DEVOPS E AGILE CULTURE

DEVOPS

PEDRO IVO CORREIA DE ARAÚJO

LISTA DE FIGURAS

SUMÁRIO

1 DEVOPS

No mercado de trabalho atual, a quantidade de profissionais que atuam com desenvolvimento *back-end, front-end* e testes é muito maior que a quantidade de administradores de sistemas (*sysadmin*), analistas de redes, implantadores de infraestrutura, entre outros. Se fosse para exemplificar a proporcionalidade em uma imagem, o DevOps, em que o Dev representaria o lado de desenvolvimento (incluindo a área de qualidade), e Ops a operação e sustentação, seria algo assim:

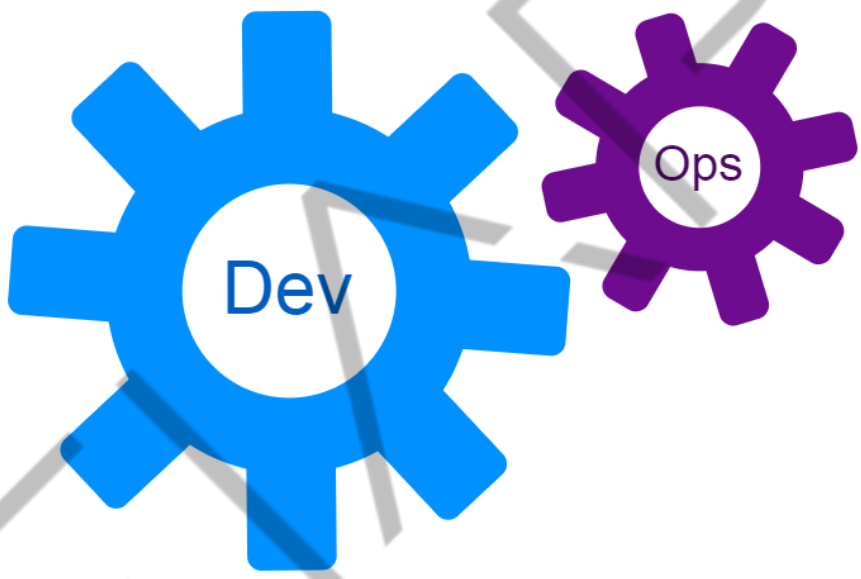

Figura 1.1 – Representação da proporcionalidade do DevOps
Fonte: Google Imagens (2019)

Antes de entrar no conceito de DevOps, para ficar mais clara essa proporcionalidade e identificar a principal importância de se adotar DevOps, vamos imaginar a seguinte situação: Uma empresa chamada SoftWeb, que possui aproximadamente 1.000 funcionários, desenvolve um produto, que é um software Web/Mobile, cujos concorrentes no mercado desenvolvem algo similar, forçando, assim, uma atualização constante de novas funcionalidades no sistema para manter a SoftWeb competitiva no mercado.

A SoftWeb disponibiliza cerca de 70% dos seus funcionários somente para desenvolver o seu produto, contando com uma média de 300 desenvolvedores e apenas 15 sysadmins. Na parte de gestão, temos um gerente geral responsável tanto

pela área de desenvolvimento quanto pela área de sysadmins, vamos chamá-lo de Alberto.

Alberto recebe da diretoria, basicamente, três objetivos principais para atingir as metas e os resultados da empresa no ano:

1. Aumentar em 35% as funcionalidades do produto que desenvolve, utilizando novas tecnologias, para torná-lo mais competitivo no mercado.
2. Atualizar 25% do seu produto com novas tecnologias para torná-lo mais atraente.
3. Aumentar de 90% para 99% a disponibilidade do produto, ficando, assim, mais tempo on-line, em comparação aos seus concorrentes.

Figura 1.2 – Ilustração didática do gestor passando metas
Fonte: Google Imagens (2019)

Com essas diretrizes recebidas, Alberto monta um plano de ação e convoca individualmente suas duas equipes para passar o direcionamento do ano. Na reunião com a equipe de Desenvolvimento, Alberto diz:

“Caros, vamos aumentar o número de funcionalidade do nosso sistema, para as novas demandas que surgirem, utilizem as novas tecnologias do mercado. Iremos

realizar treinamentos constantes na empresa para capacitá-los, também vamos migrar parte do nosso produto já existente para novas tecnologias, para, assim, ficarmos mais atualizados e atraentes no mercado! Vamos com tudo!”

Já com a equipe de sysadmins, o discurso de Alberto é o seguinte:

“Pessoal, estamos com uma meta muito agressiva com relação à disponibilidade do nosso software, temos que aumentar para 99%, isso significa que precisamos ser muito mais rigorosos com as mudanças que estão por vir, para impedir que novos *bugs* parem o nosso sistema, mudanças devem ser muito bem avaliadas e até mesmo impedidas, caso exista a possibilidade de causar indisponibilidade no ambiente de produção! Vamos com tudo!”.

