Containers para pessoas ansiosas

Índice

Sobre o autor

Meu nome é **Fábio Luciano**, trabalho como arquiteto de soluções na empresa SONDA.

Entre alguns fatos sobre mim, listo os seguintes como mais importantes:

- Sou gay e casado □□□;
- Sou apaixonado por tecnologia e por software livre □;
- Sou **ansioso** pra cacete □.

Acho importante dizer essas coisas íntimas, por que acredito que isso, no final das contas, me resume em alguns aspectos. E como falaremos intimamente sobre containers, acho interessante abrir essas *verdades* sobre mim. Além do fato de serem verdades, o último ítem, a ansiedade, foi o que possibilitou meus estudos nos tópicos abordados, e também está possibilitando a escrita deste documento, já que estou usando o tempo em que redijo para me distrair da muvuca da minha cabeça.

Acho importante dizer também que esse documento/apresentação é pra ser despretensioso. Não pretendo ser coloquial, mas têm coisas que não podemos fugir do nome real, mas vou tentar usar analogias para explicar alguns assuntos que julgar necessário explicar um pouco a mais.

Logo, a confecção deste documento é quase que terapêutica, por que irei:

- Organizar meus conhecimentos sobre o assunto em algum lugar;
- Enfrentar o medo de falar publicamente; e
- **Disseminar o conhecimento**.

Ah! Se você têm interesse por meu currículo profissional, aqui está alguns links pra que dê uma olhada:

- https://fabioluciano.dev
- https://integr8.me
- https://github.com/fabioluciano
- https://people.php.net/fabioluciano
- https://naoimporta.com
- fabio@naoimporta.com

Sobre este documento

Sobre a distribuição deste documento

- Repositório
- HTML
- Documento PDF
- Apresentação

Todo

- Este é um documento em construção

Tecnologias

- DocBook
- AsciiDoc
- AsciiDoctor
- PlantUML
- Docker
- Shell
- Travis
- Github Pages
- Asciinema

Badges

[Apache] | *https://img.shields.io/badge/license-Apache-blue.svg*

[GitFlow] | *https://img.shields.io/badge/Conversational_Development-friendly-yellow.svg*

[Conversational Development] | *https://img.shields.io/badge/GitFlow-friendly-yellow.svg*

[Conventional Commits] | *https://img.shields.io/badge/Conventional_Commits-1.0.0-yellow.svg*

O Que veremos

Sempre quis falar sobre containers, porém, nunca havia encontrado uma abordagem interessante ao assunto, visto a quantidade infinita de conteúdo sobre este assunto na internet. Com este contexto, decidi não falar sobre `Docker` especificamente, mas sim sobre todas as tecnologias que o tornou possível e além disso, a partir da disseminação de tecnologias relacionadas a containers, ilustrar como sua existência mudou completamente a industria de infraestrutura e desenvolvimento de software.

Não é objetivo deste documento ser a "fonte da verdade" sobre conteinerização ou sobre `Docker`, mas sim um compilado de informações sobre o assunto que compilei com o tempo. Por diversas vezes, utilizarei de uma linguagem despojada e despretensiosa e sempre que possível atribuirei fontes sobre o assunto. Vou te ajudar amigo ansioso.

Caso encontre algum problema, abra uma issue na página do projeto no GitHub.

Para seguir uma linha de raciocínio, escolhi dividir os assuntos abordados em seis partes. São elas:

Open Source

Na *primeira* parte deste documento, conversaremos sobre **Open Source**. Discutiremos sobre as aplicações do movimento por diversos setores da indústria(até o alimentício). Falarei sobre as principais licenças que caracterizam projetos do movimento livre, e também sobre os modelos implementados para organização destes projetos, falaremos sobre tipos de distribuições, como exemplo das possibilidades da implementação do `código-livre`, passearemos sobre grandes projetos, e por final, lhe farei um convite.

Sistemas Operacionais e sua arquitetura

Nesta *segunda* parte , elucidarei como os sistemas operacionais evoluíram o suficiente para possibilitar tecnologias de virtualização disponíveis atualmente. Além disso, apresenta o `Kernel` do Linux de forma a introduzir conceitos utilizados por grande maioria dos provedores desta tecnologia.

Conteinerização

Na *terceira* parte, vamos andar pelos caminhos mais íntimos da containerização. Observaremos como os conceitos há muito utilizados pelo Kernel do linux foram utilizado para criar soluções incríveis

Conversational Development

Na *quarta* parte deste documento,

(Dev(Sec)?|Chat)Ops

Na *quinta* parte deste documento,

Arquitetura de Software

Na *sexta* parte deste documento,

Arquitetura de Soluções

Na *sétima* e última parte deste documento,

Open Source

Características

- Descentralização;
- Colaboração;
- Liberdade.

Licenças

- GPLv3;
- Apache;
- MIT.

Modelos e organizações

- A Catedral e o Bazar
  - Catedral
  - Bazar

Cases de Sucesso

- Linux
  - Metadistribuições
    - Slackware;
    - Gentoo;
    - Arch Linux;
  - Distribuições
    - Debian;
    - Red Hat;

Olhar pessoal

- As comunidades relacionadas são incríveis (irc, fórum, etc.);
- As documentações são incríveis (Arch Linux);
- Salvo exceções, sempre há alguém disposto a ajudar.

Sistemas Operacionais

Características

- Interface entre o usuário e o hardware
- Gerenciamento de dispositivos (Entradas e Saídas)
- Prover um ambiente para o funcionamento de programas
- Interface para o gerenciamento de dados
- Monitorar a saúde do hardware

Gostaria de falar sobre o Kernel do `Windows`, mas a verdade é que eu não sei nada sobre, e sinceramente, nunca procurei saber.

Kernel do Linux

♡

Características

- Tudo é representado como arquivo;
  - Pseudo filesystems
    - `dev`, `proc` e `sys`, `fd`
- Possui clara separação entre o espaço do usuário e o do kernel;
- Os processos são separados em `namespaces`;
- A Comunicação entre o `user space` e `kernel space` é feita por intermédio de bibliotecas ou sinais.

User Space

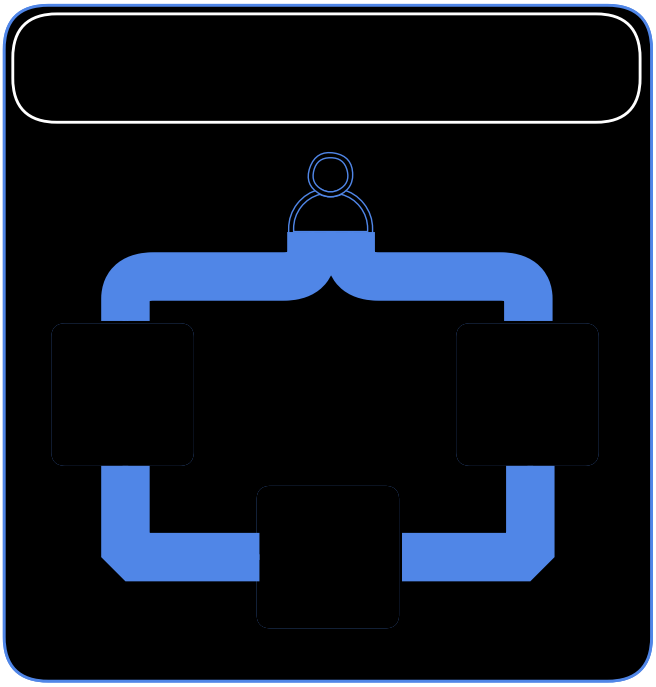

Figura 1. Representação do user space

- Onde os processos do usuário são executados;
- Região fracionada e limitada da memória.

