APRESENTAÇÃO
DE APOIO

DEVOPS AVANÇADO

Marcelo Veiga Neves – Aula 03

PUCRS online uol edtech

Professores

CÁSSIO TRINDADE

Professor Convidado

Profissional da área de TI, trabalhando há mais de uma década com a formação de profissionais, dando aulas no Instituto Federal do Rio Grande do Sul, na Faculdade Dom Bosco, Universidade Luterana do Brasil (ULBRA), Pontifícia Universidade Católica do RS (PUCRS) e na TargetTrust. Atualmente atuando como Arquiteto de Software na PUCRS, sendo responsável pela condução e elaboração de mais de 90 projetos diretamente com alunos do curso de Engenharia de Software, trabalhando com as mais variadas tecnologias. Mais de 30 anos de experiência nas áreas de desenvolvimento de software, aplicativos para celuares e sistemas corporativos e para internet desde projetos de e-commerce para o Sonae Portugal e site de classificados digitais do Grupo RBS a dezenas de aplicativos mobiles.

MARCELO VEIGA NEVES

Professor PUCRS

Possui doutorado em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (2015), mestrado em Ciência da Computação na Universidade Federal do Rio Grande do Sul (2009) e graduação em Ciência da Computação pela Universidade Federal de Santa Maria (2005). Tem experiência na área de Ciência da Computação, atuando principalmente nos seguintes temas: redes de computadores, processamento de alto desempenho e sistemas embarcados.

Ementa da disciplina

Estudo sobre entrega contínua (CD), uso de contêineres, orquestração e monitoramento. Experimentação de ferramentas: GitHub Actions, Docker Compose e Kubernets e ferramentas de monitoração.

Prof. Dr. Marcelo Veiga Neves

- Formação acadêmica
  - Doutorado em Ciência da Computação pela PUCRS – 2015
    - Doutorado sanduíche no IBM T. J. Watson Center – New York
  - Mestrado em Ciência da Computação pela UFRGS – 2009
  - Graduação em Ciência da Computação pela UFSM – 2005

- Áreas de interesse/pesquisa:
  - Big data e análise de dados, Computação em nuvem, Internet das Coisas, Virtualização, Redes de computadores e HPC.

Prof. Dr. Marcelo Veiga Neves

- Experiência profissional
  - Professor da PUCRS há 8 anos
  - Experimentação com infraestruturas de TI, administração de sistemas e programação há quase 20 anos
  - Mais de 15 anos de experiência com projeto de software baixo nível e para sistemas embarcados
  - Mais de 7 anos de experiência com desenvolvimento de soluções escaláveis para nuvem
  - Empreendendo desde 2016

Conteúdo

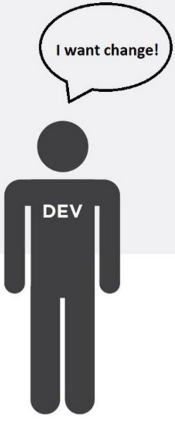
- Introdução
- Principais práticas em DevOps
- Infraestrutura automatizada e gerência de configuração
- Infraestrutura como código
- Monitoramento e Observabilidade
- Feedback contínuo e comunicação
- Considerações finais

Introdução

- O que é DevOps?

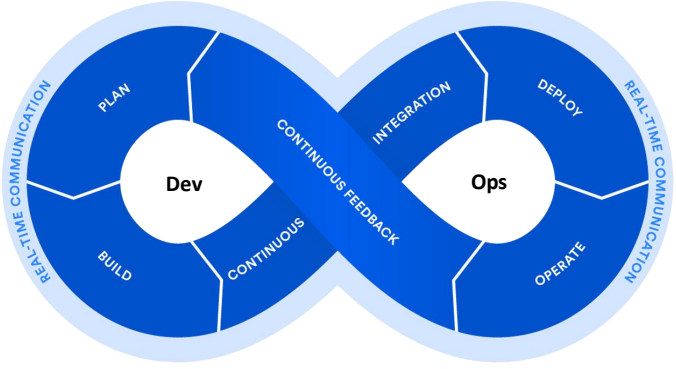
- DevOps é uma prática de engenharia de software que possui o intuito de unificar o desenvolvimento de software (Dev) e a operação de software (Ops)

