

DevOps Essentials.



Sumário

1	O que é DevOps	5
1.1	Manifesto Ágil	6
1.2	Agile Conference	6
1.3	Velocity Conference	6
1.4	DevOpsDays	7
1.5	Atualidade	7
1.6	C.A.M.S. (Culture, Automation, Measure, Sharing)	7
1.6.1	C de Cultura	7
1.6.2	A de Automação	8
1.6.3	M de Medição	8
1.6.4	S de Compartilhamento	8
1.7	Afinal, o que é Agile?	9
1.8	Spotify Squads	9
2	Preciso mesmo usar linux como workstation?	13
2.1	Motivo Técnico: Hardware	13
2.2	Segundo Motivo Técnico: Vivência.	14
2.3	Terceiro Motivo: Cultura	14
2.4	Windows	14
2.5	MacOS e Linux	14
3	Cloud & On Premises	17
3.1	Cloud	17
3.2	On premises	17
4	Inteirar-se sobre o Mercado DevOps	18
5	Explorar as principais Certificações	20
5.1	Técnicas	20
5.1.1	LPI DevOps Tools Engineer (LPIC-OT)	20
5.1.2	Docker Certified Associate (Docker DCA)	20

5.1.3	Kubernetes Certified Administrator (CKA)	21
5.1.4	Kubernetes Certified Application Developer (CKAD)	21
5.1.5	Certificações Cloud	21
5.2	Não Técnicas	21
5.2.1	Exin DevOps	21
5.2.2	Exin Agile Scrum	22
6	Entender uma Pipeline de Desenvolvimento Ágil	23
7	Explorar uma Pipeline DevOps	24
7.1	Planejar	24
7.2	Desenvolver ou Codificar	25
7.3	Construir	25
7.4	Testar	25
7.5	Lançar ou Deploy	25
7.6	Implantar e Operar	25
7.7	Monitorar	26
8	CI / CD	27
8.1	Continuous Integration	27
8.2	Continuous Deployment e Continuous Delivery	27
9	O que é Git Ops?	28
9.1	ChatOps	29
10	Quais os benefícios e porque eu preciso aprender Git Ops ?	30
11	Conhecer quais são as principais Ferramentas DevOps	32
11.0.1	Plan	32
11.0.2	Code	32
11.0.3	CI/CD	33
11.0.4	Test	33
11.0.5	Build / Release / Deploy	33
11.0.6	Monitor / Operate	33
11.1	Tabela Periódica DevOps	33
12	Conhecer as plataformas	34
12.1	Github	34
12.2	Code Anywhere	35
12.3	Heroku	35
13	Explorar alguns comandos Básicos do Git	36

13.1	Primeiros comandos git	38
13.2	Trabalhando com repositórios Remotos	41
14	Criar uma Pipeline DevOps	48
14.1	Começando nossa pipeline	49
14.2	Plan	52
14.3	Criando o APP	54
14.4	Coletando API Key Heroku	54
14.5	Configurar Secrets no github	55
14.6	Github Actions	57
14.7	Modificar a aplicação e verificar o deploy	61
14.8	git	64
14.9	Redes	64
14.10	Linux	64
14.11	Monitoramento	64
14.12	Aprenda uma Cloud	65
14.13	Aprenda Terraform	65
14.14	Aprenda Containers	65
14.15	Outros	65
14.16	Aprenda Sistemas Operacionais	65
14.17	Aprenda Ansible	65
14.18	Outros	66

1

O que é DevOps

Definir o que é DevOps é uma tarefa quase impossível, mas vou tentar explicar para vocês. . .

DevOps (Development + Operations) é um processo de desenvolvimento e entrega de software que enfatiza a comunicação entre os profissionais de desenvolvimento e operações.

O Termo DevOps tem tomado uma grande importância no mundo todo. Empresas como Amazon, Netflix, Facebook, Walmart, Spotify, entre tantas outras, estão investindo em funcionários, ferramentas e ambientes que sigam as práticas DevOps para melhorar a entrega e a qualidade de seus produtos.

Hoje, no entanto, o conceito de DevOps também está sendo utilizado para classificar funcionários, departamentos e times dentro do ambiente empresarial. Funcionários que possuem o perfil DevOps têm competência para trabalhar tanto na área de Desenvolvimento de Software como na área de Administração de Sistema.

Objetivos de um ambiente com práticas DevOps:

- Melhorar a frequência dos deploys;
- Automatizar processos;
- Diminuir a ocorrência de erros em novas versões;
- Curtos períodos de tempo para mudanças e melhorias;
- Recuperação rápida em caso de falhas no ambiente;
- Padronização nos processos de configuração e Servidores # História do DevOps

Para conhecermos sobre a história do DevOps, primeiro precisamos voltar no tempo e entender sobre Desenvolvimento e Manifesto Ágil.

Manifesto Ágil

Tudo começou na década de 60, com o surgimento de diversas metodologias para processos de desenvolvimento, como por exemplo o Kanban. Elas surgiram com o intuito de auxiliar o desenvolvedor a entregar seus projetos o mais rápido possível, facilitando a resposta às mudanças. Apesar da ampla utilização dessas metodologias, somente em 2001 um grupo de 17 desenvolvedores se reuniram em um resort para realmente discutir quais os pontos positivos em diversas metodologias que utilizavam.

A partir dessa discussão, surgiu o Manifesto para Desenvolvimento Ágil de Softwares, que ficou popularmente conhecido como Manifesto Ágil. O Manifesto Ágil afirma que melhores resultados no desenvolvimento de software podem ser obtidos através da valorização de:

- Indivíduos e interações, mais que processos e ferramentas;
- Software em funcionamento, mais que documentação abrangente;
- Responder a mudanças, mais que seguir um plano.

Agile Conference

Entre 2001 e 2008 diversas discussões foram propostas sobre como agilizar as entregas, sendo uma delas a palestra Infraestrutura Ágil de Andrew Schafer. Foi durante essa palestra que Andrew conheceu Patrick Debois e, juntos, criaram o grupo Agile System Administrator na plataforma Google Docs.

Assim, o termo DevOps surgiu em 2009, mas a ideia por trás já existia anos antes. A primeira vez que houve uma discussão sobre o assunto foi durante a Agile Conference de 2008, onde Andrew Schafer apresentou sua palestra de infraestrutura ágil para apenas uma pessoa: Patrick Debois.

Após a palestra, ambos discutiram diversos assuntos relacionados, sendo que posteriormente foram responsáveis por criar o grupo Agile System Administrator para difundir o assunto e convidar mais pessoas para colaborar.

Velocity Conference

Em 2009, na Velocity Conference da O'Reilly, John Allspaw e Paul Hammond apresentaram a famosa palestra chamada "10+ Deploys per Day: Dev and Ops Cooperation at Flickr". Durante esse evento, foram discutidos assuntos como:

- Interação entre os Desenvolvedores e a equipe de operações;
- Como conseguir um aumento dos deploys com ferramentas e mudanças culturais.

Patrick Debois comentou no twitter que lamentava não estar presente na palestra, recebendo do Flickr a seguinte resposta:

“Porque não organizar sua própria conferência de Velocity na Bélgica?”

DevOpsDays

Motivado pela palestra dos engenheiros da Flickr, Patrick Debois decidiu criar sua própria conferência na Bélgica, que ficou conhecida como DevOpsDays e foi um sucesso. Para lembrar o dia, foi criada a tag #DevOps no Twitter, que posteriormente seria usado como o nome da cultura.

Atualidade

Desde então, cada vez mais pessoas estão se tornando adeptas da cultura DevOps. Alguns dos acontecimentos mais recentes que favoreceram isso foram o surgimento de diversas tecnologias de apoio a essa ideologia, como, por exemplo, os livros Phoenix Project e Site Reliability Engineering do Google.

Phoenix Project é uma fábula que conta como um analista de TI salvou o departamento de uma companhia com a cultura DevOps # Compreender a importância do DevOps

Para entendermos a importância do DevOps, precisamos primeiro entender seus quatro pilares de sustentação.

C.A.M.S. (Culture, Automation, Measure, Sharing)

Respect the Culture, Automate if Possible, Measure Results and keep Sharing the Feedback

C de Cultura

Respeite a cultura.

Precisamos colaborar, compartilhar e entender a importância de manter uma relação saudável entre todas as áreas para que as equipes multidisciplinares possam trabalhar juntas e atingir os resultados.

A de Automação

Automatize se possível

Quando falamos de DevOps, queremos eliminar o máximo de trabalho laboral possível, trabalho laboral é aquele trabalho repetitivo. Se você, por exemplo, gasta todo dia 30 minutos para efetuar uma determinada ação, porque não gastar um tempo superior, como, por exemplo, 2 horas para automatizar esta rotina e poupar esses 30 minutos a partir de então?

M de Medição

Measure Results

Precisamos medir tudo que é possível: de processos a pessoas. Afinal, a única maneira de verificar se estamos no caminho certo ou melhorando é através da medição. O processo de melhoria contínua é o coração do DevOps!

S de Compartilhamento

Sharing the Feedback

S de compartilhamento? Soa estranho, não é? Mas é exatamente isto. O **S** é de **Shar-ing**, palavra inglesa que significa compartilhamento. Podemos dizer que esse é o pilar mais importante do DevOps.

Ambientes DevOps têm como uma das características fundamentais a cultura **Blameless** ou, em português, “Sem Culpa”, que é exatamente o que cria um ambiente propício ao compartilhamento. Não é sobre ninguém ser culpado por uma determinada ação que levou os sistemas a ficarem indisponíveis, é sobre todos se sentirem seguros e não terem medo de cometer erros. O erro faz parte do processo de aprendizado e deve ser compartilhado, assim como todas as melhorias que aplicamos em nosso ambiente de trabalho. # Correlacionar Metodologias Ágeis com DevOps

Quando falamos de DevOps, também precisamos entender sobre metodologias ágeis, uma vez que todo o processo DevOps é conhecido por ser um processo ágil.

Afinal, o que é Agile?

Agile é um processo de desenvolvimento de software que foca em pequenos entregáveis e um processo cíclico que sempre mobiliza melhorias. O produto final, nesse modelo, é entregue em pequenas frações visando o conceito de **melhoria contínua**. Nesse sentido, é diferente do modelo ainda utilizado em ambientes industriais, chamado **waterfall**, onde o projeto é entregue por completo e, portanto, não responde às diferentes necessidades do cliente durante o processo de desenvolvimento.

Com o Agile, trabalhamos com a metodologia **Scrum**, que é um fluxo de trabalho composto de **sprints** (corridas) com ciclos curtos, normalmente de 1 a 2 semanas. Nessas sprints, são realizadas reuniões diárias de 15 minutos chamadas de **Daily Standup**, normalmente feitas com a equipe de pé, por isso o nome Standup. O objetivo é falar o que foi feito no dia anterior e discutir os próximos passos, desbloqueando a equipe caso exista algum problema impedindo o desenvolvimento. No término da sprint, é realizada uma reunião de **Retrospectiva** ou **Review**, que serve para apontar o que foi aprendido e o que foi feito. Em seguida, ocorre a reunião de **Planning**, onde trabalhamos o **backlog**, uma lista de tarefas a serem executadas para selecionar quais serão as **tasks** que serão trabalhadas na próxima sprint.

Temos duas figuras importantes no **Scrum**, o **Scrum Master** e o **Product Owner**.

Scrum Master é responsável por guiar o time e dar suporte, se comunicar com outras equipes e destravar sempre que possível os membros do time.

Product Owner é o dono do produto, é ele que sabe qual é a visão final do entregável e os requisitos, por isso dá orientação sobre possíveis dúvidas.

Spotify Squads

Várias empresas têm seu próprio modelo de desenvolvimento ágil, normalmente baseado no agile/scrum, mas um dos modelos que se tornou bem famoso é o modelo de **spotify squads** que leva o nome da empresa que o criou.

Neste modelo de implementação ágil, as pessoas são focadas em ter autonomia para desenvolver e escalar os serviços. No modelo spotify, trabalhamos com **Squads** ou esquadrões e **Tribes** ou tribos. Temos também outras figuras que falaremos adiante.