Kernel Space

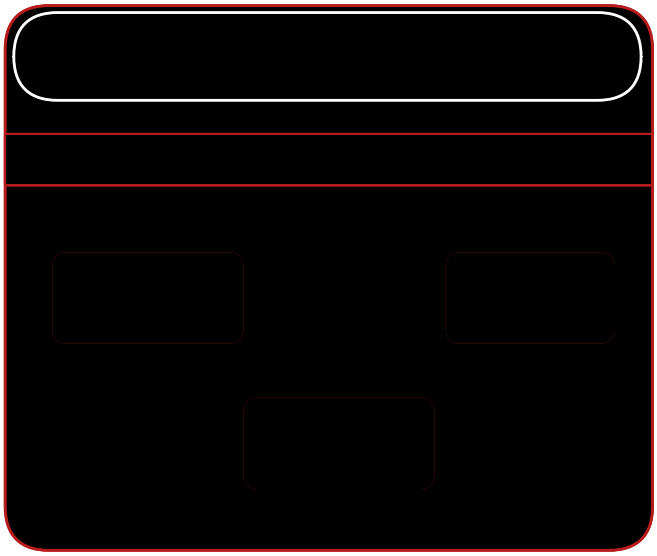

Figura 2. Representação do kernel space

- Onde os processos do kernel são executados;

- Região fracionada e privilegiada da memória.

System Calls

- Único meio de comunicação entre o `User Space` e `Kernel Space`;
- Gerenciamento de Processos, memória, arquivos, dispositivos, rede e etc.

Sinais

- **SIGHUP** - Reinicializa o processo;
- **GITTERM** - Termina de forma graciosa `Ctrl` + `c`;
- **SIGKILL** - Termina sem fazer nenhum tipo de checagem `Ctrl` + `d`.

Memory Management

- Gerenciamento da memória;
- Decide como os dados serão persistidos e recuperados;
- Memória virtual;
- Paginação;
- Controle de Acesso.

Scheduler

- Define em tempo de execução a alocação de recursos para um processo;
- Priorização dinâmica.

Drivers

- Também chamado de módulos;
- Interface de comunicação entre o `Kernel` e os devices.

```
lsmod ①
```

① Lista os módulos(drivers) carregados no Kernel.

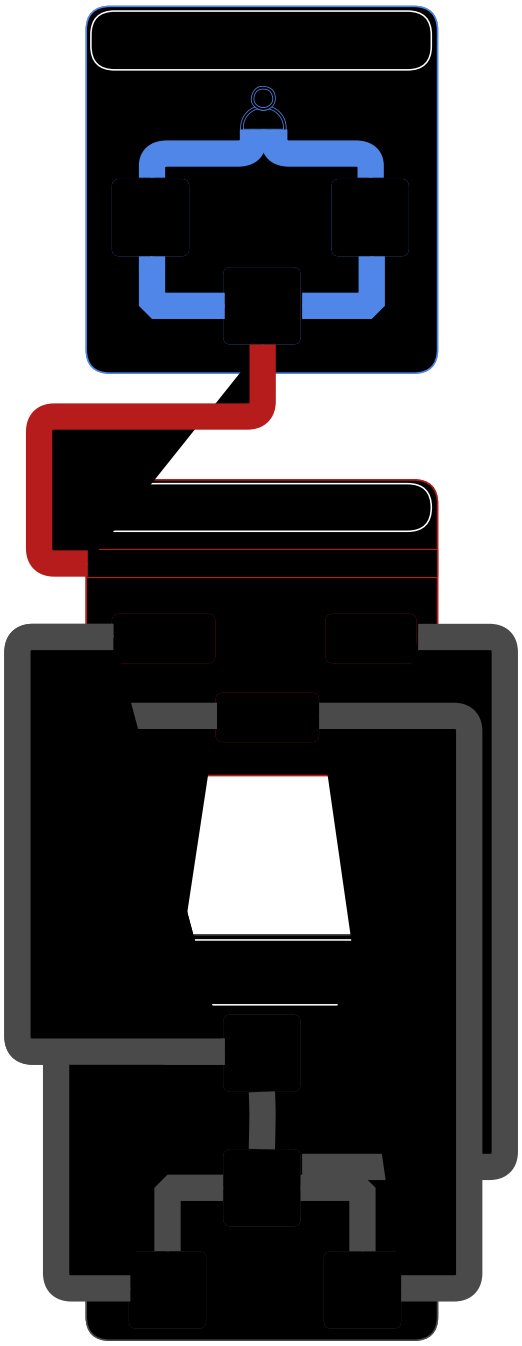

Figura 3. Representação da stack do Kernel do Linux

Representação da comunicação dos elementos da Stack do Kernel do Linux

```
Failed to generate image: PlantUML image generation failed:
/documents/src/resources/plantuml/resources/plantuml/plantuml.cfg (No such file or
directory)
@startuml
skinparam dpi 300

title Comunicação da aplicação com o kernel

box "User Space"
  participant Aplicação
  participant GLIBC
  participant vDSO
end box

box "Kernel Space"
  participant SysCall
  participant Kernel
  participant Hardware
end box


Aplicação -> GLIBC
alt vDSO
  create vDSO
  GLIBC -> vDSO
  vDSO --> GLIBC
else SysCall
  GLIBC -> SysCall
end
  SysCall -> Kernel
  Kernel -> Hardware

  Hardware --> Kernel
  Kernel --> SysCall
  SysCall --> GLIBC
  GLIBC --> Aplicação


@enduml
```

Exemplo

printf.c

```
#include <stdio.h> ①
#include <unistd.h> ①

void main () {
  printf(":)\n"); ②
  usleep(520000*1000);
}
```

① Importa a biblioteca de manipulação de entrada e saída

② Imprime na saída padrão

```
gcc printf.c -o printf ①
```

① Compila o programa e define o arquivo de saída

```
chmod +x ./printf ①
strace ./printf ②
```

① Concede permissão de execução ao programa

② Imprime todas as SysCalls chamadas pelo programa

[Execução no Asciinema] | *https://asciinema.org/a/M5H7Jl6TEr0kIlcRtWOTWdiMO.png*

Figura 4. Execução dos passos anteriores no Asciinema

Chroot (Jail)

- Muda o diretório raiz de um processo;
- Enclausuramento de recursos;

```
ldd ./printf ①
mkdir -p chroot/{lib,lib64,} ②
cp -v /lib/x86_64-linux-gnu/libc.so.6 ./chroot/lib ③
cp -v /lib64/ld-linux-x86-64.so.2 ./chroot/lib64 ③
cp ./printf ./chroot/bin ③
sudo chroot ./chroot/ /bin/printf ④
```

① Imprime as bibliotecas que o binário utiliza

② Cria a estrutura de diretórios necessário (FHS)

③ Copia as bibliotecas e binário para e estrutura criada

④ Exacuta o programa enclausurado

Namespaces

- Mount (`mnt`)
- Process (`pid`)
- Network (`net`)
- Interprocess Communication (`ipc`)
- UNIX Timesharing System (`uts`)
- User ID (`user`)
- CGroup
- Padrão namespace global;
- Os namespaces podem ser visualizados em `/proc/<processid>/ns/*`
- unshare
- setns
- clone

```
unshare --user /bin/bash ①
UNSHARED_PID=`echo $$` ②
cd /proc/$UNSHARED_PID/ns ③
ls -l ④
```

① Destaca a execução do binário em um namespace `user` diferente

② Imprime o `pid` do processo criado e o associa a uma variável

③ Muda o workdir para o diretório com a lista de namespaces do processo

④ Lista os namespaces

```
hostname ①
unshare --uts /bin/bash ②
hostname teste ③
hostname ④
Ctrl + d ⑤
hostname ⑥
```

① Recupera o hostname atual

② Destaca a execução do binário em um namespace `user` diferente

③ Altera o hostname da máquina

④ Recupera o novo hostname

⑤ Envia o `SIGNAL` SIGKILL

⑥ Imprime o hostname no processo principal

UnionFS

- Diretórios e arquivos em diferentes branches;
- Top-Down search nas layers;
- Todas as layers inferiores são `read-only`;
- COW(`copy-on-write`).