Após o direcionamento do gerente geral, Alberto, as duas equipes voltam para o seu trabalho motivadas, no entanto, sem perceber, Alberto colocou as duas equipes em confronto! Como assim?

Basicamente, para a equipe de desenvolvimento contribuir com as metas, necessita desenvolver novas funcionalidades e atualizar parte do legado para novas tecnologias, isso ocasionará aumento no número de mudanças e entregas no ambiente de produção, que cai em contradição com os objetivos da equipe de sysadmins, cujos membros estarão muito mais receosos em referência às mudanças proporcionadas pelos desenvolvedores, em que o cenário mais fácil para se manter a estabilidade do sistema seria um baixo número de mudanças e entregas em produção. Para agravar ainda mais o cenário, ambas as equipes são tratadas individualmente e trabalham separadamente, cada um com seus objetivos e metas para serem alcançados.

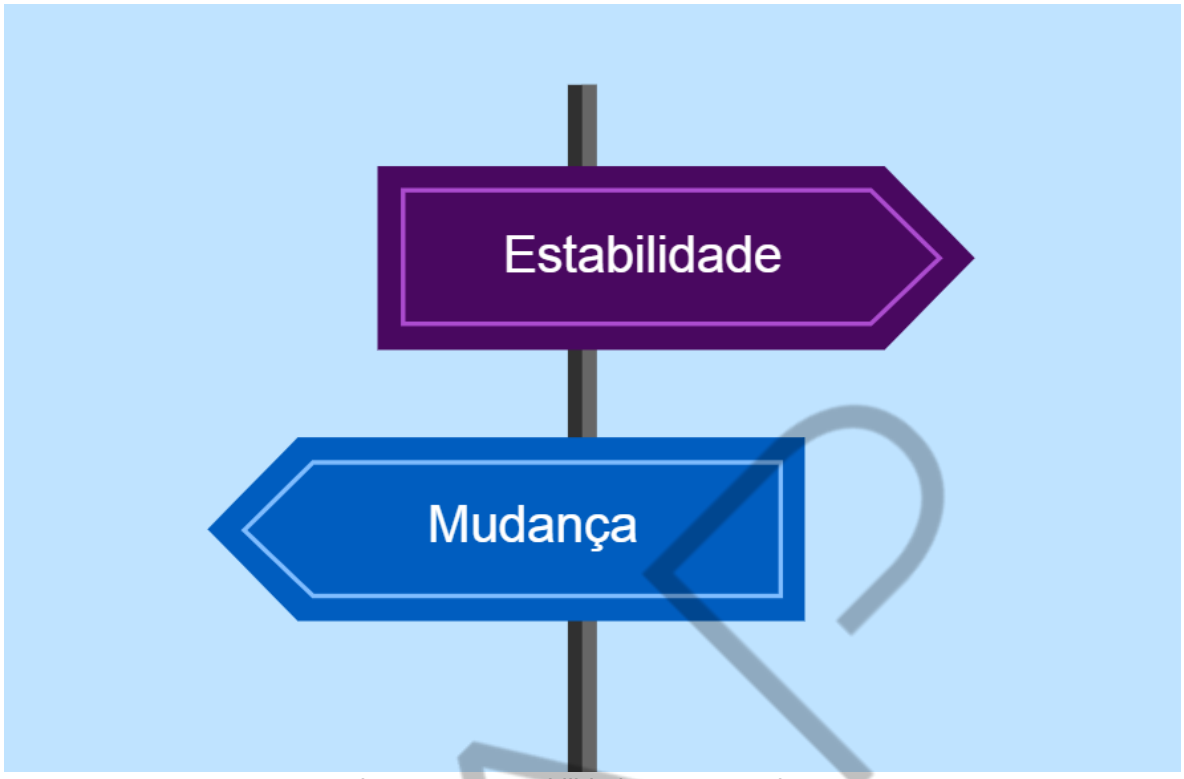

Figura 1.3 – Estabilidade versus mudança
Fonte: Google Imagens (2019)

Apesar de a situação descrita ser fictícia, é o que acontece muitas vezes nas empresas, os times de desenvolvimento e operações não trabalham em conjunto, possuem metas e objetivos distintos, contribuindo para aumentar o “vale” de distâncias entre as duas áreas. Para solucionar a situação e promover mudanças com estabilidade, além de construir uma ponte que unifique os lados para remover esse “vale” na organização, vamos aprender sobre **DevOps**!

1.1 O que é devops?

Antes de qualquer coisa, DevOps é uma cultura que utiliza práticas e ferramentas para aumentar a capacidade de uma organização de desenvolver e entregar softwares, serviços, aplicativos e demais produtos de tecnologia com alta velocidade, porém, sem pôr em risco a estabilidade.