Os **Squads** são similares ao scrum, times funcionais multidisciplinares e autônomos, normalmente de 6 a 12 pessoas, que focam em uma feature específico. Cada squad tem uma missão, um **Scrum Master** e um **Product Owner**.

Quando múltiplos **Squads** coordenam entre si para desenvolver uma mesma feature, aparece

uma nova persona no desenvolvimento, a **Tribe** que normalmente consiste em grupos de 40 a 1500 pessoas para manter alinhamento. Cada tribo tem seu **Tribe Lead**, que é responsável por coordenar os esquadrões e encorajar a colaboração.

Mesmo os **squads** tendo autonomia para desenvolver, normalmente precisamos de uma força extra de algum especialista em um determinado assunto, é aí que entra o **Chapter**. **Chapters** são o agrupamento de especialistas de uma determinada tecnologia, como por exemplo banco de dados, e com eles que podemos buscar apoio.

Membros de squads que são apaixonados em um determinado tópico podem criar uma **Guild**, que nada mais é que uma comunidade de interesse. Um detalhe interessante é que não existem líderes em guilds e qualquer um pode se juntar a uma **Guild**.

Cada tribo tem um **TPD TRIO** que é uma combinação de **Team Lead**, **Product lead** e **design lead** para garantir que existe um alinhamento contínuo entre essas três perspectivas.

Quando o processo é muito grande, múltiplas tribos precisam trabalhar entre si para conquistar um objetivo, **Alliances** são formadas com a combinação dos **TPD Trios** que trabalham em conjunto para manter as tribos alinhadas em um objetivo principal que é maior que o objetivo de cada tribo.

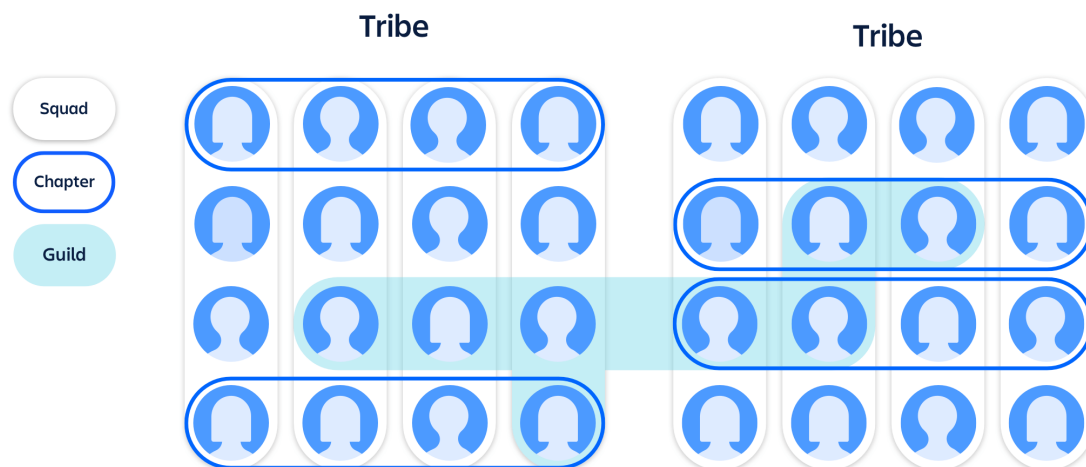


Fig. 1.1: spotify model

Por ser um modelo de muito sucesso, diversas empresas dizem que o modelo spotify é um **framework** de desenvolvimento, porém a própria spotify diz que não é bem assim, já que

apenas representa a visão da empresa em escala com perspectiva técnica e cultural e é um exemplo de como organizar múltiplos times no desenvolvimento e implementar melhorias.

Mas porque estamos falando do **Modelo Spotify**? Vocês vão ouvir falar muito sobre isso ao longo da sua jornada DevOps... diversas empresas trabalham com modificações do modelo Spotify para adaptá-lo a sua necessidade. Vamos entender melhor! # O que preciso aprender para me tornar um profissional DevOps?

Para nos tornarmos profissionais DevOps, precisamos aprender sobre diversos temas.

- Cultura;
- Programação;
- Gerenciar Servidores, Virtualização, Redes e Segurança;
- Criação de Scripts;
- Instalar e configurar middlewares;
- Instalar softwares;
- Versionar arquivos com GIT;
- Automatizar tarefas e gerenciar configurações;
- Infraestrutura como Código;
- Observabilidade;
- Orquestração de Containers;
- Compartilhar etc.

Aprenda sobre a cultura – DevOps é um movimento e uma cultura muito antes de ser um “cargo”, por isso os aspectos culturais são muito importantes.

Aprenda uma linguagem de programação – É altamente recomendado que você saiba programar, afinal, o futuro já está aqui e tudo está virando programação - até mesmo infraestrutura e redes. Python, Go, Nodejs... existem várias opções, não necessariamente é preciso aprender a mesma linguagem que sua empresa utiliza. Eu particularmente recomendo Python pelo poder e simplicidade.

Aprenda a gerenciar Servidores, Virtualização, Redes e Segurança – A principal tarefa de um profissional DevOps é gerenciar servidores. Assim, entender seu funcionamento é muito importante, até a parte mais baixa da tecnologia (arquitetura, memória, processador) são conhecimentos necessários, incluindo sistemas operacionais e especialmente Linux. Se você tem dúvida onde começar, escolha uma distribuição como o Ubuntu. Além disso, entenda como os protocolos básicos de rede como HTTP, DNS e FTP funcionam. Para implementar a segurança você tem que conhecer os protocolos e entender como não deixar brechas para invasores.

Aprenda a criar scripts – Scripts podem ser considerados como uma programação, onde o

Shell Script é um dos recursos mais utilizados, juntamente aos scripts escritos em Python, que é uma linguagem onde é possível fazer muita coisa escrevendo pouco.

Aprenda a instalar e configurar middlewares – Middlewares são programas que fornecem serviços para aplicações de outros programas, como, por exemplo, o Apache, Nginx e JBoss/Wildfly. Estes são utilizados em toda a internet para prover desde pequenos blogs WordPress até grandes marketplaces.

Aprenda como implantar software – Após aprender a configurar os middlewares, você vai precisar saber implantar as aplicações em servidores. Por exemplo, pode ter em um servidor com mais de uma aplicação rodando e utilizar o Nginx como proxy reverso para ambas.

Aprenda controle de versionamento com GIT – GIT é o sistema de versionamento mais utilizado na indústria de TI. Você não precisa ser um Expert, mas é interessante saber o básico para entender a tecnologia.

Aprenda a Automatizar suas tarefas e Gerenciar configurações – Automatizando tarefas básicas e usando gerenciamento de configurações com softwares como Ansible, Chef, Puppet fará você ganhar tempo para dedicar a tarefas que trarão crescimento pessoal e profissional. No futuro, você terá tantos ambientes para gerenciar que será impossível fazer de forma manual.

Aprenda Infraestrutura como Código – IaC é um dos pontos mais fortes para um ambiente DevOps. Afinal, ter um ambiente altamente confiável e escalável é uma das tarefas mais importantes para a TI.

Aprenda a monitorar sua Infraestrutura e seu Software – Imagine um atleta olímpico. Ele precisa de constante monitoramento de seus sinais vitais, alimentação, etc. . . A infraestrutura e o software não são diferentes! O monitoramento com ferramentas apropriadas, como Zabbix e Prometheus, permite prever e até evitar incidentes que possam causar um impacto negativo na organização.

Aprenda sobre Orquestração de Containers – Aprender Docker e Kubernetes é altamente recomendado, pois com estas ferramentas você consegue criar e “destruir” um ambiente inteiro em pouco tempo. Unindo isto a IaC, você consegue construir um ambiente altamente dinâmico.

Aprenda a Compartilhar – O ponto mais importante de uma cultura DevOps é que o processo de aprendizagem é contínuo. Compartilhar o conhecimento adquirido pode ajudar ainda mais e ser um dos pontos chaves em seu processo de mudança.

2

Preciso mesmo usar linux como workstation?

Resposta curta? Sim.

Resposta longa? Depende de muitas variáveis.

Vamos começar pensando no porque de aprendermos linux. . .

Aprendemos linux porque a maior parte dos servidores na internet rodam sistemas linux e para isso temos diversos motivos, mas vamos falar sobre três deles.

Motivo Técnico: Hardware

Um servidor linux consome menos recursos do que servidores Windows, por exemplo.

Se você abrir a página da Microsoft para ver quanto de Hardware é necessário para a instalação de um Windows Server 2016, encontrará os seguintes requisitos mínimos: Processador de 1.4Ghz e 64 bits 2 GB de memória RAM 45 GB de espaço em disco.

Lógico que, hoje, existe a versão “core edition” que não possui a interface gráfica – e consome incríveis 512MB RAM – porém, ainda consome cerca de 32GB de disco rígido. Além disso, tente “controlar” um Windows Server 2016 com 2GB de RAM e você terá vontade de colocar fogo em seu servidor e jogá-lo do vigésimo andar. Assim, é necessário ao menos 4GB para ter

uma experiência saudável à mente humana.

Quando falamos de Linux, podemos encontrar o seguinte cenário - aproveito para trazer duas distribuições de “famílias” distintas:

Ubuntu Server 18.04 (requerimento mínimo): Processador 300Mhz x86 384 MB de memória RAM 2,5 GB de espaço em disco

CentOS 7 / RHEL 7 (requerimento recomendável): Processador 1Ghz 64 Bits 1 GB de memória RAM 5 GB de espaço em disco

Segundo Motivo Técnico: Vivência.

Quando você utiliza sistemas linux em sua workstation, está não só se acostumando com a usabilidade do sistema como entendendo os servidores, que rodam o mesmo linux que você está utilizando em seu computador, só que sem a interface gráfica. Você vai entender também que nem tudo na vida se resume a tão famigerada “Interface Gráfica”, que é importante saber utilizar o terminal e não é esse monstro que parece ao ser visto de longe.

Terceiro Motivo: Cultura

O Linux nasceu como um sistema “livre” feito por diversas pessoas e com a premissa do compartilhamento, que “por acaso” é um dos pilares mais importantes do DevOps.

Windows

Hoje, temos alternativas como o WSL, o Windows Subsystem for Linux, que funciona muito bem e ajuda a ter esse contato com o terminal e a linha de comando dos servidores linux. Não é a mesma coisa, mas já é um ótimo primeiro passo.

MacOS e Linux

Tanto o Linux quanto o MacOS tem seu kernel baseado em UNIX. Inclusive, algumas pessoas dizem que o MacOS é um linux, outros dizem que ambos são compatíveis por uma série de similaridades. No final, ambos fornecem uma facilidade para trabalhar no terminal. > Deixando claro minha preferência por aqui, eu prefiro trabalhar com Linux <3# Por onde devo começar?

Uff, mas é muita coisa! Vou desistir agora mesmo, parece impossível... você pode estar pensando.

Na verdade, não é! O que sempre acontece é que um profissional não é especialista em todas as áreas. Um bom profissional DevOps consegue navegar dentre todas estas áreas, com especialidade em algumas delas. No meu caso, a especialidade é Containers, em outras apenas domino o mínimo, como linguagem de programação. Eu entendo que não preciso ser fluente em Python, por exemplo, mas é preciso saber LER um código e entender o que está sendo feito. É obvio também que quantas mais áreas um profissional DevOps dominar, melhor será.

Mas e então, por onde começar? Não existe um caminho certo ou errado, existe o caminho que funciona para cada um. No meu caso, como já tinha uma boa base de redes e datacenter, comecei a estudar devops com servidores linux, um pouquinho de shell, infraestrutura como código com ansible e me especializei em container. Assim, fui seguindo por todos os outros tópicos à medida que era exposto aos temas em meu ambiente de trabalho.

Uma boa dica é trabalhar com OKRs!

OKR significa Objectives, Key Results.

Pode parecer bem complicado, mas pensando de forma geral não é tanto assim. Primeiro definimos um tema. Vamos usar como exemplo os Containers.

Agora que escolhemos o tema, precisamos definir o objetivo, que significa uma direção clara do que você ou a sua empresa pretende conquistar. O objetivo precisa ser preciso, então nada de “Subir um container com docker” ou “orquestrar containers com Kubernetes”.

Vamos definir nosso objetivo-exemplo. . .