- Práticas + Cultura + Ferramentas

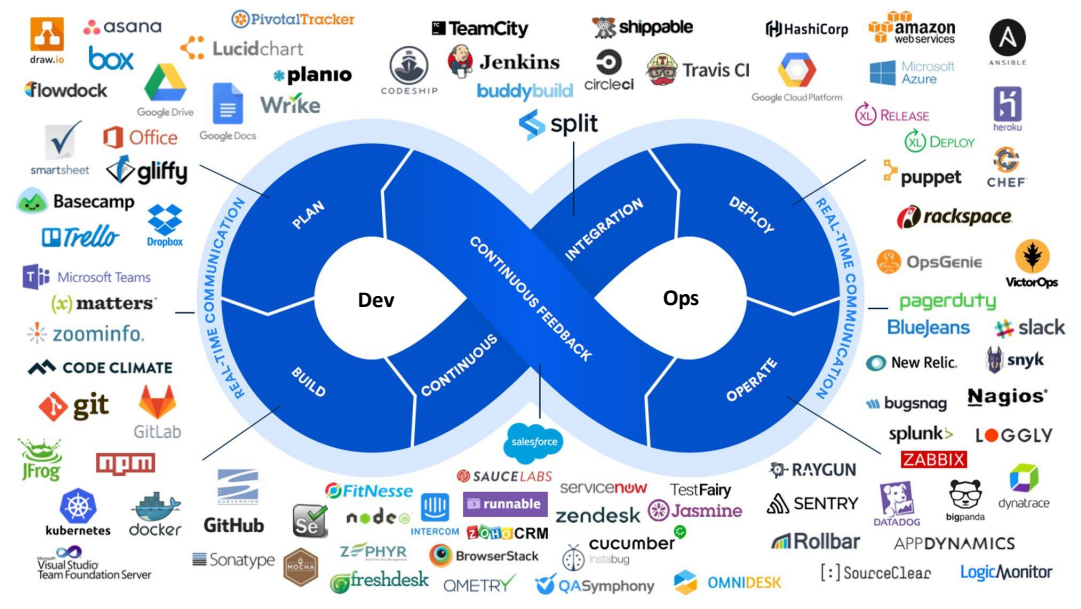


- Sysadmin
- Webmaster
- DBA

Fonte: https://medium.com/@ahamed.mush/devops-a-culture-33d5bdd5a483



Fonte: https://br.atlassian.com/devops


draw.io
asana
box
Flowdock
Google Drive
Lucidchart
PivotalTracker
planio
Google Docs
Wrike
CODESHIP
TeamCity
Jenkins
buddybuild
shippable
circleci
Travis CI
HashiCorp
Google Cloud Platform
amazon web services
Microsoft Azure
ANSIBLE
heroku
CHEF
smartsheet
Office
gliffy
Basecamp
Trello
Dropbox
split
RELEASE
DEPLOY
puppet
rackspace
Microsoft Teams
(x) matters
zoominfo.
CODE CLIMATE
git
GitLab
JFrog
npm
kubernetes
docker
Visual Studio Team Foundation Server
GitHub
Sonatype
REAL-TIME COMMUNICATION
PLAN
CONTINUOUS
BUILD
Dev
CONTINUOUS FEEDBACK
INTEGRATION
DEPLOY
OPERATE
Ops
REAL-TIME COMMUNICATION
salesforce
SAUCELABS
FitNesse
nodejs
INTERCOM
runnable
servicenow
TestFairy
zendesk
Jasmine
ZOHO CRM
cucumber
Instabug
ZEPHYR
BrowserStack
freshdesk
QMETRY
QASymphony
OMNIDESK
OpsGenie
VictorOps
pagerduty
BlueJeans
slack
New Relic.
snyk
bugsnag
Nagios
splunk>
LOGGLY
ZABBIX
RAYGUN
SENTRY
DATADOG
bigpanda
dynatrace
Rollbar
APPDYNAMICS
[:] SourceClear
LogicMonitor