Quando a organização adota a cultura DevOps, o ritmo de entrega dos produtos é mais rápido do que o das empresas que usam processos tradicionais de desenvolvimento de software e gerenciamento de infraestrutura.

1.2 Funcionamento do DevOps

Com a adoção da cultura DevOps, as equipes de desenvolvimento e operações que exemplificamos anteriormente não são mais tratadas de forma separada, essas equipes são combinadas em uma só, para, assim, ambas compartilharem os conhecimentos e entenderem melhor o dia a dia do desenvolvimento e operação. Os sysadmins são envolvidos desde o início do desenvolvimento do software, e os desenvolvedores acompanham o software até o fim, principalmente na parte das entregas e funcionamento do ambiente de produção.

Uma dúvida comum ao se adotar a cultura DevOps é: Como desenvolvedor, precisarei ter os mesmos conhecimentos do sysadmins, e como sysadmin precisarei ter os mesmos conhecimentos dos desenvolvedores? A resposta é não; como desenvolvedor você não precisa conhecer de forma profunda todas as habilidades do sysadmin, mas deve saber o essencial para acompanhar e contribuir com a implantação/operação do software, e o sysadmin deve conhecer o essencial para acompanhar e contribuir com o desenvolvimento do software.

Esse movimento acaba, de forma indireta, estimulando um ambiente multidisciplinar, no qual os desenvolvedores e sysadmins compartilham os conhecimentos, experiências e dores de cada área e, juntos, usam práticas para automatizar processos que sempre foram manuais ou lentos, empregando tecnologias e ferramentas que os ajudam a desenvolver e operar aplicativos de modo rápido, estável e seguro. Com o aumento de conhecimento operacional por parte dos desenvolvedores, a autonomia aumenta e tarefas simples, que normalmente exigiriam a ajuda de um sysadmin ou demais equipes de infraestrutura, começam a ser realizadas de forma independente, e o sysadmin passa a conhecer mais do software para conseguir atuar de forma melhor com ele, diminuindo a dependência do desenvolvedor para entender certos fluxos ou comportamentos que antes só a equipe de desenvolvimento conheceria.

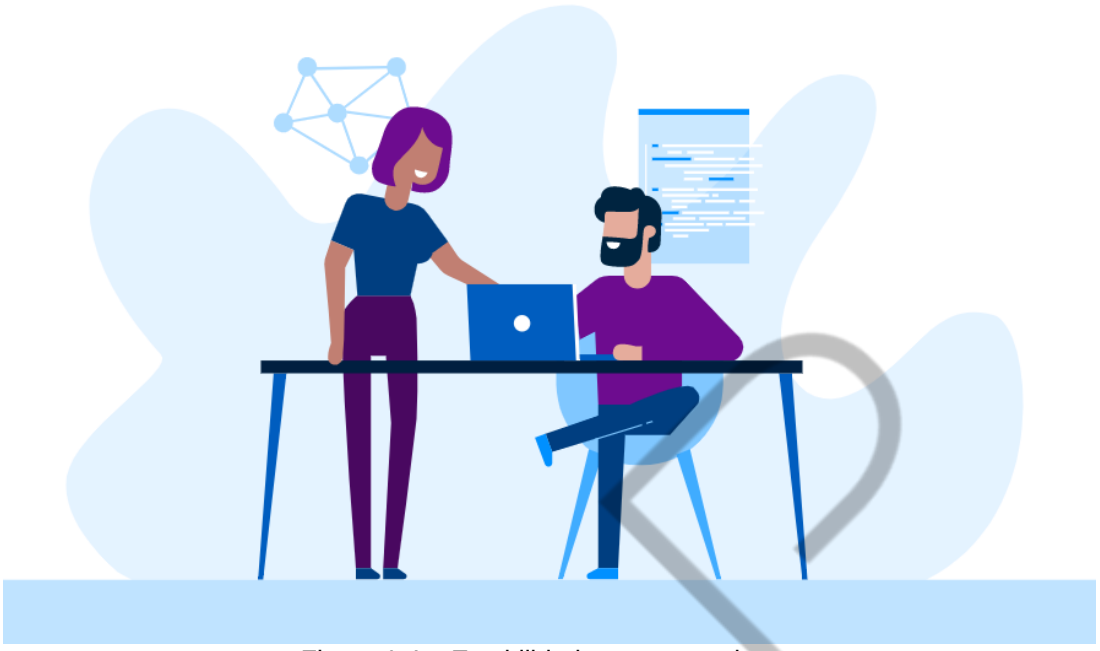

Figura 1.4 – Estabilidade versus mudança
Fonte: Google Imagens (2019)

1.3 Benefícios do DevOps

Vamos abordar agora os principais benefícios ao se adotar uma cultura DevOps:

1.4 Aumento da velocidade de entrega

DevOps proporciona, por meio de ferramentas, a automação de processos manuais e lentos, contribuindo, assim, para o aumento da frequência e do número de entregas do seu produto. Quanto mais rápido você conseguir entregar, mais rápido identificará possíveis problemas e poderá corrigir os erros com maior agilidade. Além de identificar necessidades do mercado e conseguir entregar de forma mais rápida, você poderá criar uma vantagem competitiva.