Containerização

- Empacotamento de código e aplicações;
- Empacotamento de dependências e versões;
- Empacotamento de configurações.
- **Reusabilidade** e **Reproducibilidade**.

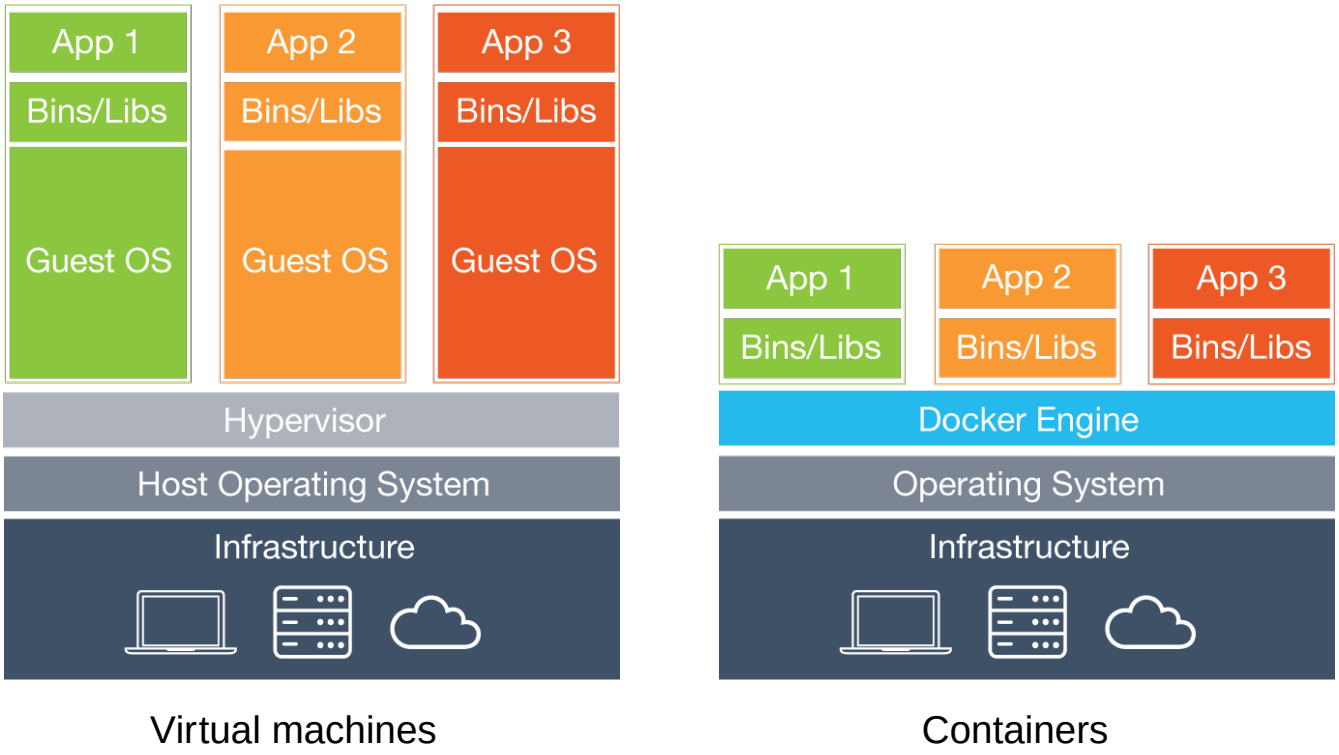

Figura 5. Representação da diferença entre VMs e containers

Virtual Machines

- Hardware
- Sistema Operacional(Host)
- Hypervisor - Habilita a virtualização/gerenciamento dos recursos
- Sistema Operacional(Hospedeiro)
- Binários e bibliotecas
- Aplicações

Containers

- Hardware
- Sistema Operacional(Host)
- Daemon(Daemon based)
- Binários e bicliotecas

- Aplicações

Vendors

Table 1. Principais vendors de ferramentas separadas por características

Daemon	**Daemonless**	**Systemd**
Docker	Buildah	Rkt
	Podman	

Docker

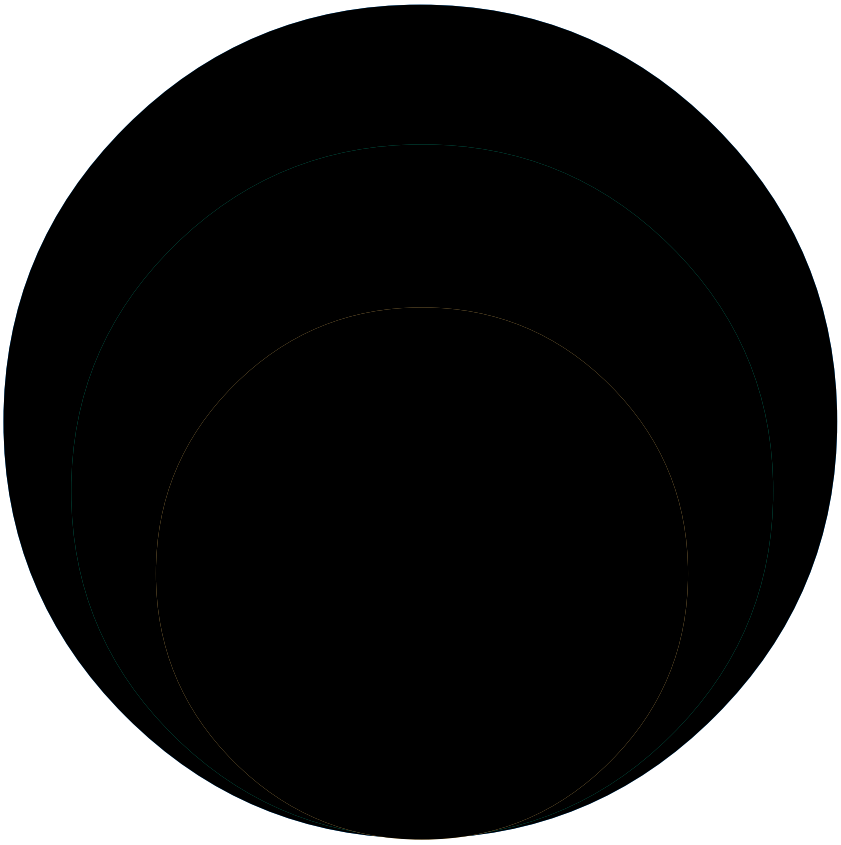

Figura 6. Representação de todos os elementos da Stack do Docker

- Mais popular software de containers;
- Elementos da stack:
  - Daemon
  - API
  - Client