Principais práticas em DevOps

- Integração contínua (CI)
- Distribuição/implantação contínua (CD)
- Virtualização (IaaS) e containers
- Automação de infraestrutura
- Gerenciamento de configuração
- Infraestrutura como código
- Monitoramento e observabilidade
- Feedback contínuo, comunicação e colaboração

Integração contínua (CI)

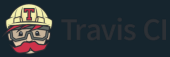
- Projetos de software normalmente são desenvolvidos de forma modular e por múltiplos desenvolvedores
  - Cada desenvolvedor implementa e testa a sua parte
  - Necessidade de testes de integração no final de cada ciclo de desenvolvimento (ex: sprints, milestones)
  - Problema: ciclos muito longos, erros demoram a aparecer

- Integração contínua é a prática de automatizar o build e teste de todos os componentes "continuamente"
  - Isto é, sem esperar um ciclo completo

Integração contínua (CI)

- Combinação das seguintes técnicas:
  - Versionamento de código (ex: Git)
  - Automatização de build/compilação (ex: Makefile, Ant, Maven, etc.)
  - Automação de testes (ex: JUnit, Cypress, Selenium, etc.)

- Exemplos de ferramentas
  - Jenkins, AWS CodeBuild, CircleCI, Travis CI, GitHub Actions, BitBucket Pipelines, etc.



Distribuição contínua (CD)

- Pode ser tratado como
  - Continuous delivery (Entrega contínua)
  - Continuous deployment (Implantação/publicação contínua)
- Extensão da integração contínua para diminuir o “lead time”
  - Tempo entre uma nova linha de código ser escrita e o software estar disponível para o usuário final (produção)
- Exemplos de ferramentas
  - AWS CodeDeploy, Octopus Deploy, TeamCity, GitHub Actions, etc.
  - Muitas ferramentas realizam todo o processo de CI/CD

Infraestrutura automatizada

- Problemas típicos no gerenciamento de infraestruturas:
  - Tarefas manuais repetitivas, grande chance de gerar erros, dificuldade de manter/rastrear histórico de modificações, etc.
  - Dificuldade de escalar!

- Conceitos:
  - Automatização de tarefas de gerenciamento
  - Gerenciamento de configuração
  - Infraestrutura como código

Automatização de tarefas de gerenciamento

- Como administrar/gerenciar servidores?
  - Conectar no servidor via SSH, executar comandos, instalar pacotes, modificar arquivos de configuração, adicionar usuários, modificar permissões, etc.
  - E se forem dezenas ou centenas de servidores?
  - E se forem sistemas operacionais diferentes? Ou mesmo distribuições Linux diferentes?

- Ferramentas de automação
  - Shell scripts, Ansible, Chef, CFEngine, Puppet, etc.

CFEngine

Shell do Sistema Operacional

- Saber utilizar o *shell* (terminal ou console) do sistema operacional é uma habilidade imprescindível para um Desenvolvedor Full Stack
- É um requisito para muitas das práticas de DevOps
  - Ex: CI/CD e automação de tarefas em geral

- Roteiro para conhecer e praticar os principais comandos do Linux:
  - Acessar: https://github.com/mvneves/sysadmin-linux/wiki

Ansible

ANSIBLE

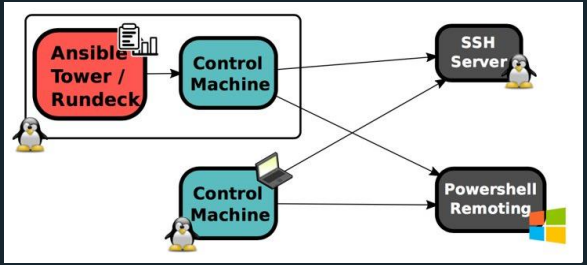
- Ansible é uma ferramenta de automação de TI de código aberto para gerenciamento de configuração e implantação de aplicações
- Por que usar o Ansible? Ele simplifica processos complexos e repetitivos, aumentando eficiência e confiabilidade
- Vantagens do Ansible: fácil de aprender, agentless (sem agente), arquitetura baseada em SSH, ampla comunidade e extensível com módulos e roles
- Exemplo de usuários:

Arquitetura do Ansible

- Agentless: não requer instalação de agentes nos hosts gerenciados, minimizando a sobrecarga e os requisitos de manutenção
- Baseado em SSH: utiliza protocolo SSH seguro para comunicação com hosts gerenciados
- Playbooks: arquivos de definição de automação escritos em YAML que descrevem as tarefas e ações necessárias
- Inventário de hosts: arquivo que lista todos os hosts gerenciados e suas respectivas informações (grupos, variáveis, etc.)
- Ansible Tower e Rundeck (opcional): são soluções para gerenciamento e orquestração de automação baseadas em interface web

Arquitetura do Ansible

- Não usa agentes, faz “push” via SSH (simplicidade!)
- Arquitetura do Ansible:



Playbooks do Ansible

ANSIBLE

- Playbooks são arquivos YAML que descrevem a automação desejada em termos de tarefas, handlers e variáveis
- Exemplo de playbook:

```yaml
- name: Atualizar e instalar pacotes
  hosts: all
  tasks:
    - name: Atualizar repositórios
      apt:
        update_cache: yes
    - name: Instalar pacote htop
      apt:
        name: htop
        state: present
```

Ansible

- Demo!
- Para reproduzir, acesse: https://github.com/mvneves/ansible-demo

Infraestrutura como código (IaC)

- *Infraestructure-as-code* (IaC)
- Permite aplicar as melhores práticas de desenvolvimento de aplicações na infraestrutura
  - Exemplo: versionamento, *code review*, testes automatizados, etc.

Principais benefício de IaC

- Consistência: Automatiza e padroniza o processo de implantação de infraestrutura
- Rastreabilidade: O código-fonte é facilmente rastreável, permitindo auditorias e identificação de alterações
- Reutilização: Permite a reutilização de componentes de infraestrutura, agilizando a implantação de novos ambientes
- Colaboração: Facilita a colaboração entre equipes de desenvolvimento, operações e suporte

Princípios fundamentais de IaC

- Versionamento: O código de infraestrutura é armazenado em sistemas de controle de versão, como Git, permitindo rastrear alterações e colaboração entre equipes
- Imutabilidade: As alterações na infraestrutura são feitas substituindo componentes, em vez de modificá-los, reduzindo o risco de falhas e a complexidade do gerenciamento
- Infraestrutura declarativa: A infraestrutura é descrita em arquivos de configuração (ex: YAML), declarando o estado desejado, e as ferramentas de IaC cuidam do processo de implantação
- Testes e validação: O código de infraestrutura deve ser testado e validado antes da implantação, garantindo a qualidade e a conformidade com as políticas de segurança e governança
- Automação: As etapas de provisionamento, configuração e gerenciamento da infraestrutura são automatizadas, reduzindo o tempo de implantação e a possibilidade de erros humanos

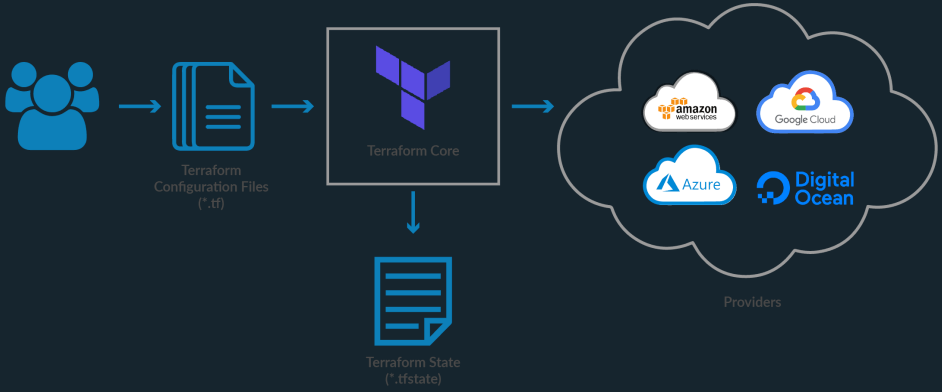
Ferramentas de IaC

- Específica de provedor de nuvem
  - AWS CloudFormation
  - Azure Resource Manager
  - Google Cloud Deployment Manager
- Independente de provedor de nuvem
  - Terraform
  - Pulumi

Terraform

- Terraform é uma ferramenta para construir, modificar e versionar infraestruturas de forma segura e eficiente
- Suporta múltiplas plataformas e provedores de serviços (AWS, Google Cloud, Azure, etc.)