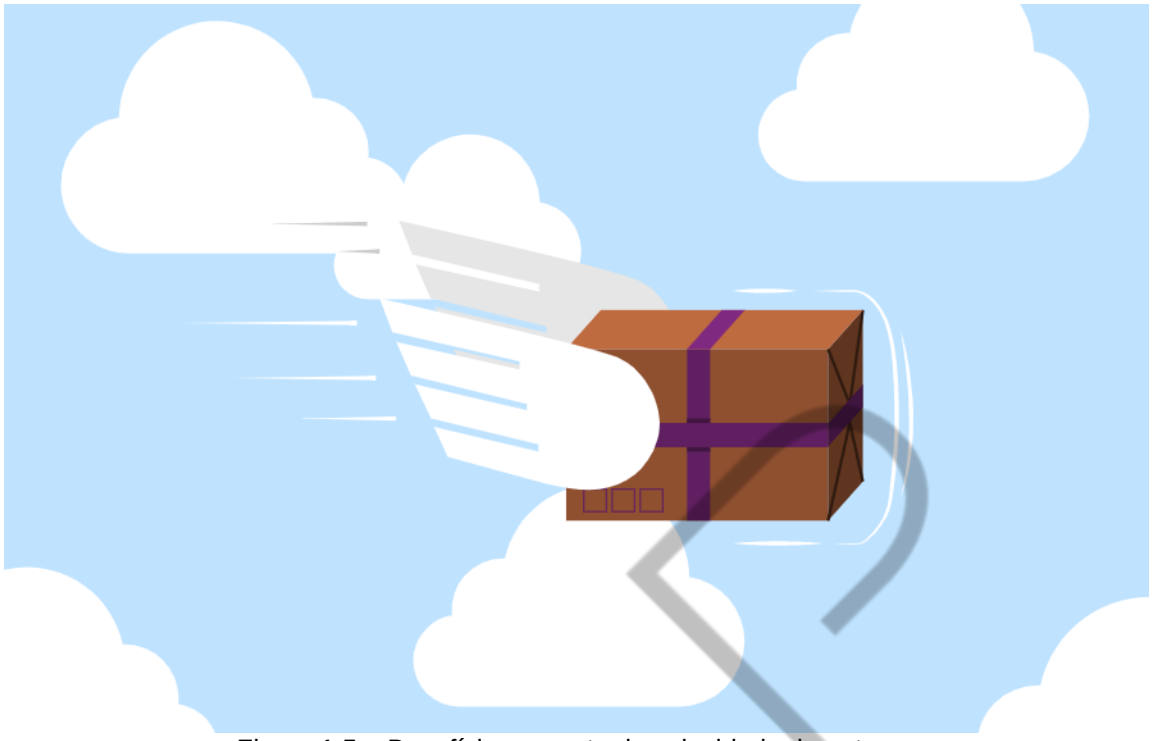

Figura 1.5 – Benefício aumento da velocidade de entrega
Fonte: Google Imagens (2019)

1.5 Escalabilidade

Com a automatização da infraestrutura, DevOps proporciona a possibilidade de gerenciar sua infraestrutura como código, diminuindo a interferência manual e, consequentemente, o risco. Você passa a escalar seu código de infraestrutura em diversos ambientes, pois aquilo que é igual para todos é replicado, e implantando de forma individual e automática o que é específico. Com processos automáticos, é possível identificar a necessidade de escalar sua infraestrutura de acordo com a demanda, por exemplo: seu software está recebendo mais requisições do que o esperado para a infraestrutura provisionada; identificado o cenário, antes que aconteça o problema, a automação pode, a partir de um alarme, expandir a infraestrutura para atender às requisições.

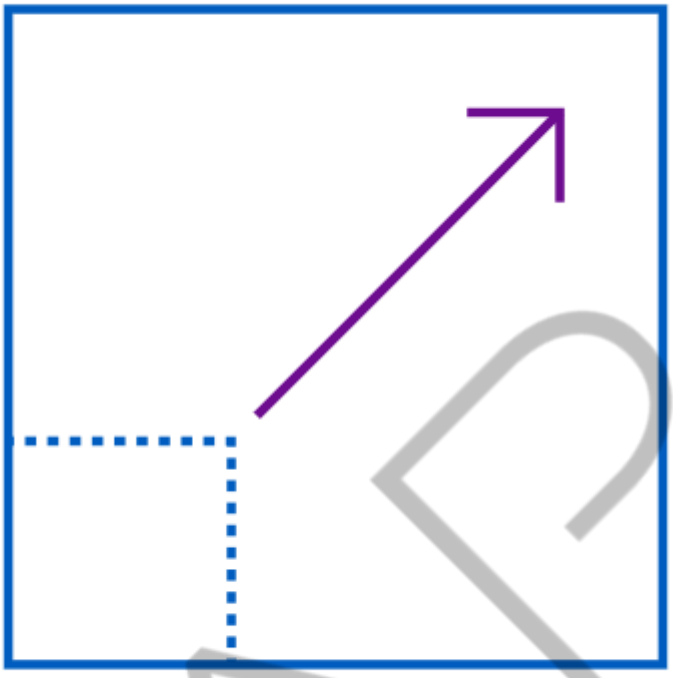

Figura 1.6 – Benefício escalabilidade
Fonte: Google Imagens (2019)