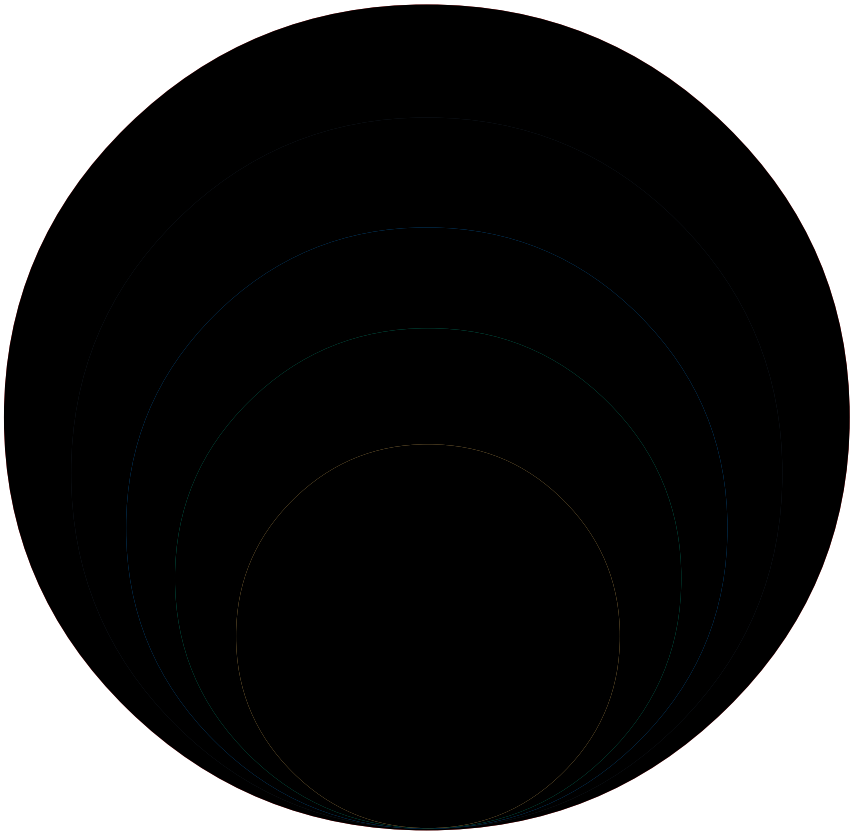

Figura 7. Representação de todos os elementos da Stack do Docker mais os elementos referentes ao Kernel do Linux

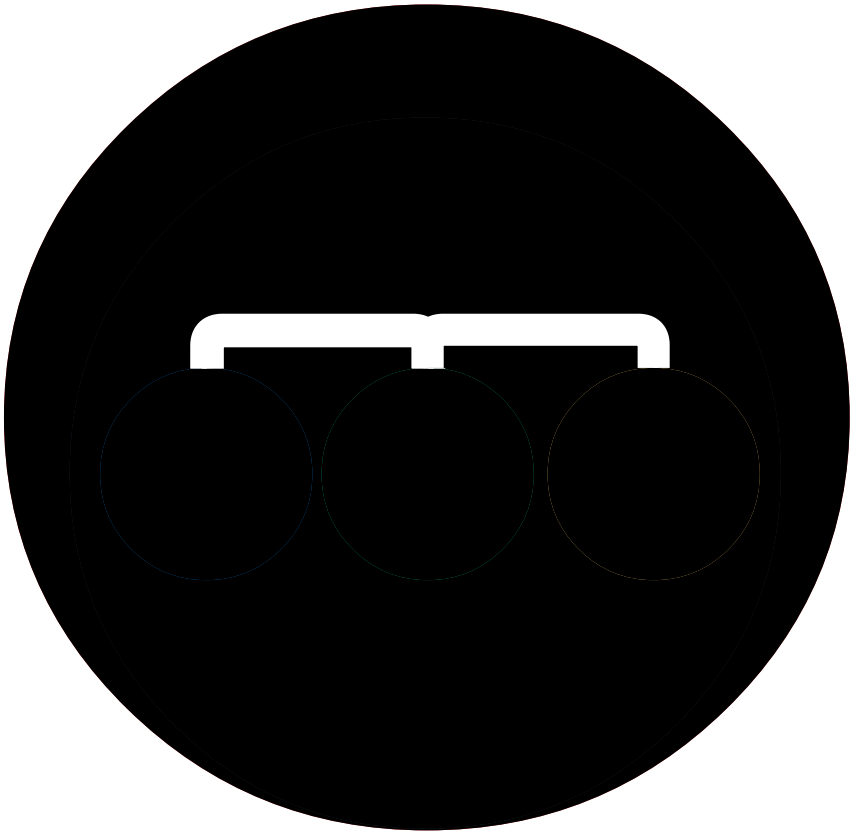

Figura 8. Representação real da stack do Docker

Ecosistema

- Docker Hub(Registrador)

Elementos

- Dockerfile
- Container
- Volumes
- Network

Dockerfile

- Segue a especificação `image-spec` do projeto `opencontainers`;
- Descreve procedimentos necessários para se criar um ambiente;
- Possui uma DSL(Domain-Specific Language)

Exemplo de arquivo Dockerfile

```
FROM alpine  ①
RUN apk update  ②
RUN apk add curl ③
```

① Informa a imagem base(conjunto de layers) que será utilizada

② Atualiza a lista de pacotes

③ Instala o pacote `curl`

- Uma imagem é um conjunto de layers sobrepostas;
- Cada instrução criará uma nova `layer` na imagem final;
- Cada layer aumenta o tamanho da imagem.