Terraform


Terraform
Configuration Files
(*.tf)
Terraform Core
Terraform State
(*.tfstate)
amazon web services
Google Cloud
Azure
Digital Ocean
Providers

Terraform

- Demo!
- Para reproduzir, acesse: https://github.com/mvneves/terraform-demo

Observabilidade

- Do Inglês, *Observability* e as vezes abreviado como O11y
- Observabilidade é a capacidade de entender o estado interno de um sistema, analisando sinais externos produzidos pelo mesmo
  - Problema: aplicações executando na nuvem, de forma distribuída e em infraestruturas cada vez mais complexas e automatizadas
- Importante para identificar, diagnosticar e resolver problemas rapidamente
- Aumenta a eficiência, confiabilidade e segurança de sistemas

Pilares da observabilidade

- Métricas: Indicadores quantitativos que representam o estado e o desempenho do sistema. Exemplos incluem latência, vazão e taxa de erros.
- Logs: Registros detalhados de eventos e interações do sistema, auxiliando na identificação de problemas.
- Rastreamento: Permite acompanhar o caminho que uma requisição percorre pelos diferentes componentes de um sistema (distribuído).

Ferramentas de observabilidade

- AWS CloudWatch: monitoramento e alerta de métricas, logs e eventos para aplicativos e serviços na AWS
- ELK Stack (Elasticsearch, Logstash, Kibana): solução popular para gerenciamento de logs e métricas
- Grafana: ferramenta de visualização e análise de dados para métricas e logs
- Datadog: plataforma de monitoramento e análise de desempenho em tempo real, com suporte para logs, métricas e rastreamento.
- Sentry: plataforma de monitoramento de erros e exceções em tempo real que auxilia na identificação e rastreamento de problemas em campo

Observabilidade

- Demonstração do CloudWatch, DataDog e Sentry em produção

Implantação de observabilidade

- Desafios: Seleção das ferramentas adequadas, integração com sistemas e serviços na nuvem, escalabilidade e custo.
- Melhores práticas para implementar observabilidade em sistemas na nuvem:
  - Definir metas e objetivos claros (O que eu quero observar?).
  - Selecionar e padronizar ferramentas e métricas.
  - Incentivar a colaboração entre as equipes envolvidas.
  - Monitorar e ajustar continuamente a estratégia de observabilidade.

Feedback contínuo

- Feedback contínuo é o processo de coleta, análise e aplicação de informações para melhorar a qualidade e a eficiência do desenvolvimento e das operações de software
- Reduz riscos e erros, permitindo ajustes rápidos e melhorias no processo
- Facilita a colaboração e a comunicação entre as equipes de desenvolvimento e operações
- O feedback contínuo é uma parte essencial da abordagem de melhoria contínua em DevOps

Implantação de feedback contínuo

- Estabelecer uma cultura de abertura e transparência na comunicação
- Utilizar métricas e indicadores de desempenho para monitorar o progresso e identificar áreas de melhoria
- Integrar sistemas de monitoramento e alerta para identificar e resolver problemas rapidamente
- Realizar revisões de código e sessões de retrospectiva para discutir sucessos e áreas de melhoria.

Considerações finais

- Estamos vivendo a era das coisas definidas por software
  - Ex: software-defined everything (SDN, SDS, SDDC, etc.)
  - Importante aprender a "programar" a infraestrutura
- DevOps está eliminando a barreira entre infra e código (tanto de forma prática como cultural)
- Dicas de como continuar os estudos:
  - Criar uma conta em um provedor nuvem (ex: AWS, Google, Linode, DigitalOcean, etc.)
  - Reproduzir os roteiros apresentados nas aula
  - Experimentar ferramentas e serviços por conta própria

PUCRS online uol edtech