Objetivo: “Efetuar o deploy da aplicação CRM da empresa utilizando containers”

Depois disso definimos os “Key Results” ou resultados chaves. Os resultados chaves servem de parâmetro para determinarmos o quanto estamos perto de alcançar os objetivos.

Vamos definir alguns “KRs” para alcançar nosso objetivo-exemplo:

KR #1: Entender o funcionamento de Containers, Redes e Volumes; KR #2: Difundir o conhecimento entre as equipes de desenvolvimento e operação; KR #3: Implantar servidores Docker em Desenvolvimento e Homologação; KR #4: Efetuar o deploy da aplicação em Desenvolvimento e Homologação; KR #5: Efetuar o deploy em produção de maneira automática e escalável.

Já que temos 5 Key Results, podemos dizer que cada um tem o peso de 20%. Assim, é possível medir o quanto estamos próximos de concluir nosso objetivo.

Esse tipo de metodologia trabalha não só o lado técnico - pois OKRs são utilizados em ambi-

entes DevOps-, como também o lado psicológico. Claramente, quando você consegue ver sua evolução, por menor que ela seja, você se sente bem e funciona como uma motivação/prêmio pelo esforço.

3

Cloud & On Premises

Quando falamos em DevOps, sempre surgem dois termos: “Cloud” e “On Premises”. Mas, afinal, o que significam?

Cloud

Cloud, ou a tão famigerada “Nuvem”, é o termo utilizado quando tratamos de uma rede global de servidores. A palavra Nuvem é utilizada por ser uma tecnologia exatamente como as nuvens que temos no céu: um ecossistema imenso, disperso por todo o globo terrestre. Assim, diversos servidores trabalham juntos, sendo responsáveis por executar aplicativos, distribuir conteúdo, transmitir e-mails ou vídeos e até mesmo aquelas redes sociais que você acessa no dia a dia.

On premises

On premises significa sobre premissa, ou seja, no local. Usamos esse termo para nos referir a uma infraestrutura que está sendo executada localmente, em uma sala de servidores ou até mesmo um Datacenter.

Então, isso quer dizer que só temos a “Cloud” e o “On Premises”, certo? Bem, mais ou menos. Existem desdobramentos disso, como as clouds públicas, privadas e até mesmo híbridas, que não necessariamente precisam ter seu acesso aberto para a internet... mas não se preocupe com relação a isso no momento. Afinal, em todos os tipos de infraestrutura há espaço pra DevOps, até mesmo quando falamos de Windows.

4

Inteirar-se sobre o Mercado DevOps

O mercado de trabalho DevOps anda bem aquecido.

Vamos a alguns números:

DevOps em: Mundial
252.409 resultados

DevOps em: Brasil
4.774 resultados

DevOps em: Países Baixos
8.563 resultados

DevOps em: Canadá
8.903 resultados

DevOps em: Estados Unidos
86.233 resultados

DevOps em: Irlanda
2.014 resultados

DevOps em: Alemanha
15.468 resultados

Como podem ver, os números falam por si só.

Mas quais são esses profissionais “DevOps” que são tão requisitados?

Bem, temos diversas áreas correlacionadas a DevOps.

DevOps Engineer Site Reliability Engineering Cloud Platform Automation Development Lead

No geral, o profissional DevOps é aquele profissional que conhece e pratica a Cultura DevOps, que Automatiza sempre que possível, que Mede os Resultados e que Compartilha o Feedback (CAMS - Culture, Automation, Measurement, Sharing). Lembra?

5

Explorar as principais Certificações

Quando falamos de DevOps, não podemos deixar de citar as principais certificações para o mercado.

Podemos dividi-las em certificações Técnicas e não Técnicas.

Técnicas

LPI DevOps Tools Engineer (LPIC-OT)

Testa o candidato com relação a todos os pilares de DevOps, bem como conhecimento em:

- Engenharia de Software;
- Gerenciamento de Containers;
- Deploy de Máquinas;
- Gerenciamento de Configurações;
- Operações de Serviços.

Docker Certified Associate (Docker DCA)

Testa o conhecimento do candidato em tecnologias de Containers, focado em Docker e Kubernetes, nas áreas de:

- Orquestração;
- Criação, Gerência e Registro de Imagens;
- Instalação e Configuração;
- Redes;
- Segurança;
- Armazenamento e Volumes.

Kubernetes Certified Administrator (CKA)

Testa o conhecimento e prática em Kubernetes, nas áreas de:

- Arquitetura do Cluster, Instalação e Configuração;
- Cargas de Trabalho e Agendamento;
- Serviços e Redes;
- Armazenamento;
- Resolução de Problemas.

Kubernetes Certified Application Developer (CKAD)

Testa o conhecimento e prática em Kubernetes nas áreas de:

- Conceitos Base;
- Configuração;
- Pods Multi-containers;
- Observabilidade;
- Serviços e Redes;
- Persistência de Estado.

Certificações Cloud

Certificações de Cloud como GCP, AWS, Azure são muito valiosas. Normalmente testam o conhecimento do candidato nos níveis de:

- Associate/Pratitioner -> Conhecimento Básico
- Engineer -> Conhecimento Intermediário
- Specialist -> Conhecimento Pleno

Não Técnicas

Exin DevOps

Existem 3 certificações da Exin focadas em DevOps:

- Foundation -> Testa o entendimento do profissional com relação a conhecimentos Básicos de DevOps;
- Professional -> Testa o conhecimento profissional das práticas e entendimentos essenciais da cultura DevOps;
- Master -> Nivel avançado que testa candidatos com relação ao desenvolvimento e manutenção de aplicações e serviços (para esta certificação é obrigatório participar de um treinamento por um partner).

Exin Agile Scrum

Existem diversas certificações na área de Scrum & Agile voltadas para Scrum Masters, Product Owners, Coachs e Fundamentos. # O que é uma pipeline?

O nome Pipeline quer dizer tubulação, um recurso utilizado para transportar um produto - seja óleo, gás, água etc. - para um destino.

Este termo é amplamente utilizado quando falamos de DevOps.

O Objetivo de uma pipeline é automatizar todo o processo de entrega de infraestrutura e/ou software de forma rápida, garantindo qualidade, testes, estabilidade e escalabilidade.

6

Entender uma Pipeline de Desenvolvimento Ágil

Quando falamos de Desenvolvimento Ágil de software, precisamos entender que está diretamente ligado a uma “pipeline” de software.

Uma pipeline de entrega de software normalmente se constitui das seguintes etapas:

- Planejamento;
- Análise;
- Desenho;
- Implementação;
- Testes e Integração;
- Deploy e Revisão.

É um processo em ciclos, onde definimos um entregável e trabalhamos até sua conclusão. Depois disso, começamos um novo ciclo, definindo novamente um entregável e trabalhando até sua conclusão.

7

Explorar uma Pipeline DevOps

Uma pipeline DevOps tem como base uma pipeline agile, mas ligeiramente diferente.

Também trabalhamos com pequenos entregáveis, porém o processo é um ciclo sem fim. Temos dois círculos: trabalhamos em sua metade com a parte de Dev, e a outra metade com a parte de Ops.

Dev: - Planejar; - Desenvolver ou Codificar; - Construir; - Testar.

Ops - Lançar; - Implantar e Operar; - Monitorar.

A diferença básica é que sempre estamos voltando ao início do ciclo e “melhorando” o processo.

Esse processo é chamado de melhoria contínua.

Normalmente, quando falamos de uma pipeline DevOps, falamos de ferramentas utilizadas em cada etapa e não necessariamente temos uma definição exata de qual ferramenta usar em cada uma delas. Hoje, no entanto, temos algumas ferramentas que são as “preferidas” ou “mais utilizadas”.

Planejar

Para a etapa de planejamento, as ferramentas mais utilizadas são boards Kanban. Podemos citar o Jira, Trello, Wekan, Notion dentre várias outras.

Neste processo, listamos os requerimentos e criamos pequenos “cards” ou “cartões” no qual iremos definir quais serão as atividades de uma sprint.

Desenvolver ou Codificar

Durante a etapa de Desenvolvimento, ou coding, diferente do que as pessoas normalmente pensam com relação a uma linguagem de programação, estamos dizendo da organização do código e da colaboração entre as equipes. A ferramenta mais utilizada nesta etapa é, sem sombra de dúvidas, o git. Juntamente a ele utilizamos algum servidor git, normalmente github, gitlab, bitbucket, gitea, gogs, ou alguma outra solução.

Construir

A etapa de construção, ou build, está diretamente ligada à geração do artefato entregável, seja ele um pacote java, python ou até mesmo um executável. Nesta etapa, normalmente é utilizado o Gradle, Maven, Packer.

Testar

Nesta etapa, efetuamos diversos testes para garantir que o entregável é de fato o que foi planejado. Quando falamos de desenvolvimento, normalmente o profissional qualificado para desenvolver os testes é o QA Engineer ou Quality Assurance Engineer. É o profissional que garante a “qualidade” da entrega. Nesta etapa, utilizamos normalmente as ferramentas Selenium, JUnit, pytest, inspec.

Lançar ou Deploy

Essa é uma das etapas com mais ferramentas a serem utilizadas, dentre elas temos o Terraform, Ansible, Chef, Puppet, Powershell.

Implantar e Operar

A etapa de implantação é que mobiliza mais ferramentas a serem utilizadas, é nela que fazemos com que nosso artefato testado entre em execução. Nesta etapa, podemos utilizar ferramentas como docker, rkt, kubernetes, nomad, openshift e também as mais diversas clouds como Microsoft Azure, Amazon AWS e Google Cloud Platform entre outras.

Monitorar

Na etapa de monitoramento, garantimos que o sistema está saudável. É uma das etapas mais importantes de uma pipeline DevOps, onde monitoramos endereços IP's, registros DNS, arquivos de logs e status de API's. São utilizadas ferramentas como Prometheus, Grafana, Elasticstack, Datadog, Splunk, Nagios.

8

CI / CD

Para fazermos a integração de todos os passos de uma pipeline DevOps, precisamos entender um conceito muito interessante que é chamado de **CI/CD**. **CI**, ou Continuous Integration, se traduz no conceito de Integração Contínua e **CD**, Continuous Deployment ou Continuous Delivery, é a Entrega ou Implantação Contínua.

Continuous Integration

Podemos dizer que o Continuous Integration é o que liga todas as partes do processo, o gatilho que faz com que um processo ou ferramenta A, se conecte a um processo ou ferramenta B, que trabalha garantindo uma integração contínua entre cada elo do processo.

Continuous Deployment e Continuous Delivery

Continuous Deployment e Continuous Delivery é o processo de entrega do produto final, seja um website ou algum outro produto. A diferença básica é que, no **Deployment**, o produto final é entregue de maneira totalmente automatizada, sem a necessidade de intervenção. Enquanto isso, no **Delivery**, o produto é entregue a um usuário final que decide se o produto vai “ao ar” ou não e quando. Ambas as estratégias são amplamente utilizadas, portanto, não há certo ou errado. Sobre a escolha, vai depender apenas do modelo de negócio da empresa.

9

O que é Git Ops?

Muitas pessoas tentam dizer que devops é uma coisa e gitops é outra, mas não é bem dessa maneira.

Em resumo, podemos dizer que o conceito de gitops é a implementação da automação no git server (Github, Gitlab, Bitbucket, etc..) onde todo o desenvolvimento ágil é “puxado” através do Git.

Para entender esse tipo de abordagem, precisamos entender alguns conceitos:

Repositório Local - Repositório Local é o diretório git localizado em sua máquina onde é feito toda a codificação do produto;

Repositório Remoto - Repositório Remoto é o repositório Git gerenciado através de uma ferramenta git server;

Branches - Branches são linhas que divergem do tronco principal de desenvolvimento, normalmente utilizamos branches para implementar novas features (recursos) ou fixes (correções) sem que o código que está rodando em produção seja modificado no meio deste processo;

Pull Request - Pull request, ou merge request, é o processo em que o código de uma branch não principal é solicitado a ser migrado com o código principal, tornando assim possível a atualização do produto;

Review - O processo de review consiste na equipe devops avaliar o código enviado por meio

de um pull request a fim de encontrar possíveis problemas e discutir melhorias ou correções antes das modificações entrarem no código principal;

Merge - Processo em que é feito a fusão do código requisitado pelo pull request com o código da branch principal;

Webhook - Webhook é uma forma de recebimento de informações quando um evento acontece. É utilizado como gatilho para disparo de funcionalidades, tarefas ou ações após um sistema se comunicar com um segundo sistema.