1.6 Velocidade

As equipes que possuem a cultura DevOps têm maior independência, assumindo a responsabilidade ponta a ponta dos produtos e serviços para, então, realizar as entregas e melhorias de forma mais rápida, contribuindo assim para o atingimento de resultados. Com as equipes juntas, ambas estão olhando para o mesmo objetivo, também não é necessário demandar algo de infraestrutura para fora do time, o que poderia levar mais tempo ou espera por priorização.

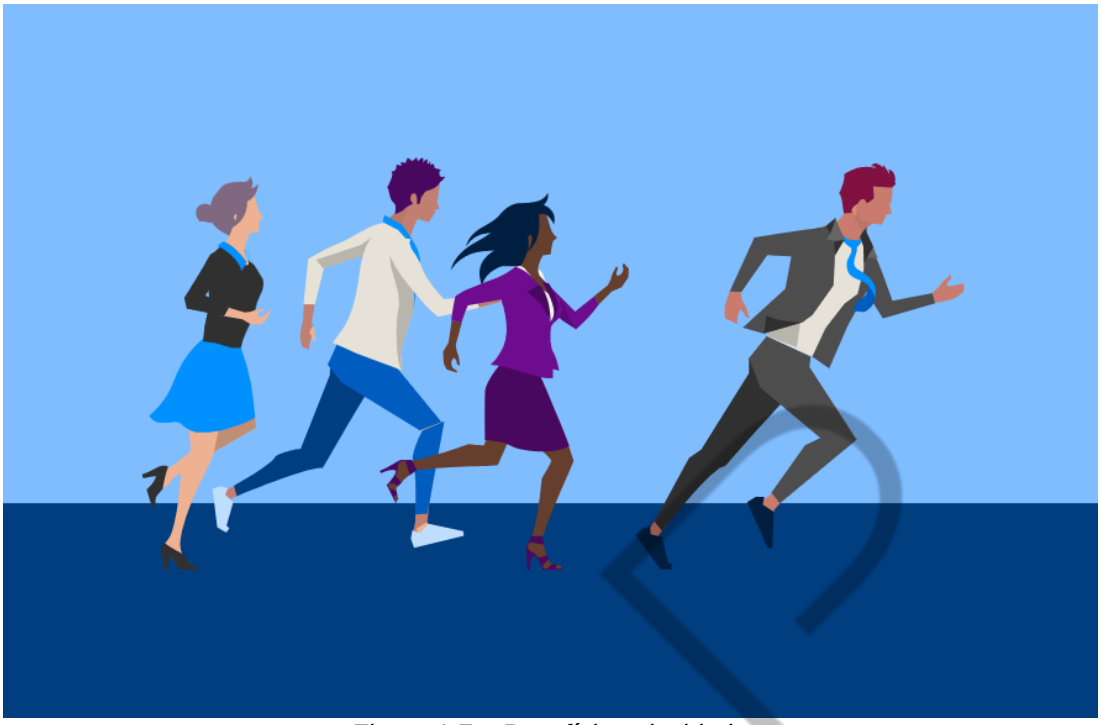

Figura 1.7 – Benefício velocidade
Fonte: Google Imagens (2019)

1.7 Colaboração contínua

A junção das equipes as torna mais eficientes, promovendo a cultura da responsabilidade ponta a ponta e do sentimento de "dono" do que é feito. As equipes de desenvolvimento e sysadmin colaboram juntas, compartilham muitas responsabilidades e acordam seus fluxos de trabalho, como consequência, acontece a redução de processos ineficazes e a economia de tempo.

Figura 1.8 – Benefício colaboração contínua
Fonte: Google Imagens (2019)

1.8 Confiabilidade

DevOps promove a garantia da qualidade das atualizações de software e alterações de infraestrutura por meio de processos automatizados de testes em diversos níveis, para, assim, aumentar a confiança das entregas e contribuir com a sua velocidade. Os testes são parte fundamental do processo, e são programados para serem executados no decorrer de todo o ciclo.

Figura 1.9 – Benefício confiabilidade
Fonte: Google Imagens (2019)

1.9 Segurança

A adoção da cultura DevOps aumenta a segurança por meio de políticas de segurança automáticas, como controles de acesso entre aplicações, permissionamento, autorização e técnicas de gerenciamento de configuração.