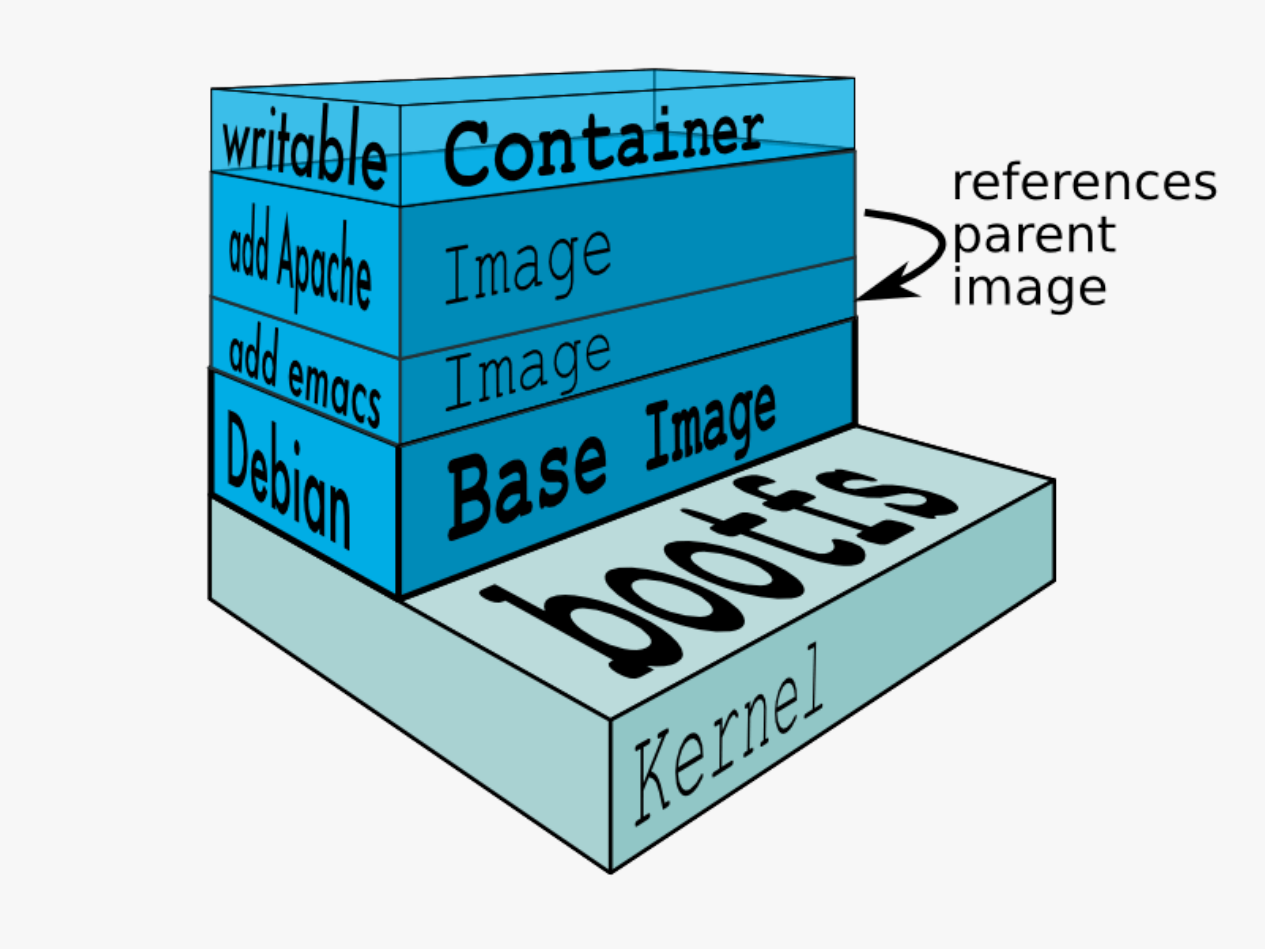

Figura 9. Representação das layers do Docker

Container

- Segue a especificação `runc` do projeto `opencontainers`;
- É uma imagem em execução com a adição de uma branch writtable no topo;
- Pode possuir volumes e networks;
- Controle de recursos com `CGROUPS` e `CAPABILITIES`.

Volumes

- Onde os dados serão persistidos;
- Possuem três tipos:
  - Host
  - Anônimo
  - Nomeado
- Pode ser estendido com a utilização de plugins(nfs)

Network

- Onde os dados do container serão trafegados
- Pode ser estendido com a utilização de plugins(cluster)
- Segue a especificação `cni` do projeto `containernetworking`;

Benefícios

- **Isolamento** - Uma imagem com binários e dependências.
- **Paridade** e **Portabilidade** - Prod/Dev
- **Entrega rápida** - Automação de build e deploy

Conversational Development (ConvDev)

- Desenvolver software é uma conversa;
  - O que precisa ser implementado;
  - O que precisa ser modificado;
  - Como será implementado;
  - Onde será implantado;
  - Como será controlado.
- É a aplicação do agile com um objetivo simples: Finalizar uma conversa;
- A conversa deve ter como objetivo a criação de um produto;
- Passa por todas as áreas da linha de negócio;
- Alinha-se a organização para que os processos sejam executados.

Princípios

- Ciclos de conversas pequenos;
- A conversa deve passar por todos os estágios do CDS;
- Abra a conversa, mas não espere consenso;
- Resultados orientados por conversação.

IaC e (Dev(Sec)?|Chat)Ops

Infraestrutura como código (IaC)

- Trata a infraestrutura como código;
- Está intimamente ligado às práticas do `DevOps`;
- Permite a adoção de práticas do desenvolvimento;
  - Controle de versão
  - Revisão por pares
  - Segmentação de ambientes
  - Testes

Ferramentas disponíveis

- **Terraform**
- Cloud Formation
- **Ansible**
- Chef
- Puppet

Terraform

- A infraestrutura é definida utilizando HCL;
- Utiliza grafos para controlar as dependências dos recursos;
- Possui implementação para diversos providers(AWS, Azure, Google, etc.);
- Todos os recursos necessários podem ser criados programáticamente.

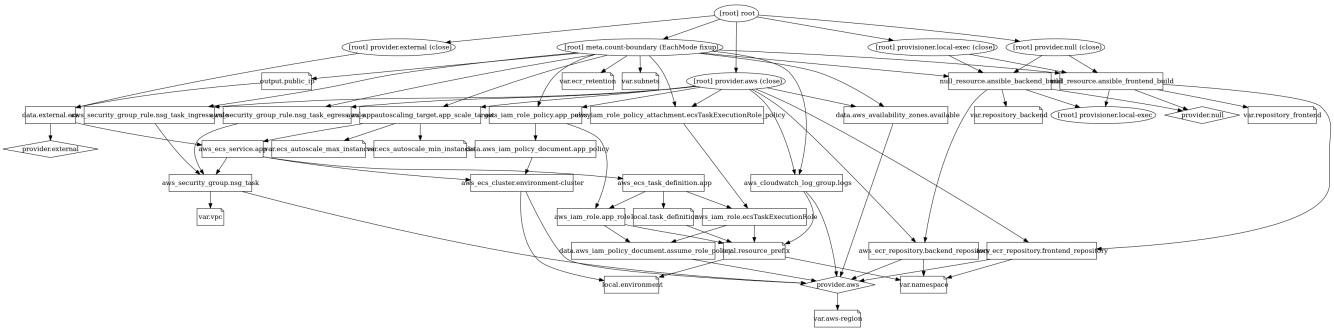

Figura 10. Representação de uma infraestrutura

Exemplo

```
resource "aws_subnet" "subnet-public-1" {
  vpc_id            = "${aws_vpc.vpc-devops.id}"
  cidr_block        = "${cidrsubnet(aws_vpc.vpc-devops.cidr_block, 8, 1)}"
  ipv6_cidr_block   = "${cidrsubnet(aws_vpc.vpc-devops.ipv6_cidr_block, 8, 1)}"
  availability_zone = "${data.aws_availability_zones.available.names[0]}"

  map_public_ip_on_launch = true

  tags {
    Name = "${cidrsubnet(aws_vpc.vpc-devops.cidr_block, 8, 2)} - subnet-public-1"
  }
}
```

Ansible

- Engine de automação;
- Idempotência
- Utiliza `yaml` para descrever os procedimentos;
- Módulos:
  - Infraestrutura na núvel;
  - Gerenciamento de Configuração;
  - Construção e implantação de aplicações..
- As máquinas alvo da execução são organizadas em um `inventory`
- Utiliza o protocolo `SSH` para se comunicar as máquinas do `inventory`;
- As execuções são agrupadas em `playbooks`;
- Cada execução é chamada de `play`.