Para trabalhar com gitops, desenvolvemos o código em nosso repositório local, através de uma branch não principal, enviamos o código para nosso repositório remoto e criamos um pull request para ser feito o review. Após este review, efetuamos o merge do código. No fim desses passos, é ativado um webhook para dar prosseguimento ao processo de entrega.

ChatOps

Muitas das vezes, trabalhamos com **ChatOps**, que também mobiliza os conceitos de **GitOps**

ChatOps é a junção das tarefas de Automação e Colaboração. Trata-se de delegar responsabilidade de tarefas e ações para um robô interno que também faz parte da organização e está escutando as conversas. ChatOps não é somente tecnologia, é também uma metodologia e uma mudança cultural, assim como DevOps.

ChatOps = Chat + DevOps

O funcionamento do ChatOps é simples: utilizando a aplicação de Chat (Slack, Rocketchat, Microsoft Teams), adicionamos bots para executar ações em nossa infraestrutura através de comandos enviados pelas equipes DevOps.

10

Quais os benefícios e porque eu preciso aprender Git Ops ?

GitOps traz bastantes benefícios para uma pipeline devops. O principal é a cultura de compartilhamento e trabalho distribuído, o qual permite que diversas pessoas consigam trabalhar de forma simultânea a fim de atingir a entrega de maneira mais rápida, ou seja, mais ágil.

Outros benefícios do GitOps são:

Aumento de Produtividade - A automação da implantação contínua acelera o tempo médio de implantação. Assim, a equipe pode enviar mais alterações por dia, aumentando a produção de desenvolvimento exponencialmente.

Experiência de desenvolvedor aprimorada - Os desenvolvedores podem usar ferramentas familiares como o Git para gerenciar atualizações e recursos para ferramentas de operação de maneira mais rápida. Em consequência, os desenvolvedores que entrarem na equipe mais recentemente podem trabalhar e ser produtivos em poucos dias, uma vez que o código é compartilhado.

Maior Confiabilidade - Através do git, podemos reverter ou efetuar rollbacks, diminuindo o tempo médio de recuperação (MTTR - meantime to recover).

Estes são apenas alguns dos muitos benefícios elencáveis! Como conclusão, podemos dizer que quanto mais se trabalha com GitOps, mais benefícios da utilização são aproveitados! #

Entender a relação do DevOps com Open Source

DevOps está diretamente ligado ao Open Source por um de seus pilares mais importantes, o Compartilhamento.

Uma das chaves do desenvolvimento de software Open-source é a colaboração e a transparência.

DevOps é sobre se importar com toda a linha de entrega, seja você vindo de Desenvolvimento(Dev) ou Operação(Ops). No DevOps, nos preocupamos com o entregável, por isso, a questão do “não é problema meu” não deve existir. O objetivo de todos é entregar um produto, e se esse produto não está sendo entregue é algo que afeta toda a empresa.

As equipes DevOps devem estar inteiramente comprometidas com este compartilhamento. Por esse motivo, geralmente optam por ferramentas OpenSource, uma vez que seu código é aberto e sua base é o compartilhamento em prol da comunidade.

O maior benefício de se utilizar OpenSource é o simples termo: **Vendor lock-in** ou **Aprisionamento tecnológico**.

O aprisionamento tecnológico acontece quando você fica preso a um determinado fabricante, o que te torna extremamente dependente e impede que troque de fornecedor.

11

Conhecer quais são as principais Ferramentas DevOps

Já que vamos trabalhar com DevOps, é bom conhecer as principais ferramentas atualmente utilizadas no mercado por área:

Lembrando que estamos citando apenas ferramentas OpenSource ou com planos Free:

Plan

Com as ferramentas de planejamento desenhamos diagramas e

- Notion
- Trello
- Github
- Gitlab
- Diagrams

Code

- Git
- VSCode

CI/CD

- Jenkins
- Github
- Gitlab

Test

- Inspec
- Molecule
- Selenium

Build / Release / Deploy

- Docker
- Kubernetes
- Ansible
- Terraform

Monitor / Operate

- Prometheus
- Grafana
- Zabbix
- Graylog
- Fluentd
- Slack
- RocketChat

Tabela Periódica DevOps

Para aqueles que gostam de se aventurar, a **Digital.ai** desenvolveu uma tabela periódica de DevOps com diversas ferramentas úteis! Confira em <https://digital.ai/periodic-table-of-devops-tools>.

12

Conhecer as plataformas

Agora que já entendemos os conceitos básicos do DevOps, vamos implementar uma pipeline completa! Não se preocupe com os recursos do seu computador, utilizaremos apenas plataformas web gratuitas para desenvolver nosso projeto.

Github

Começaremos pelo Github, plataforma onde vamos guardar nosso código e executar todo o processo de CI/CD de forma automática.

Para isto, acesse o website <https://github.com> e crie sua conta clicando em **Sign up**

1. Digite seu e-mail
2. Digite uma senha
3. Digite um nome de usuário
4. Escolha se deseja receber atualizações de produtos por email (y para sim ou n para não)
5. Resolva o puzzle para provar que você é humano.
6. Clique em Create Account
7. Verifique seu e-mail e digite o código de confirmação

Code Anywhere

O Code Anywhere será nosso Cloud IDE, onde iremos escrever nosso código para enviar para o github.

Acesse o website <https://codeanywhere.com/> e crie sua conta clicando em **Sign up**

1. Preencha seu “Nome” e “Sobrenome”
2. Preencha seu “Email”
3. Preencha sua “Senha”
4. Clique em “Sign up for free”

Heroku

Heroku é um PaaS (Plataforma como Serviço) para publicar aplicações na cloud.

Acesse o website <https://www.heroku.com/> e crie sua conta clicando em **Sign up**

1. Preencha seu “Nome” e “Sobrenome”
2. Preencha seu “Email”
3. Preencha o nome da sua Empresa
4. Preencha seu Cargo
5. Preencha seu País
6. Selecione sua linguagem primária de desenvolvimento
7. Clique em “Create Free Account”
8. Verifique seu e-mail e clique no link de confirmação
9. Digite uma senha
10. Clique em “Set Password and Log in”

Agora que temos todas as contas criadas, podemos seguir para as plataformas e entender como trabalhar com cada uma delas. Vamos lá?

13

Explorar alguns comandos Básicos do Git

Para trabalhar com o Git iremos utilizar nossa conta do github juntamente com nosso IDE o codeanywhere.

Primeiramente vamos criar um repositório de teste para nossos estudos

Acesse a página do github e clique em “Create Repository”

1. Coloque o nome do repositório de estudo-git
2. Crie uma boa descrição para o repositório como por exemplo “Repositório para estudo dos comandos git”
3. Configure o mesmo como privado ou público. > Um repositório privado só é acessível pelo seu usuário e os colaboradores do mesmo, o público tem seu código acessível por toda internet.
4. Clique em Create Repository

Ao criar esse repositório já vemos alguns comandos git que podemos utilizar, por hora vamos seguir para o codeanywhere para inicializar nosso ambiente de desenvolvimento, iremos voltar no github em alguns minutos.

Acesse a página do CodeAnywhere e clique em “Create Container”

Um container no code anywhere é um ambiente de desenvolvimento onde iremos definir todos os padrões do nosso IDE para desenvolver nosso código

Vamos colocar o nome do nosso container de “**Ambiente de Desenvolvimento**” e utilizaremos a stack “**Blank**”, clique em create

Aguarde alguns minutos para que o deploy do nosso ambiente seja executado e possamos trabalhar com o mesmo.

Uma vez que seu state apareça como Started podemos clicar em OPEN IDE

Um detalhe interessante do CodeAnywhere é que sua interface é muito parecida com o VSCode.

Temos 3 áreas que precisamos conhecer da nossa IDE

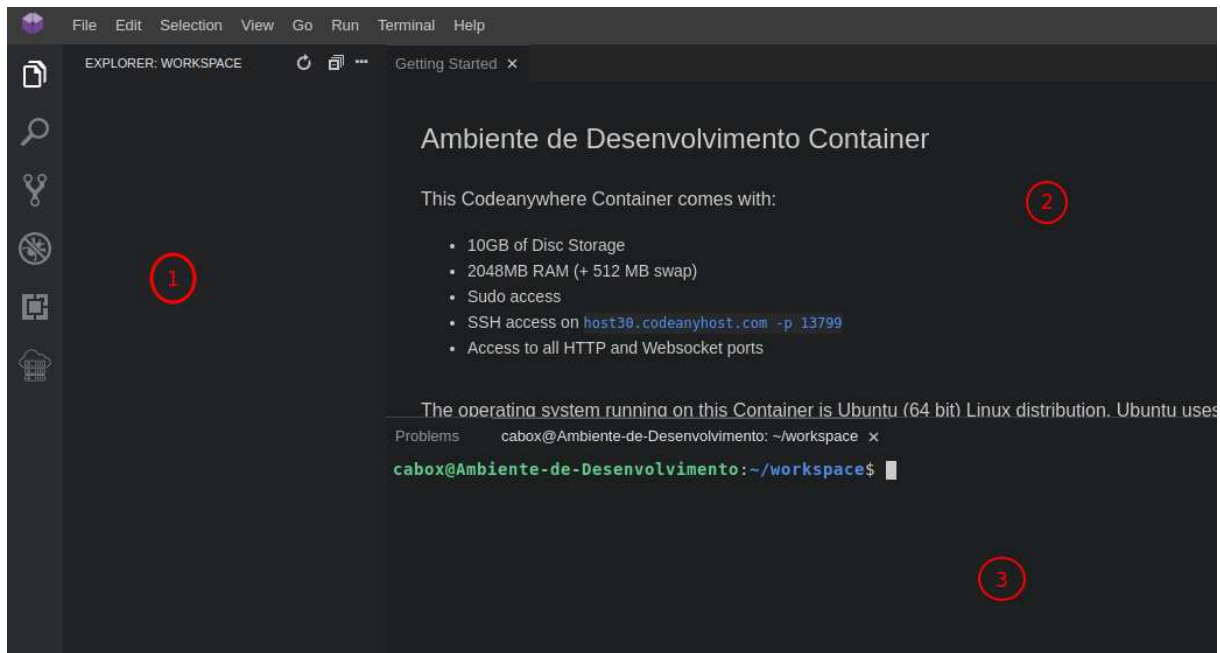


Fig. 13.1: CodeAnywhere IDE

1. **Workspace** -> Aqui é onde vemos nossa área de trabalho, os arquivos e os diretórios que iremos trabalhar.
2. **Editor** -> Através do editor podemos criar/modificar/editar os arquivos do nosso workspace
3. **Terminal** -> O clássico terminal linux, onde podemos executar os comandos shell.

Primeiros comandos git

Todos os comandos serão executados em nosso terminal.

Primeiramente vamos verificar se temos o binário do git instalado na máquina.

Faremos isto através do comando:

```
$ git --version
```

Agora que verificamos que temos nosso binário do git instalado, vamos criar um diretório para trabalharmos com o git.

```
$ mkdir estudo-git  
$ cd estudo-git
```

Veja que a pasta criada já é visível em nosso **workspace**

Para inicializar um novo repositório, iremos executar o comando

```
$ git init
```

Uma vez criado nosso repositório iremos configurar nosso nome e email padrão através do comando

```
$ git config --global user.name "Seu Nome"  
$ git config --global user.email "seuemail@seudominio.com.br"
```

Agora que já temos nossas informações já configuradas, vamos iniciar o versionamento do diretório

Crie um arquivo qualquer

```
$ touch arquivo1
```

Adicione o arquivo para a área de staging

```
$ git add arquivo1
```

O comando git add adiciona o arquivo em uma área chamada staging, é basicamente uma área de “espera” onde o arquivo aguarda ser enviado para o código principal

Visualize o estado atual do repositório

```
$ git status
```

```
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   arquivo1
```

O comando git status mostra o estado do repositório no momento atual

Grave a nova versão do repositório:

```
$ git commit -m "Meu primeiro commit"
```

O comando git commit salva o estado dos arquivos que estão na área de staging. O parâmetro -m atribui uma mensagem ao commit, servindo para identificar facilmente o que foi realizado na alteração.