Figura 1.10 – Benefício segurança
Fonte: Google Imagens (2019)

1.10 Práticas DevOps

Para promover a cultura DevOps, existem algumas práticas principais que ajudam as organizações a inovarem mais rapidamente por meio da automação e da simplificação dos processos de desenvolvimento de software e gerenciamento de infraestrutura. A maioria dessas práticas é realizada por meio de certas ferramentas.

Uma prática DevOps essencial é a execução de atualizações muito frequentes, porém pequenas e está muito ligada à metodologia de desenvolvimento ágil, como, por exemplo, SCRUM ou KANBAN.

Outra prática, essa relacionada à arquitetura, é a adoção de microsserviços, para tornar seus aplicativos mais flexíveis e permitir mudanças de forma mais rápida. A arquitetura de microsserviços desacopla sistemas grandes e complexos e os transforma em sistemas menores e independentes. Os sistemas são divididos em vários serviços individuais, e cada um deles abrange uma parte ou função única do negócio, além de ser operado independentemente dos serviços de mesmo nível e do sistema como um todo. A arquitetura de microsserviços reduz a sobrecarga gerada pela coordenação da atualização de grandes sistemas e, quando cada serviço é

combinado com equipes pequenas e ágeis que assumem a responsabilidade ponta a ponta sobre cada serviço, as empresas conseguem trabalhar mais rapidamente.

Com a combinação de arquitetura de microsserviços e uma maior frequência de entregas, consequentemente, teremos um número significativamente maior de implantações, que podem apresentar desafios, e com as práticas de DevOps, como a integração e entrega contínua entre desenvolvimento e operação, sempre executando os testes necessários, os problemas são minimizados. As práticas de automação de infraestrutura, como a infraestrutura como código e o gerenciamento de configuração, contribuem para que os recursos de infraestrutura trabalhem de forma elástica e disponível para alterações constantemente. Práticas de monitoração e indexação de log contribuem para a identificação e prevenção de problemas, além de auxiliar no acompanhamento do desempenho de softwares e da infraestrutura, para que possam reagir rapidamente quando ocorrer problemas.

A seguir, vamos listar as melhores práticas da cultura **DevOps**:

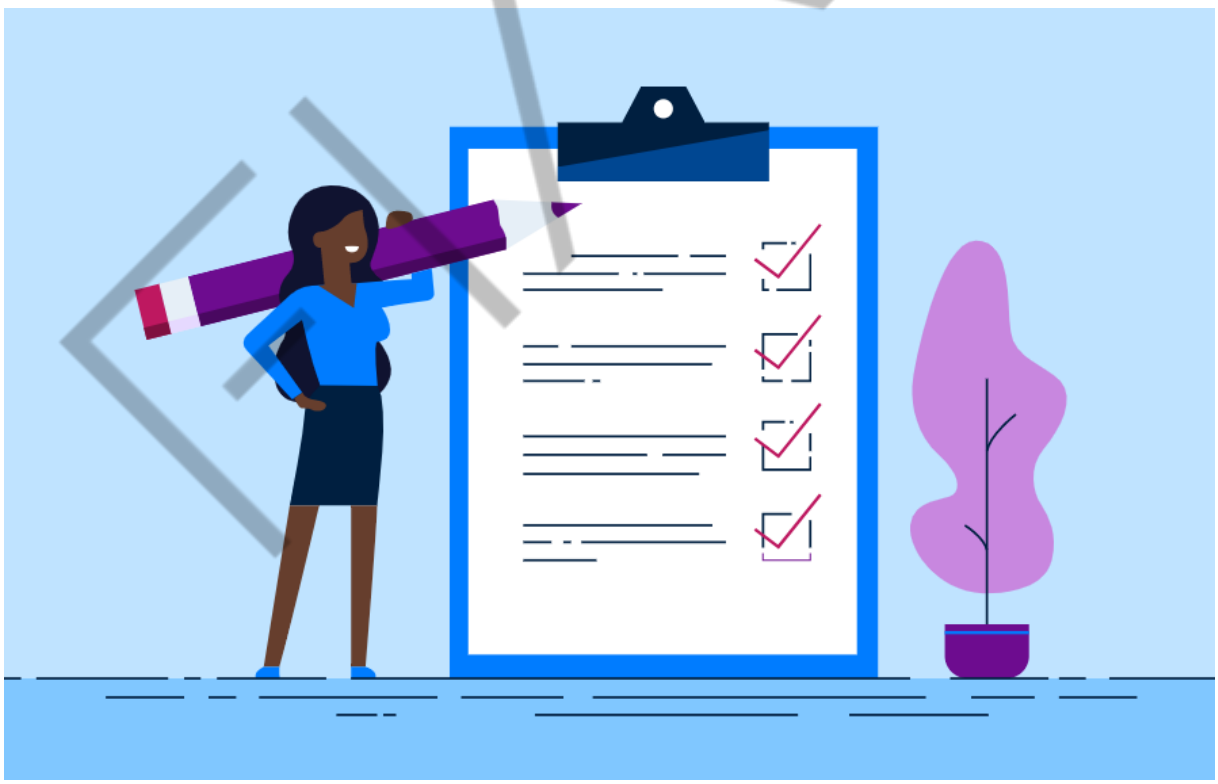

Figura 1.11 – Imagem ilustrativa lista de práticas DevOps
Fonte: Google Imagens (2019)