Exemplo

```yaml
- name: Docker | Install or Upgrade
  block:
  - name: Docker | Install the package
    package:
      name: docker.io
      state: present
      force_apt_get: true
    become: true
  - name: Docker | Add current user to docker group
    user:
      name: "{{ ansible_user }}"
      groups: docker
      append: yes
    become: true
```

Terraform + Ansible ♡

DevOps

- Filosofia e práticas a serem implementas;
- A aplicação impactam duas áreas Desenvolvimento(Dev) e Operação(Ops);
- Sua aplicação envolve o negócio, pessoas e tecnologia;
- A adoção `DevOps` não resolverá problemas do projeto;
- A adoção `Devops` fará com que os problemas sejam expostos;
- `DevOps` não é **Jenkins**;
- `DevOps` não é um cargo;
- Seu maior valor será adquirido com equipes que seguem metodologias ágeis;

Problemas a serem resolvidos

- Longo processo pra implantação e entrega;
- Falta de automação de processos no CDS;
- Demora na identificação e resolução de problemas.

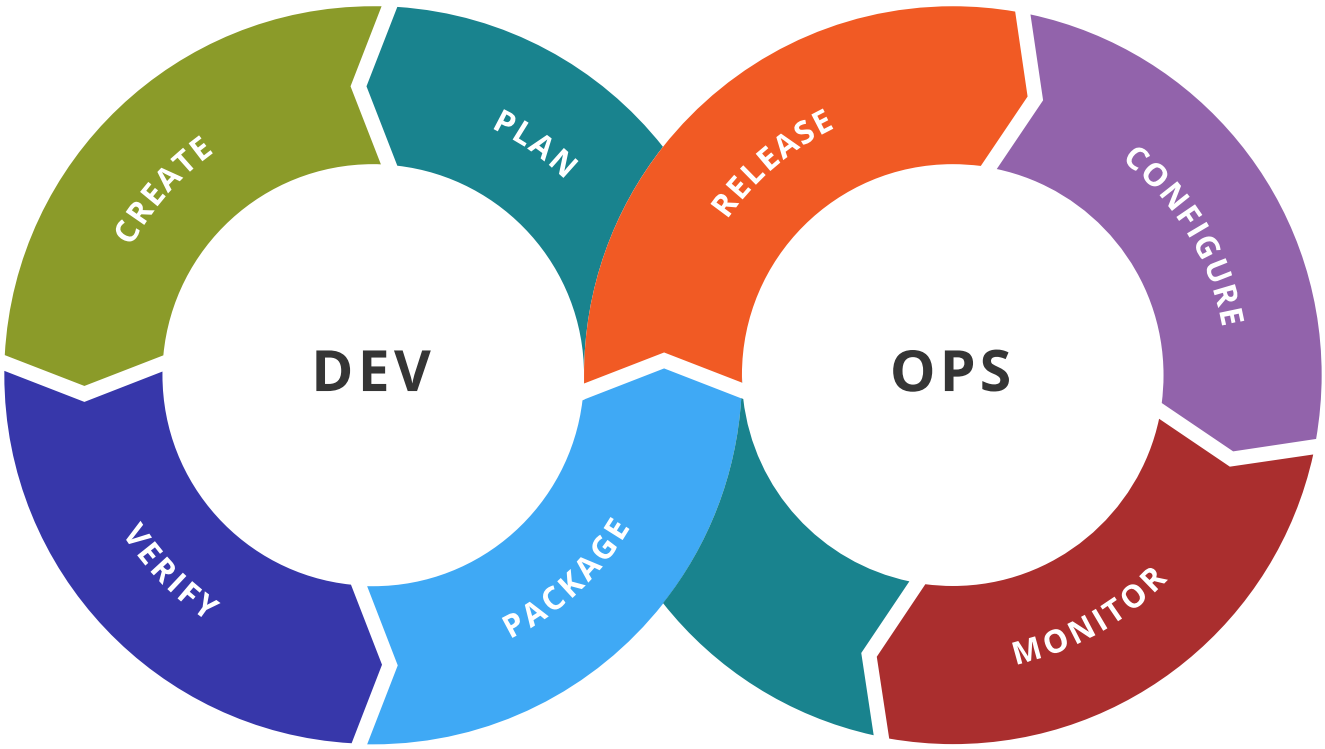

Figura 11. Representação do ciclo DevOps

DevOps em uma palavra

Automação

Automação leva a consistência

- Automação do processo de build;
- Automação de verificações de qualidade;
  - Testes unitários;
  - Testes funcionais;
  - Testes de integração;
  - Testes de regressão;
- Automação do processo de release;
- Automação de verificações de acessibilidade;
- Automação do processo de gerenciamento de configuração;
- Automação do processo de deploy.
- Automação do monitoramento;

Development x Operation

- Possuem objetivos diferentes, quase opostos;
  - **Desenvolvimento** - Liberação de **artefatos** de maneira **rápida** e **constante**;
  - **Operação** - Não são favoráveis a mudança, pois introduzem riscos alterando a estabilidade;

- O Desenvolvimento quebra a estabilidade da Operação;
- A operação impede a implantação de um artefato do desenvolvimento.
- O trabalho dos Devs e Ops são caixas-preta mutualmente;

Objetivos

- `Dev` e `Ops` compartilham objetivos e métricas;
- Envolver os dois times com o objetivo da implantação da aplicação;
- Alcançar o cumprimento de prazos e o **time to market**;
- Liberação de pequenas frações testadas exaustivamente;
- Minimizar problemas da entrega e implantação;
- Recuperação rápida em caso de falhas;

Práticas

- Integração Contínua;
- Qualidade Contínua;
- Entrega Contínua;
- Implantação Contínua;
- Monitoramento Contínuo.

Resultado

- `Dev` e `Ops` trabalham juntos pra alcançar a estabilidade e rapidez das entregas;
- Processo de desenvolvimento estável e reproduzível;
- **Qualidade**

DevSecOps

- Segue os mesmos princípios que o `DevOps`
- Acrescenta verificações de segurança ao processo;
- O Objetivo é encontrar vulnerabilidades em qualquer estágio do CDS;

ChatOps

- Conversation-driven
- Procedimentos são traduzidos em comandos;
- Os comandos são orquestrados por um robô;
- Comumente utilizado em ferramentas como `Slack`, `HipChat`, `IRC`;

- Utiliza um robô que monitorará as conversas(Hubot);
- Os comandos executarão:
  - Ações, que serão traduzidos para procedimentos;
  - Ações, que trarão informações.

