticsearch** – armazenamento e indexação de logs

☐ **Logstash** – pipeline de processamento de logs

☐ **Filebeat** – agente leve para enviar logs ao Logstash/Elasticsearch

☐ **Kibana** – interface de visualização e análise

☐ **Dashboards e Visualizações** – criação no Kibana

☐ **Filtros e queries** – para encontrar eventos específicos

☐ **Integração com aplicações** – enviar logs customizados

☐ **Alertas** – configurar notificações baseadas em logs (via Kibana ou Elastic integrations)

🔒 FASE 7: SEGURANÇA (DevSecOps)

1. APRENDA CONCEITOS DE SEGURANÇA

O que estudar:

- Vulnerabilidades comuns em aplicações
- Como escanear código em busca de problemas
- Gerenciamento de segredos (senhas, certificados)
- Princípios de segurança em containers

2. FERRAMENTAS DE SEGURANÇA

O que aprender:

- Snyk ou SonarQube (análise de código)
- HashiCorp Vault (gerenciamento de segredos)
- Falco (monitoramento de segurança em runtime)

🎯 CRONOGRAMA SUGERIDO

Meses 1-3: Preparação

- Lógica de programação
- Conceitos de desenvolvimento
- Python básico

Meses 4-7: Fundamentos

- Linux
- Git
- Redes básicas
- Docker

Meses 8-10: Avançado

- Kubernetes
- Cloud (AWS)
- Terraform

Meses 11-14: Automação

- CI/CD
- Jenkins
- ArgoCD
- Pipelines

Meses 15-18: Especialização

- Monitoramento
- Segurança
- Projetos práticos

💡 DICAS IMPORTANTES PARA INICIANTES

1. Não tente aprender tudo de uma vez

Aprender DevOps é como aprender a dirigir: primeiro você domina o básico, depois encara o trânsito. Foque em uma coisa de cada vez.

2. Pratique muito

Ler é importante, mas só praticando você realmente aprende. Crie pequenos projetos, experimente, erre e aprenda com isso.

3. Participe da minha comunidade no WhatsApp

Lá eu compartilho **dicas práticas, insights de carreira, oportunidades de vagas** e crio um espaço de **networking com outras pessoas que também estão começando na área de DevOps**. Aprender em grupo torna a jornada mais leve e acelerada.

- link para a comunidade: https://chat.whatsapp.com/GU4Ht0bH13761Y8Qau8T3X

4. Monte um portfólio

Crie um GitHub com seus projetos. Empregadores querem ver o que você sabe fazer.

5. Seja paciente

Esta transição leva tempo. É normal se sentir perdido no início.

🛠️ FERRAMENTAS ESSENCIAIS PARA ESTUDAR

Gratuitas:

- Visual Studio Code (editor de código)
- VirtualBox (máquinas virtuais)
- Docker Desktop
- Git
- AWS Free Tier

Recursos de Estudo:

- YouTube
- Coursera/Udemy (cursos pagos)

- Kubernetes.io (documentação oficial)
- AWS Training (cursos gratuitos)

🚀 PRÓXIMOS PASSOS

1. Escolha por onde começar

Se você nunca mexeu com tecnologia, comece pela Fase 1. Se já tem algum conhecimento, pule para onde faz sentido.

2. Crie um cronograma realista

Dedique pelo menos 2-3 horas por dia aos estudos.

3. Comece hoje

O melhor momento para plantar uma árvore foi há 20 anos. O segundo melhor momento é agora.

4. Seja consistente

É melhor estudar 1 hora todos os dias do que 8 horas uma vez por semana.

💼 PERSPECTIVAS DE CARREIRA

Caminhos possíveis:

- DevOps Engineer
- Cloud Architect
- Site Reliability Engineer (SRE)
- Platform Engineer
- DevSecOps Specialist

🎯 MENSAGEM FINAL

Migrar de carreira para DevOps é desafiador, mas totalmente possível. Muitas pessoas já fizeram essa transição com sucesso. O segredo é ter paciência, ser consistente e nunca parar de aprender.

Lembre-se: DevOps é sobre pessoas e processos, não apenas ferramentas. Desenvolva suas habilidades de comunicação junto com as técnicas.

Boa sorte na sua jornada! 🚀

Este roadmap foi criado com base na minha experiência profissional e na trajetória de pessoas que fizeram a transição com sucesso. Use-o como guia, mas adapte conforme sua realidade e objetivos.