Para modificar nosso commit, iremos criar outros arquivos

```
$ touch arquivo2 arquivo3
```

Adicione alguns arquivos a área de staging

```
$ git add arquivo2 arquivo3
```

Faça o commit atual do repositório

```
$ git commit -m "Atualizando o repositório"
```

Vamos modificar novamente nosso repositório desta vez criando 20 arquivos dentro do mesmo

```
$ touch app{1..20}.log
```

Agora temos 20 arquivos fora do nosso commit que precisam ser adicionados.

```
$ git status
```

Para adicionar todos os arquivos não é necessário adicionar um a um, podemos utilizar o parâmetro `--all` ao comando `git add`

```
$ git add --all  
$ git status
```

Efetue o commit dos arquivos

```
$ git commit -m "Adicionando arquivos de log"
```

Podemos verificar o histórico do nosso repositório através do comando

```
$ git log
```

Trabalhando com repositórios Remotos

Agora que já conhecemos os comandos básicos do git podemos trabalhar com repositórios remotos.

Um dos grandes benefícios de se utilizar o git é a possibilidade de termos nosso repositório local (na nossa máquina) e o repositório remoto (servidor)

Nosso repositório remoto será o github. Vamos acessar nosso github e abrir nosso repositório estudo-git criado anteriormente.

Ao abrir nosso repositório vemos um guia rápido com algumas informações importantes.

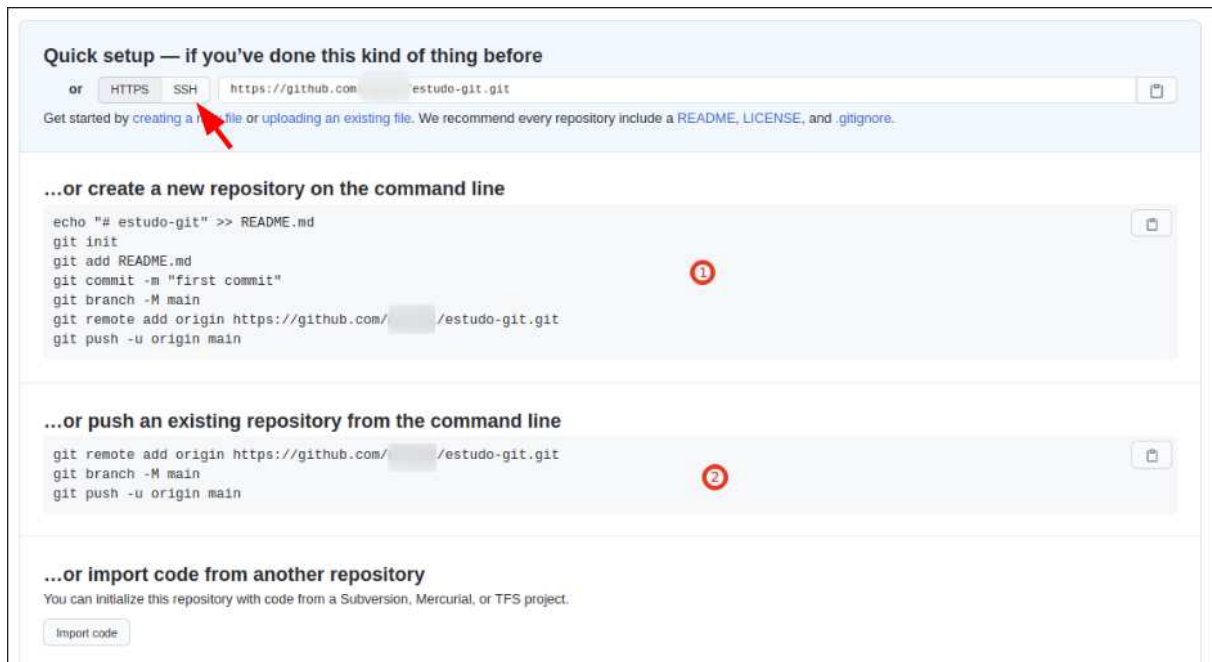


Fig. 13.2: github-repo

Primeiramente selecione ssh

Vamos focar nos itens 1 e 2

1. Para criar um repositório do Zero
2. Para enviar um repositório existente para o github.

Iremos trabalhar com a segunda opção, pois já temos nosso repositório inicializado.

Mas primeiramente precisamos gerar um par de chaves ssh para ter uma comunicação segura entre o github e o codeanywhere container.

No terminal do CodeAnywhere podemos criar estas chaves através do comando

```
ssh-keygen
```

Apenas aperte enter até a geração das chaves

Copie o conteúdo da chave publica

```
$ cat ~/.ssh/id_rsa.pub
```

Copie o conteúdo da chave e vamos para a página do Github

Clique em na foto do seu usuário e em seguida em Settings

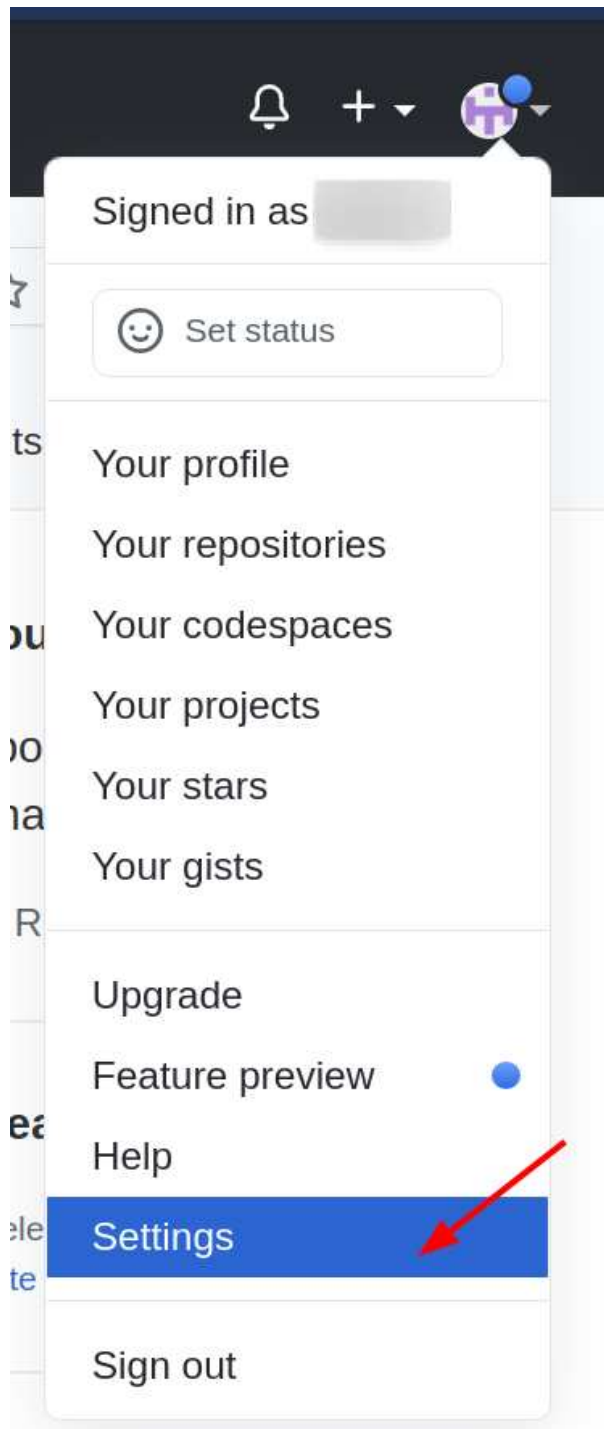
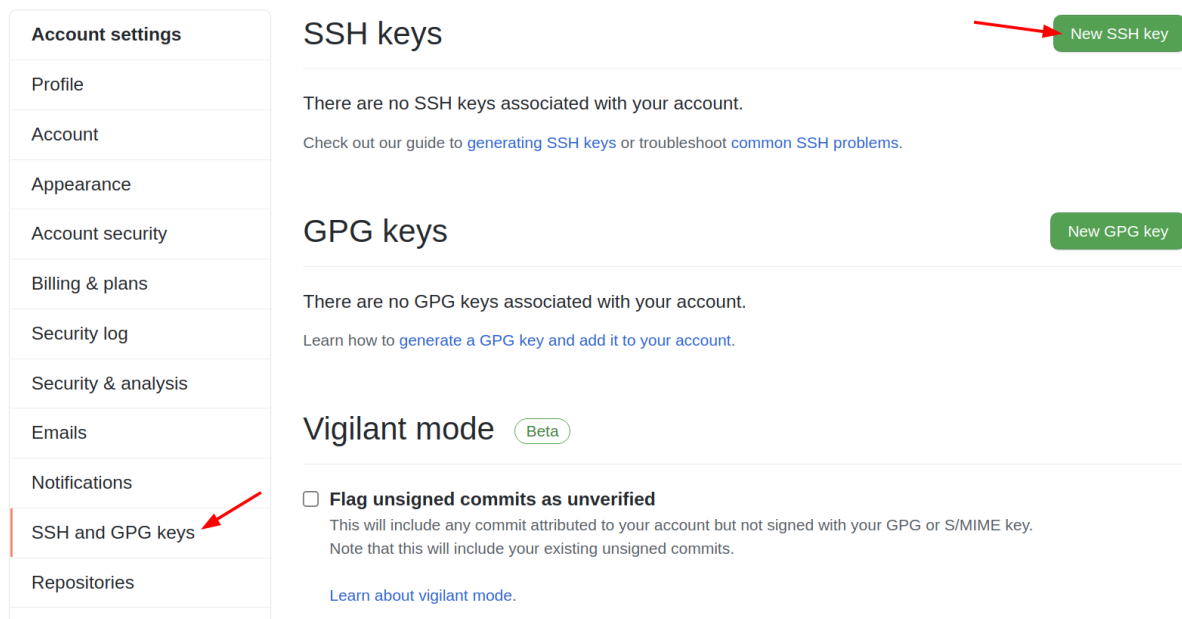


Fig. 13.3: settings

Em SSH and GPG keys acesse New SSH Key



The screenshot shows the GitHub account settings page. On the left is a sidebar menu with options: Account settings, Profile, Account, Appearance, Account security, Billing & plans, Security log, Security & analysis, Emails, Notifications, SSH and GPG keys (highlighted with a red arrow), and Repositories. The main content area is divided into three sections: SSH keys, GPG keys, and Vigilant mode. The SSH keys section has a 'New SSH key' button (pointed to by a red arrow) and text stating 'There are no SSH keys associated with your account.' The GPG keys section has a 'New GPG key' button and text stating 'There are no GPG keys associated with your account.' The Vigilant mode section is marked as 'Beta' and includes a checkbox for 'Flag unsigned commits as unverified'.

Account settings

- Profile
- Account
- Appearance
- Account security
- Billing & plans
- Security log
- Security & analysis
- Emails
- Notifications
- SSH and GPG keys**
- Repositories

SSH keys

There are no SSH keys associated with your account.

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

[New SSH key](#)

GPG keys

There are no GPG keys associated with your account.

Learn how to [generate a GPG key and add it to your account](#).

[New GPG key](#)

Vigilant mode Beta

☐ **Flag unsigned commits as unverified**

This will include any commit attributed to your account but not signed with your GPG or S/MIME key. Note that this will include your existing unsigned commits.

[Learn about vigilant mode.](#)

Fig. 13.4: ssh

Cole o conteúdo da chave publica e adicione um titulo a mesma

SSH keys / Add new

Title

codeanywhere

Key

ssh-rsa

AAAAB3NzaC1yc2EAAAADAQABAAQCMXSCNLRZRis
MIIAghIGyu4BcGqM/Ktp6D8SXP7FBGNANrvWwxXLs2xoHr
Q32MeX/fmsplFPcpXsDD1byxPrEsshz5zUQ/zti2nwZxwYYL
IFKf3wjTrJC1Wt09pkTIRAL9QR9EbZ3hHo702BRv8teK77sd.
ksEs93P1c5R71CVOmA31nUZb4CMBPqfJNC4p9ZQdbVjYr

Add SSH key

Fig. 13.5: ssh

ATENÇÃO: Após ao termino do curso lembre-se de remover esta chave.