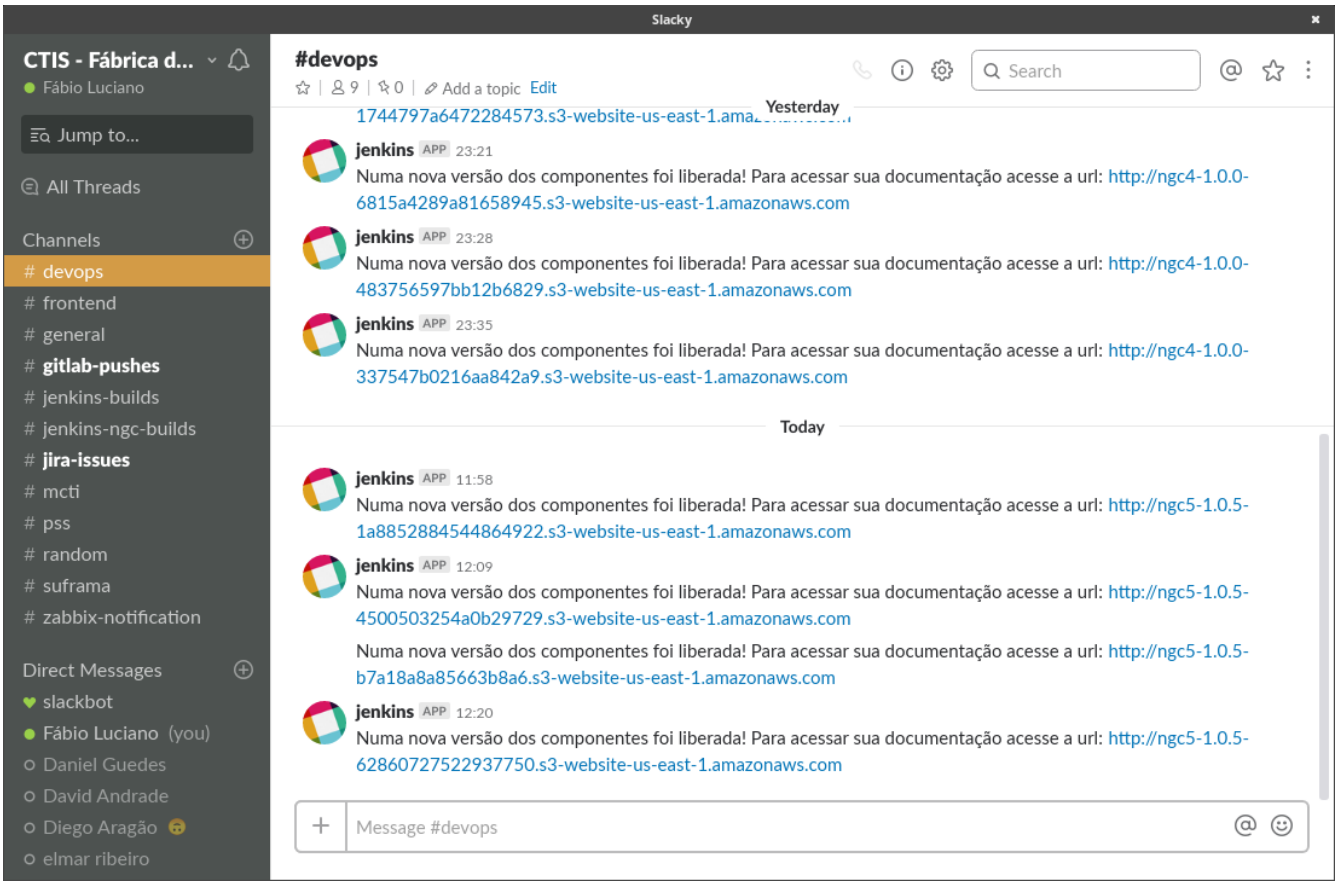

Figura 12. Representação da implementação do ChatOps

Benefícios Sociais

- Aumenta a colaboração/integração no desenvolvimento do produto;
- Aumenta o compatilhamento de informações;
- Aumenta a visibilidade e monitoramento de pontos críticos;
- Aumenta o conhecimento da equipe sobre o que está sendo desenvolvido;

Benefícios tecnológicos

- Aumenta a automação;
- Aumenta a velocidade de ações e execuções de instruções;
- Aumenta a segurança;
- Log automático de ações e diálogos;
- Redução de e-mails

Ajuda a...

- Compartilhar informações e problemas;
- Velocidade de feedbacks;
- Automação de tarefas;

Arquitetura com ferramentas Open Source

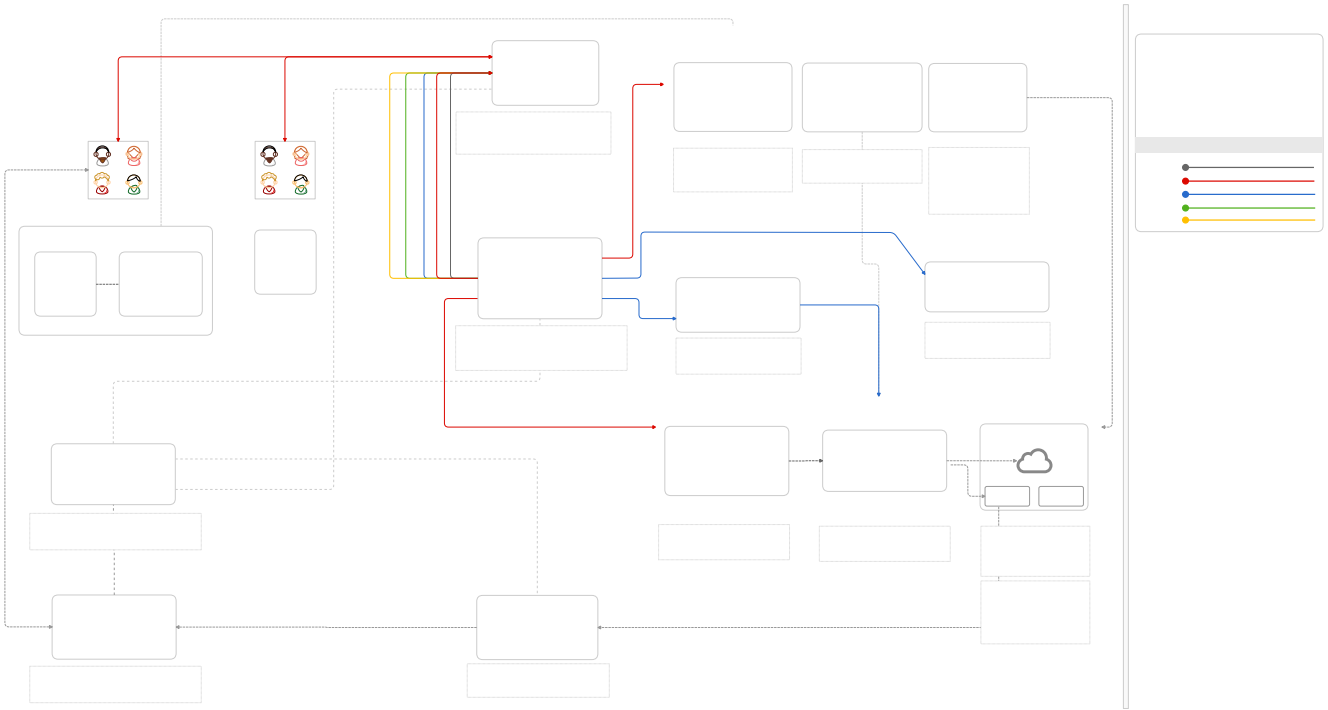

Figura 13. Representação de uma arquitetura DevOps

Arquitetura de Software

Requisitos não funcionais

12 Factor Apps

Microserviços

GraphQL

- É uma alternativa ao REST(Representational State Transfer);
- Especificação criada e utilizada pelo Facebook;
- Possui diversas implementações;
- Passa a responsabilidade para o frontend.