1.11 Infraestrutura como código

A infraestrutura como código é uma prática que utiliza técnicas de desenvolvimento de código e que permite controle de versão e integração contínua da infraestrutura, por meio de API, para que os desenvolvedores e sysadmins trabalhem com a infraestrutura de modo programático, em vez de instalar e configurar manualmente a infraestrutura. Isso permite que as equipes dentro de uma empresa operem em uma velocidade maior, uma vez que o código da infraestrutura pode ser reaproveitado, e, quando atualizado, replicado para todos os ambientes que utilizam esse trecho de código de infraestrutura.

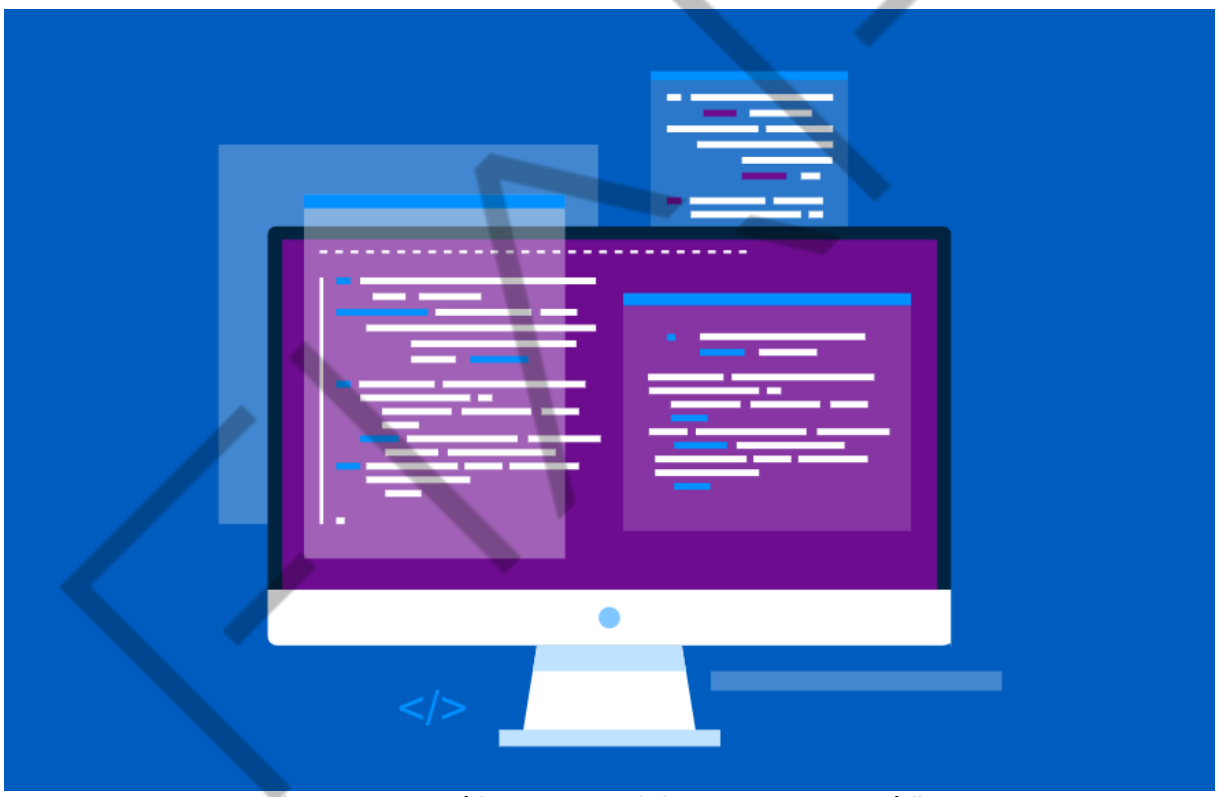

Figura 1.12 – Práticas DevOps infraestrutura como código
Fonte: Google Imagens (2019)

1.12 Arquitetura de Microsserviços

A arquitetura de microsserviços representa um conjunto de pequenos serviços que se interligam para construir um sistema. Cada serviço possui um contexto único de negócio, é executado de forma individual e independente e se comunica com outros serviços por meio de uma interface leve, na maioria dos casos baseada em HTTP.