Os passos são os mesmos que fizemos anteriormente, a diferença está no comando

```
$ git remote add origin git@github.com:<USER>/estudo-git.git
```

Este comando irá adicionar uma origem remota para nosso repositório, fazendo com que o git conheça para onde deve ser enviado o código quando executarmos um comando de `push`

Execute o comando `git remote add origin` > Lembre-se de utilizar o comando para seu usuário, ou o mesmo não irá funcionar.

Este comando irá criar uma fonte chamada `origin` com o endereço do seu repositório, podemos verificar através do comando

```
$ git remote -v
```

Após isto iremos modificar o nome da nossa branch padrão para `main`.

Em um passado não tão distante, os termos Master/Slave eram utilizados para se referir aos nós e as ações na qual eles desempenhavam. Em 2020 um forte movimento aconteceu para acabar com essa terminologia, uma vez que ela traz referências a escravidão. Não é sobre aguardar alguém ser ofendido. É sobre remover estes termos com base em coisas terríveis e desumanas.

```
$ git branch -M main
```

E utilizaremos o comando `push` para enviar o código da nossa branch local para a branch `main` da nossa `origin` remota

```
$ git push origin main
```

Atualize a página do github para ver seu código no repositório remoto.

Ainda precisamos adicionar um arquivo README.md em markdown para identificar nosso projeto.

Vamos criar um novo arquivo e adicionar um conteúdo ao mesmo

```
$ touch README.md
```

Clique no arquivo para editá-lo

```
# Repositório de Estudos git  
Repositório utilizado para aprender os primeiros passos e comandos básicos do git.
```

Salve o arquivo, adicione o mesmo a um commit e envie para o repositório remoto

```
$ git add README.md  
$ git commit -m "Adicionando arquivo README.md"  
$ git push origin main
```

Atualize o repositório e perceba que o README.md agora é exibido na página principal.

Utilizamos muito o markdown para criar arquivos de texto estilizados.

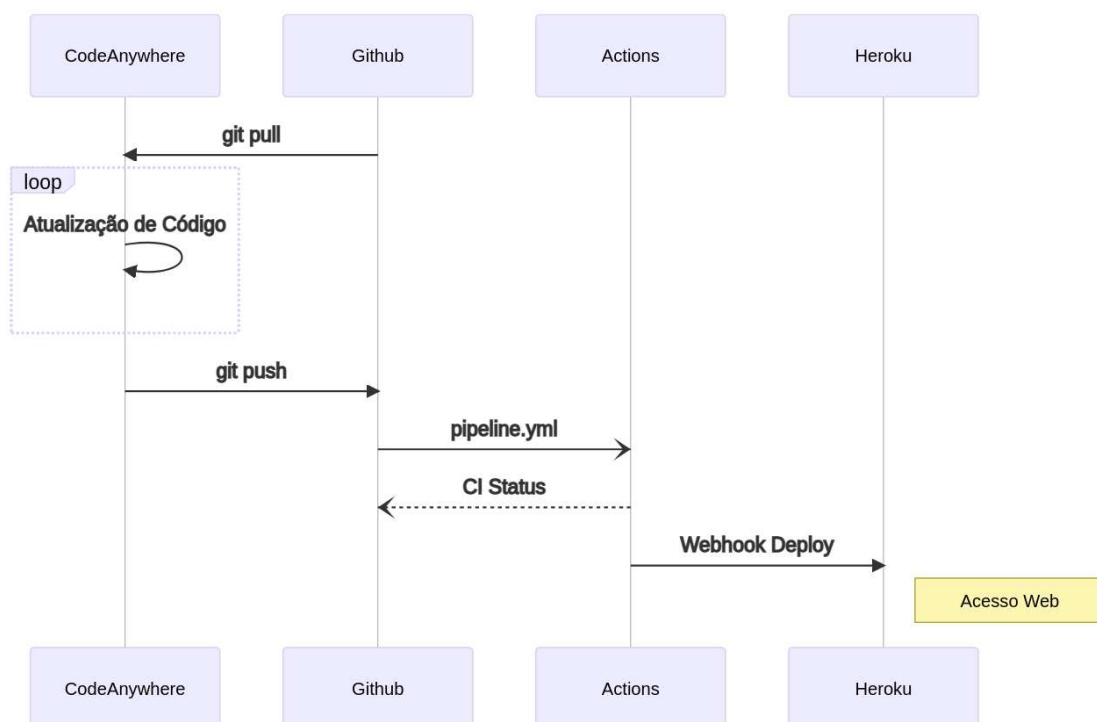
Todos os servidores de git com interface web tem uma função para renderizar o arquivo README.md, é muito importante que tenhamos este arquivo para identificar nosso projeto e dar instruções para quem quiser contribuir com o código.

Agora que aprendemos os primeiros passos com o git e a utilizar a plataforma do code anywhere e do github, podemos prosseguir para nossa pipeline DevOps!

14

Criar uma Pipeline DevOps

Agora vamos criar nossa primeira pipeline DevOps




```
sequenceDiagram
    participant CodeAnywhere
    participant Github
    participant Actions
    participant Heroku

    Github->>CodeAnywhere: git pull
    loop
        CodeAnywhere->>CodeAnywhere: Atualização de Código
    end
    CodeAnywhere->>Github: git push
    Github->>Actions: pipeline.yml
    Actions-->>Github: CI Status
    Actions->>Heroku: Webhook Deploy
    Note right of Heroku: Acesso Web
```

Utilizaremos o CodeAnywere para efetuar toda a atualização do código para enviarmos ao github. Faremos o processo de Deploy através do Github Actions que atualizará em nosso repositório os status do CI, e após a integração concluir o Github Actions irá efetuar o deploy da aplicação no Heroku que estará disponível para todos os usuários da internet.

Começando nossa pipeline

Iremos fazer o deploy da aplicação DevLpsLab-HelloWorld que é uma aplicação web em python utilizando flask.

Nosso primeiro passo é acessar o endereço do repositório em <https://github.com/4linux/DevOpsLab-HelloWorld/> e efetuar o fork do repositório para nossa conta.

Ao abrir o repositório no canto superior direito basta clicar no simbolo Fork.



Fig. 14.1: Fork

Fork quer dizer “Garfo”, quando fazemos o fork de um repositório estamos dando uma “garfada” no código origem para experimenta-lo, a partir deste fork teremos o código identico ao código de origem no estado atual, podendo assim personalizar o código e modifica-lo de acordo com nossas necessidades



Fig. 14.2: Forked

Agora que nós efetuamos o fork do nosso repositório, podemos efetuar o clone do mesmo clicando no botão verde Code e em seguida copiando a url de clone.

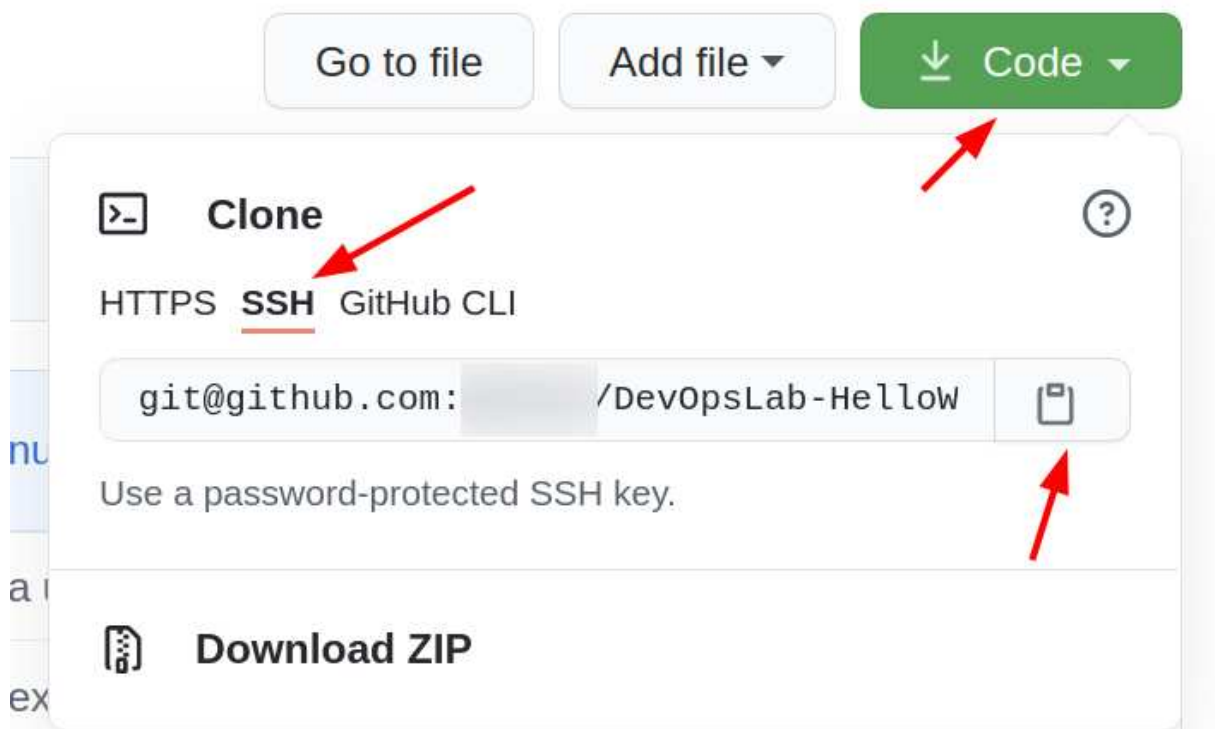


Fig. 14.3: Clone

Vamos voltar para nosso CodeAnywhere, voltar para a pasta origem do workspace e efetuar o clone em nosso terminal

```
$ cd ~/workspace
$ git clone git@github.com:<USER>/DevOpsLab-HelloWorld.git
$ cd DevOpsLab-HelloWorld
```

Verifique que agora temos nosso DevOpsLab-HelloWorld em nosso workspace.

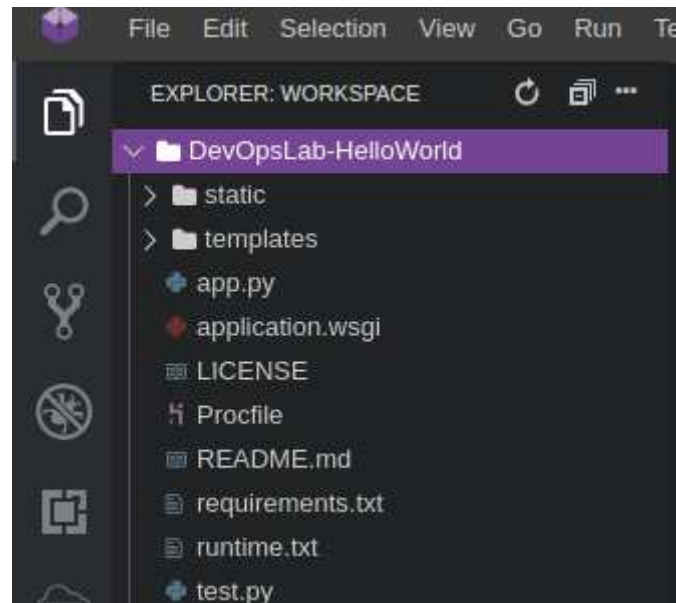


Fig. 14.4: code

Plan

Antes de começar a montar nossa pipeline, vamos criar um board no github para que possamos efetuar o planejamento do nosso software.