GraphQL vs REST

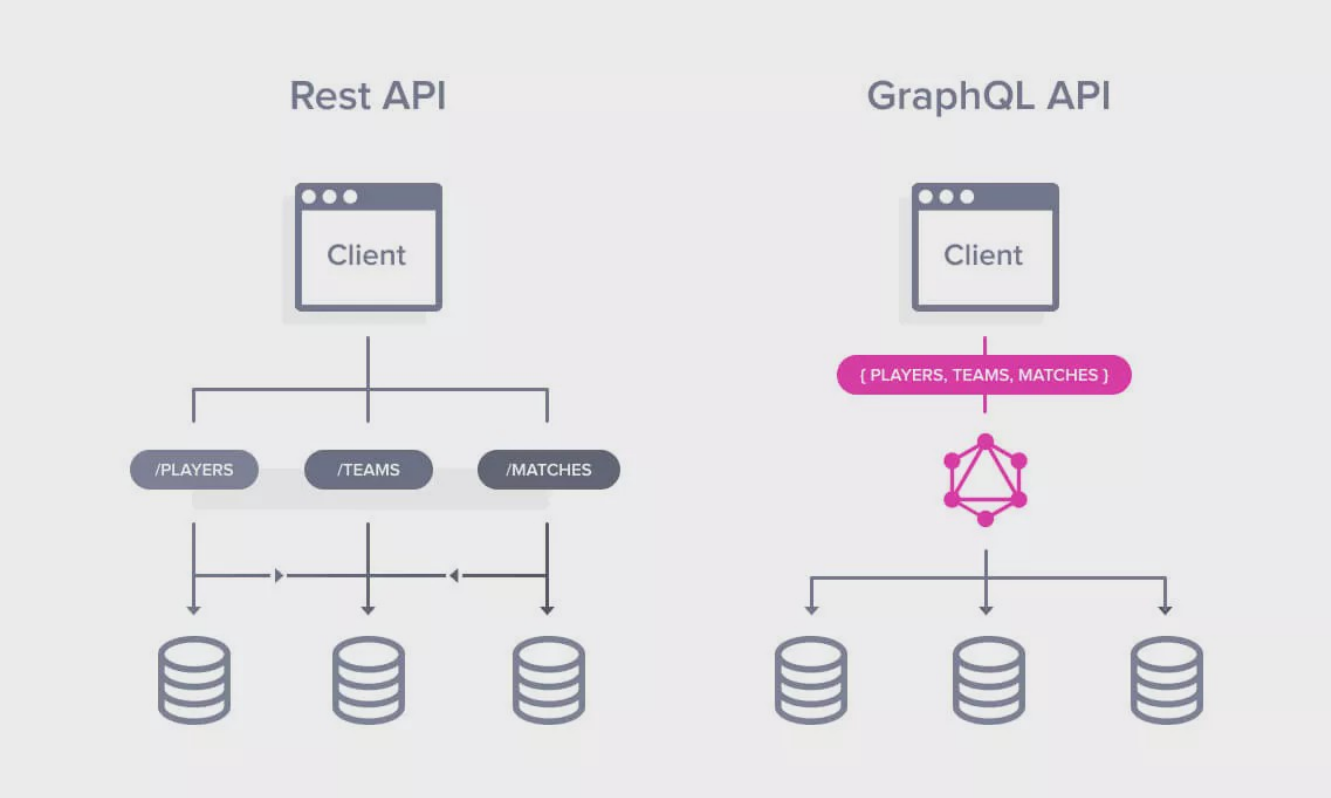

Figura 14. Representação das diferença entre as requisições do GraphQL e REST

Types

- Representa uma entidade do sistema;
- Geralmente mapeado a um Domínio/Modelo;

- Onde serão listados todos os campos de uma entidade consumível;
- É a `model` do HTTP.

Queries

- Representa uma consulta aos tipos existentes;
- Onde serão definidas as relações entre os `Types`;
- Cada `field` deve ser resolvido;
- Podem ser filtrados utilizando argumentos;
    - `Query` → `Field` → `Resolver`

Mutations

- Onde as modificações aos `Types` serão persistidas;
- Agrupa as operações `POST`, `PUT` e `DELETE` do HTTP(`REST`);
- Possui ligação direta a um `Type`;

Schema

- Agrupa `Types`, `Queries` e `Mutations` disponíveis;
- Funciona como um contrato de todas os elementos disponíveis;

Arquitetura de Soluções

Figura 15. Representação do ciclo DevOps

Cloud Computing

Vendors

- Amazon Web Service
- Google Cloud
- Microsoft Azure

Tipos de serviços

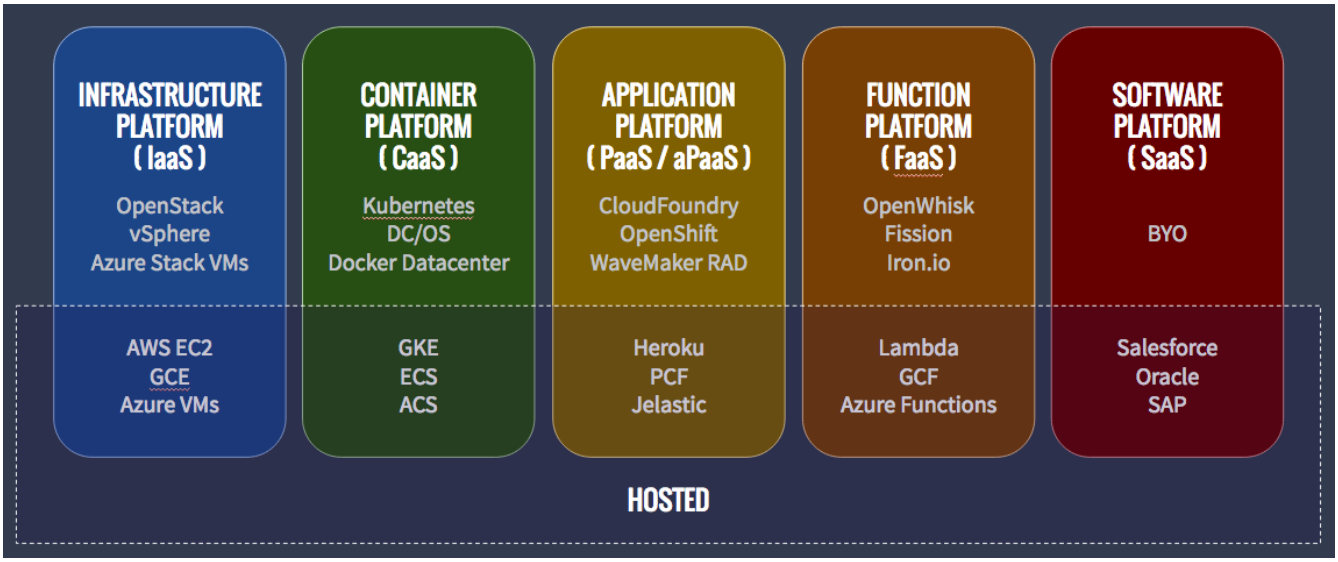

Figura 16. Representação do ciclo DevOps

Well Architected Framework

Segurança

Habilidade de proteger informações, sistemas e ativos enquanto entrega valor ao negócio, enquanto se avalia e mitiga os riscos.

Custo Efetivo

Habilidade de executar sistemas e entregar valor ao negócio enquanto mantêm o custo-benefício.

Resiliência

The ability of a system to recover from infrastructure or service disruptions, dynamically acquire computing resources to meet demand, and mitigate disruptions such as misconfigurations or transient network issues.

Excelência Operacional

The ability to run and monitor systems to deliver business value and to continually improve supporting processes and procedures.

Performático

The ability to use computing resources efficiently to meet system requirements, and to maintain that efficiency as demand changes and technologies evolve.