Abra o github e vá até o projeto “forkado” e clique em Projects e Create a Project

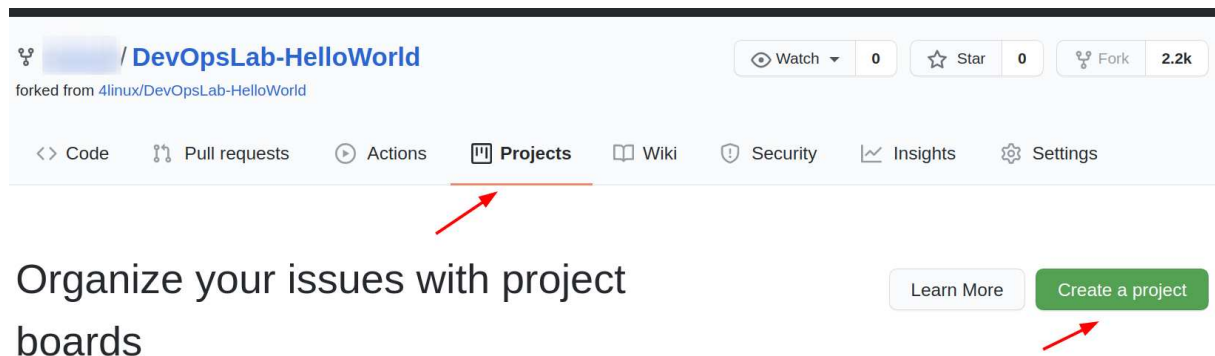


Fig. 14.5: Projects

Em Project board name coloque um nome para seu projeto, utilizaremos o nome de Pipeline DevOps

Coloque também uma descrição para o projeto, selecione o template Basic Kanban e clique em Create Project

Create a new project

Coordinate, track, and update your work in one place, so projects stay transparent and

Project board name

Pipeline DevOps

Description (optional)

Projeto de Pipeline DevOps para o DevOps Lab -Hello World

Project template

Save yourself time with a pre-configured project board template.

Template: Basic kanban ▼

Create project

Fig. 14.6: Projects

Com isso teremos um quadro kanban para utilizarmos, iremos primeiramente deletar os cards que foram criados automaticamente clicando nos ... do card e em seguida em Delete Note

Vamos criar os cards na coluna To do que significa “a fazer”

- Criar o APP no heroku
- Coletar Chave API do Heroku
- Configurar Secrets no github

- Criar pipeline do github actions
- Efetuar o primeiro deploy da aplicação
- Modificar a aplicação e efetuar o deploy

Nosso fluxo será

To Do -> In Progress -> Done A fazer -> Em Progresso -> Concluído

Criando o APP

Antes de começar, vamos mover o card Criar o APP do Heroku para In Progress

Vamos acessar o website do Heroku para criarmos um novo APP

Acesse <http://dashboard.heroku.com/apps> e clique em Create new app

Defina um nome para sua aplicação.

O nome da aplicação deve ser um nome único no mundo inteiro, minha sugestão é que vocês coloquem o nome devops-

Deixe a região como Estados Unidos e clique em Create App

Mova o card para concluído

Coletando API Key Heroku

Antes de começar, vamos mover o card Coletar chave API do Heroku para In Progress

Para configurarmos nossas integrações, precisamos criar secrets para que o Github e o Heroku possam se comunicar.

Agora que temos nosso app criado, vamos clicar no nosso perfil no canto superior direito e em seguida vamos em Account Settings

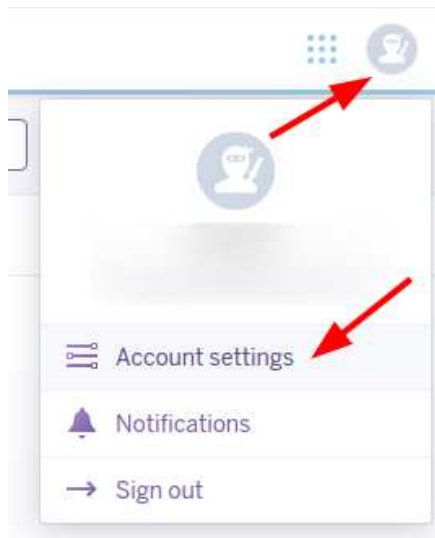


Fig. 14.7: Account Settings Heroku

Ao final da página vamos clicar em Reveal em API Key e vamos copiar o valor



Fig. 14.8: API Key Heroku

Mova o card para concluído

Configurar Secrets no github

Antes de começar, vamos mover o card Configurar Secrets no Github para In Progress

Voltaremos agora para o nosso fork no github e vamos clicar em Settings , Secrets e New Repository Secret

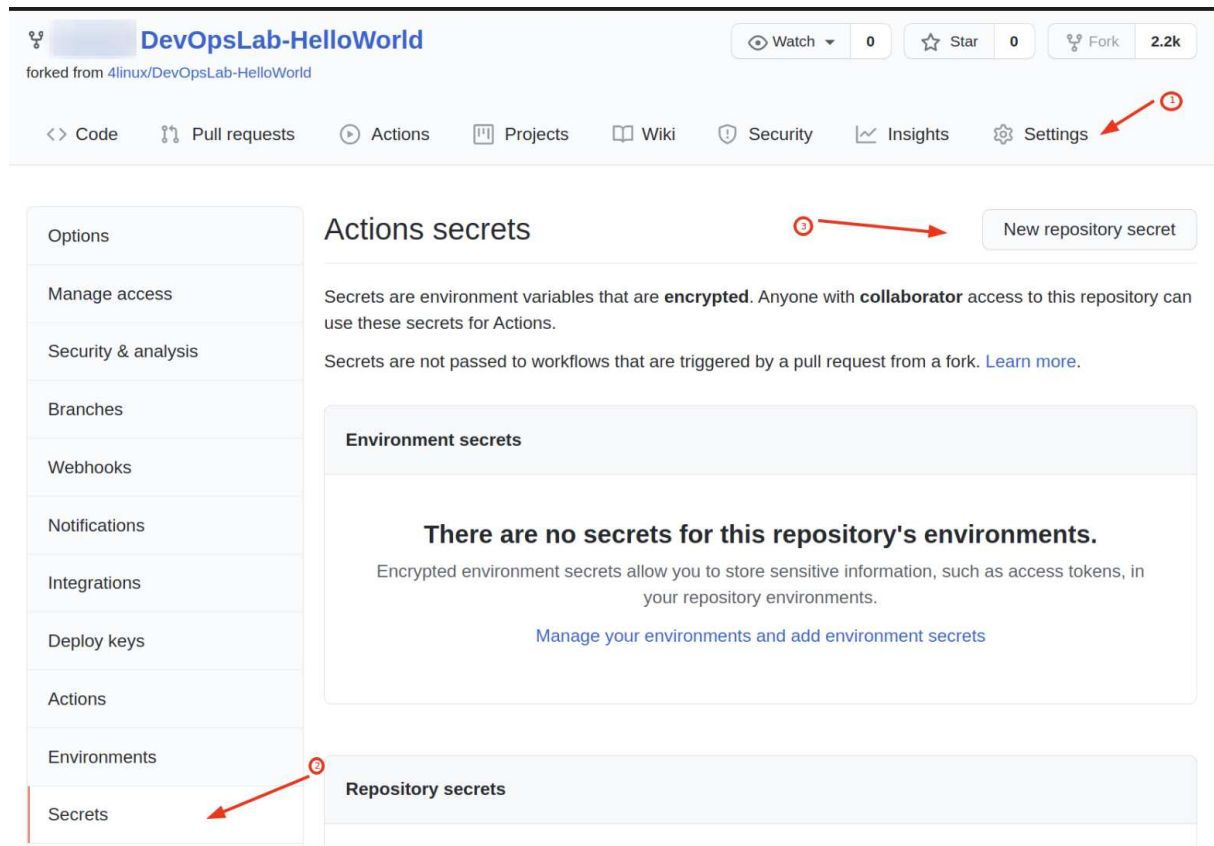


Fig. 14.9: Github Secrets

Vamos criar 3 secrets:

HEROKU_API_KEY -> Valor copiado anteriormente
 HEROKU_APP_NAME -> Nome da aplicação criada no heroku
 HEROKU_USER_EMAIL -> Email utilizado para conta do heroku

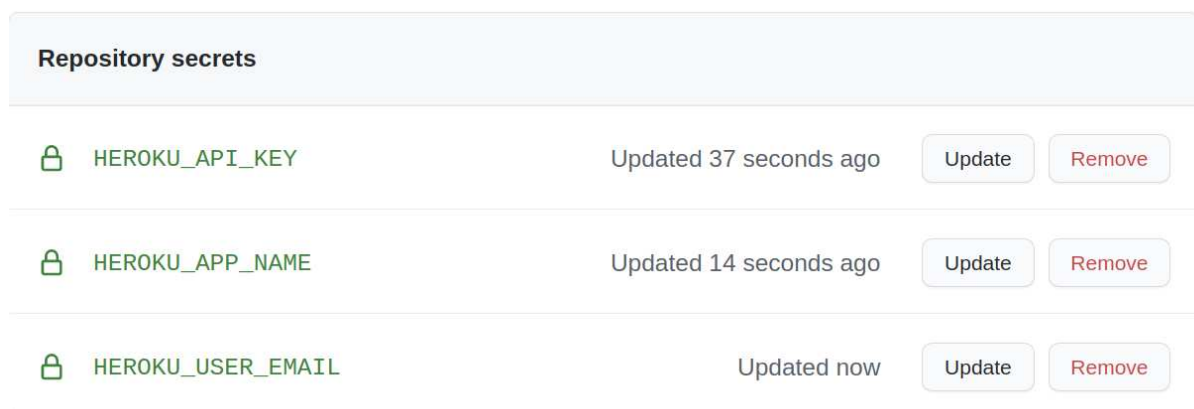


Fig. 14.10: Github Secrets

Mova o card para concluído

Agora que temos nossos secrets configurados, podemos configurar nossa pipeline no Github Actions

Github Actions

Antes de começar, vamos mover o card Criar pipeline do github actions para In Progress

Github Actions é uma ferramenta para automação de workflow para implementar CI/CD diretamente no github.

Para implementar o github actions precisamos de uma estrutura de diretórios dentro do nosso repositório

```
.github
├── workflows
└── pipeline.yml
```

Todos os arquivos dentro da pasta workflows serão lidos automaticamente pelo github para aplicar os padrões de actions

Em nosso codeanywhere vamos criar a estrutura de diretórios e o arquivo de pipeline de teste

```
$ mkdir -p .github/workflows
$ touch .github/workflows/pipeline.yml
```

Vamos abrir o arquivo .github/workflows/pipeline.yml em nosso editor e adicionar o conteúdo

```
name: CI - Lint / Tests / Deploy
on: [push]
jobs:
  lint:
    runs-on: ubuntu-20.04
    steps:
      - uses: actions/checkout@v2
```

```

- name: Set up Python 3.9
  uses: actions/setup-python@v2
  with:
    python-version: 3.9

- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    pip install flake8

- name: Lint with flake8
  run: |
    # stop the build if there are Python syntax errors or undefined names
    flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
    # exit-zero treats all errors as warnings. The GitHub editor is 127 chars wide
    flake8 . --count --exit-zero --max-complexity=10 --max-line-length=127 --
      statistics

tests:
  needs: lint
  runs-on: ubuntu-20.04
  strategy:
    matrix:
      python-version: [3.5, 3.6, 3.7, 3.8, 3.9]

  steps:
    - uses: actions/checkout@v2

    - name: Set up Python ${ matrix.python-version }
      uses: actions/setup-python@v2
      with:
        python-version: ${ matrix.python-version }

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install flake8 pytest
        if [ -f requirements.txt ]; then pip install -r requirements.txt; fi

    - name: Integrations Test
      run: |
        python test.py

deploy:
  needs: tests
  runs-on: ubuntu-20.04

  steps:
    - uses: actions/checkout@v2

    - name: Set up Python 3.9
      uses: actions/setup-python@v2
      with:
        python-version: 3.9

    - name: deploy into heroku
      uses: akhileshns/heroku-deploy@v3.12.12
      with:
        heroku_api_key: ${ secrets.HEROKU_API_KEY }
        heroku_app_name: ${ secrets.HEROKU_APP_NAME }
        heroku_email: ${ secrets.HEROKU_USER_EMAIL }

```

Neste arquivo estamos definindo nossa pipeline de Lint, Testes e Deploy, que será executada toda vez que o repositório receber um push ou seja, toda vez que o código for enviado para o github.

Será executado em máquinas ubuntu 20.04 com os passos de teste sendo executados para python 3.5 até 3.9 a fim de garantirmos que o código funciona para python 3.5 ou superior.

Nossa pipeline possui 3 passos: 1. Verificar lint com flake8 2. Testar o código nas versões Python 3.5 até 3.9 3. Deploy no heroku

Vamos alterar também nosso arquivo README.md nas linhas 4 a 6 removendo o comentário e trocando o <USER> pelo nosso username do github.

```

```

Vamos enviar nosso código para o github

```
$ git add --all
$ git commit -m "Adicionando Pipeline DevOps"
$ git push origin main
```

No github, clique em actions e em seguida no workflow em execução

Podemos ver todos os passos sendo executados e inclusive podemos clicar em cada passo para verificar o que está sendo feito.



Fig. 14.11: Pipeline

vamos mover o card Efetuar o primeiro deploy da aplicação para In Progress

Podemos ir agora no nosso Heroku e verificar o app

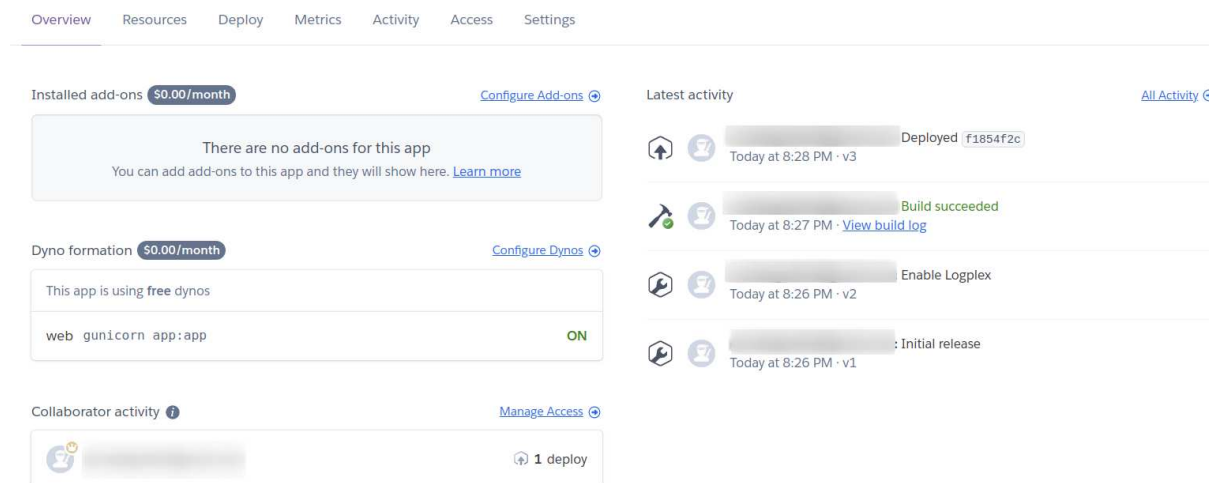


Fig. 14.12: Heroku APP

Para mais informações podemos clicar em view build log

Para acessar nosso app basta abrirmos o endereço

`https://<app>.herokuapp.com`



Fig. 14.13: APP

Mova os cards para concluído

Modificar a aplicação e verificar o deploy

Antes de começar, vamos mover o card Modificar a aplicação e verificar o deploy para In Progress

Agora que temos nossa aplicação funcional, vamos editar nosso software e nosso teste.

No code anywhere, navegue até o arquivo `index.html` no diretório `templates`

Vamos alterar a linha 13:

```
<div class="alert">Escreva uma Mensagem para o Cabeçalho da Pagina.</div>
```

Altere a mensagem para Pipeline DevOps - <Nome e Sobrenome> e salve o arquivo.

Edite o arquivo `test.py`. Vamos alterar a linha 22:

```
self.assertRegex(result.data.decode(), "Escreva uma Mensagem para o Cabeçalho da Pagina.")
```

Altere a mensagem para Pipeline DevOps - <Nome e Sobrenome> e salve o arquivo.

Adicione os arquivos e faça o commit

```
$ git add --all
$ git commit -m "Alterando o cabeçalho"
$ git push origin main
```

No github, clique em actions e em seguida no workflow em execução

Podemos ver todos os passos sendo executados e inclusive podemos clicar em cada passo para verificar o que está sendo feito.

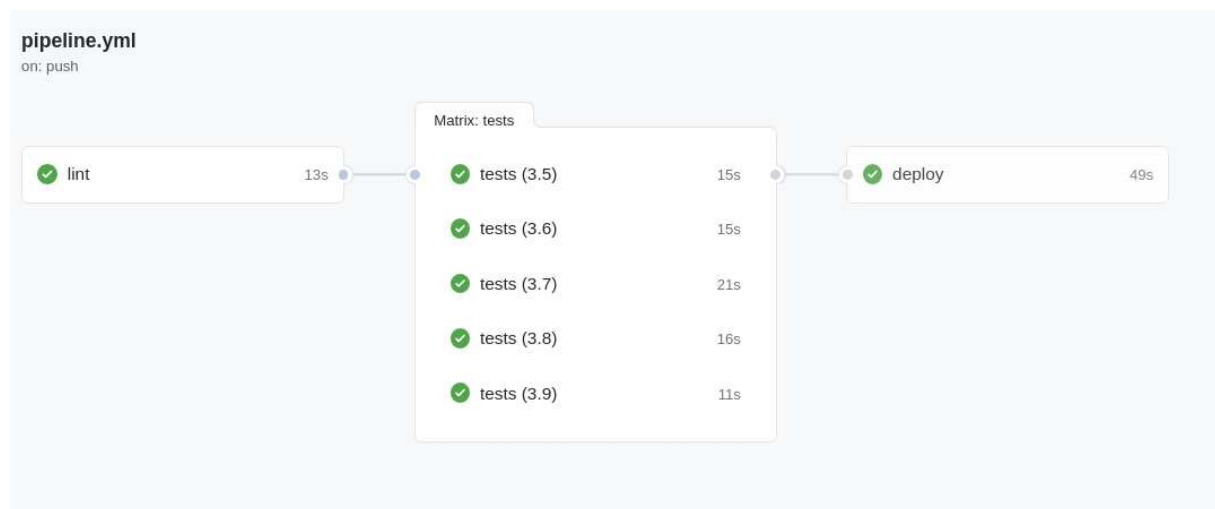


Fig. 14.14: Pipeline

Podemos ir agora no nosso Heroku e verificar o app com a mensagem de cabeçalho que modificamos

<https://<app>.herokuapp.com>

Pipeline DevOps - Caio Delgado

4.Linux

Open Software Specialists

Não fique parado, continue estudando sobre DevOps e Containers!

Trilha DevOps

- 524 - CI/CD: INTEGRAÇÃO E ENTREGA CONTINUA COM GIT, JENKINS, NEXUS E SONAR
- 525 - INFRAESTRUTURA ÁGIL COM PRÁTICAS DEVOPS
- 527 - DEVSECOPS: SEGURANÇA EM INFRAESTRUTURA E DESENVOLVIMENTO ÁGIL
- 528 - PRÁTICAS DE CONTINUOUS MONITORING PARA UMA INFRAESTRUTURA ÁGIL

Trilha Containers

- 540 - DOCKER: ADMINISTRAÇÃO DE CONTAINERS - DCA
- 541 - KUBERNETES: ORQUESTRAÇÃO DE AMBIENTES ESCALÁVEIS
- 542 - OPENSIFT: ADMINISTRAÇÃO PAAS EM CONTAINERS/
- 543 - GERENCIAMENTO DE CLUSTER KUBERNETES COM RANCHER

Fig. 14.15: APP2

Mova o card para concluído # Entrega do produto final

Se você chegou até aqui, meus parabéns, você conseguiu efetuar um deploy de uma aplicação de forma automática utilizando uma pipeline composta de linter, testes e deploy.

Caso deseje você pode continuar modificando os arquivos do curso para experimentar cenários diferentes.

Só não se esqueça de remover a chave SSH no github quando terminar, uma vez que essa chave criada anteriormente pode ter acesso aos seus repositórios, para fazer isto basta ir no github, clicar em seu perfil > settings > ssh and GPG keys > delete na chave da codeanywhere.

E agora que já entregamos nosso produto final, o que podemos fazer?

Não existe uma fórmula certa para o sucesso, mas que tal uma ajudinha na sua escolha de cloud ou on-premises? E o que é necessário para todos?

Bem, chegou a hora de dar o próximo passo e vamos falar sobre isto na próxima aula!#

Próximos passos para trabalhar com DevOps

Existem alguns assuntos da área de DevOps que são necessários para qualquer pessoa que deseja se tornar um Engenheiro DevOps ou um Site Reliability Engineer.

Vamos falar um pouco sobre estes temas:

git

Como o código está ligado diretamente ligado a DevOps, qualquer profissional precisa saber o mínimo de git para se dar bem na área. Não precisa ser um especialista, mas é preciso estar confortável trabalhando com git.

Redes

Naturalmente tudo se conecta através de redes, precisamos entender conceitos básicos como dhcp, dns, portas, ip, dentre vários outros. Saber redes claramente te deixa a frente de outras tecnologias. Como um sábio já disse no passado, se não tá funcionando, provavelmente o problema são redes. Se não for, é dns, que dá no mesmo.

Brincadeiras à parte, muitos profissionais não conhecem esses conceitos básicos e podem perder horas tentando encontrar algum problema que geralmente é simples, por isto vemos muitas pessoas falando que “provavelmente o problema é de redes”.

Linux

É até estranho dizer isso, mas sabia que maior parte da internet é composta de sistemas Unix/Linux? Fora isso, todos os sistemas OpenSource estão diretamente ligados ao Linux e até mesmo o nosso tão querido DevOps. Tá esperando o que então? Bora aprender Linux!

Monitoramento

A chave para um sistema funcionando é prever os problemas quando possível, sobretudo monitorar de maneira eficiente. Para isto, podemos utilizar o zabbix, greylog, prometheus, elasticstack dentre várias outras ferramentas. Afinal, qual escolher? Vai depender apenas de onde você vai aplicar esse monitoramento. # Próximos passos Para trabalhar com Cloud

Caso sua escolha seja trabalhar com cloud, tenho algumas indicações de ferramentas e temas para que você coloque em seu objetivo de estudo.

Aprenda uma Cloud

As clouds mais populares atualmente são a Amazon AWS e a Google Cloud Platform, porém, a Microsoft Azure também está chegando com força no mercado. Escolha uma das clouds para se especializar! Uma dica pra quem ainda está em dúvida é pesquisar aquela empresa que você se identifica e sonha em trabalhar e verificar qual cloud ela usa, ou até mesmo procurar no linkedin qual cloud tem mais vagas em aberto.

Aprenda Terraform

Quando falamos de cloud, podemos dizer que ao menos 90% das vagas pedem conhecimento sobre Terraform, que é uma ferramenta de infraestrutura como código que trabalha de maneira agnóstica a clouds, sendo assim, você consegue versionar e efetuar o deploy de infraestrutura de maneira rápida e versionável.

Aprenda Containers

Containers estão diretamente ligados à cloud, então, que tal estudar um pouco de Docker e em seguida Kubernetes? Essas ferramentas são amplamente utilizadas, sendo um item necessário na sua bagagem de ferramentas devops para cloud.

Outros

Claramente existem outros temas para você aprender quando falamos de cloud, mas por hora acho que já temos uma boa linha para dar segmento aos estudos. # Próximos passos Para trabalhar On Premises

Caso sua escolha seja trabalhar on premises, tenho algumas indicações de ferramentas e temas para que você coloque em seu objetivo de estudo.

Aprenda Sistemas Operacionais

Já falamos de linux antes né? Mas porque eu falo novamente sobre linux? Você vai lidar diariamente com máquinas linux no seu dia a dia, então é bom reforçar essa dica! Estude Linux, e não só isso, mas aprenda sobre Sistemas Operacionais.

Aprenda Ansible

Precisamos automatizar as instalações quando falamos de on-premises, e para isto precisamos de alguma ferramenta forte para resolver este problema. O Ansible tem uma ótima curva

de aprendizado e resolve boa parte dos problemas. Você também pode optar pelo Puppet, é um pouco mais complicado, mas funciona muito bem para gerenciar parques de máquinas on-premises.

Outros

Claramente existem outros temas para você aprender quando falamos de on-premises, mas por hora acho que já temos uma boa linha para dar segmento aos estudos.