Você pode usar diferentes linguagens de programação para construir os microsserviços e implantá-los independentemente, desde que consigam se expor e se comunicar na interface definida entre eles.

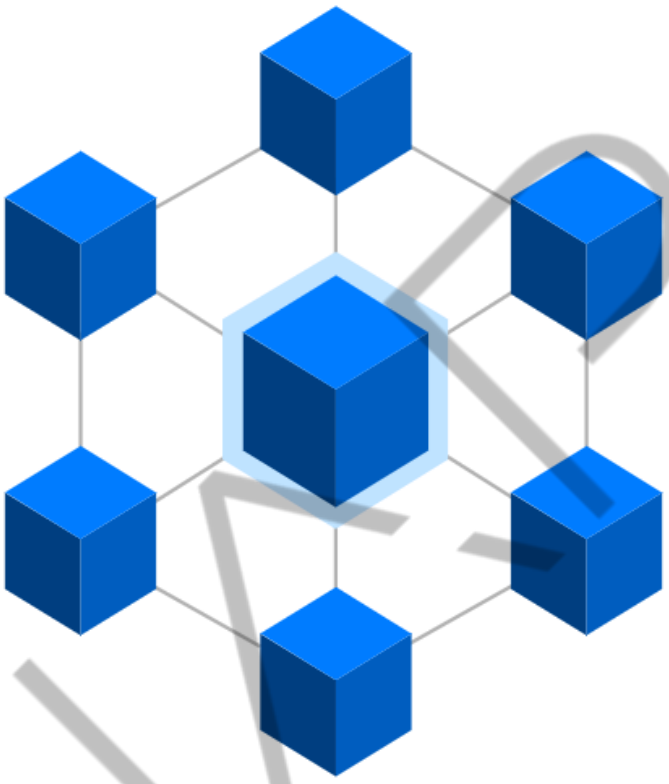

Figura 1.13 – Práticas DevOps arquitetura de microsserviços
Fonte: Google Imagens (2019)

1.13 Integração Contínua

A integração contínua é uma prática de desenvolvimento que permite a execução dos testes sempre que as alterações de código são enviadas para o repositório central. Os principais objetivos da integração contínua são encontrar e apontar os erros mais rapidamente a cada alteração, consequentemente, melhorar a qualidade do software e reduzir o tempo necessário para validação.

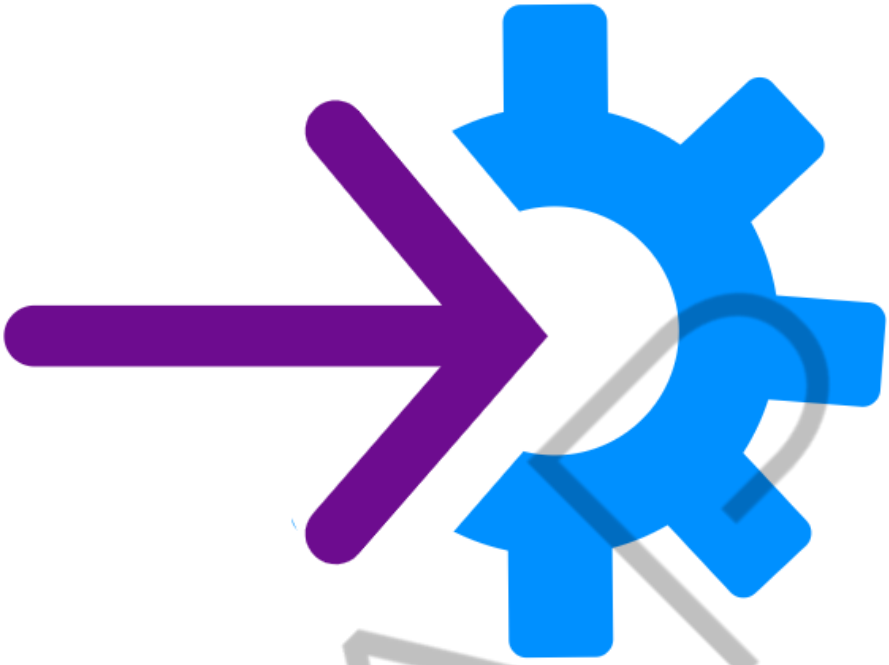

Figura 1.14 – Práticas devops integração contínua
Fonte: Google Imagens (2019)

1.14 Entrega Contínua

A entrega contínua é uma prática que permite ao desenvolvedor, ao realizar as alterações de códigos, utilizar a integração contínua para realização dos testes necessários e preparar automaticamente as modificações para uma entrega em produção. Quando a integração contínua é implementada adequadamente, os times terão um pacote de entrega confiável pronto para ser implantado a cada alteração ou conjunto de alterações enviadas para o repositório central.

Figura 1.15 – Práticas devops entrega contínua
Fonte: Google Imagens (2019)

1.15 Monitoração, Alarme, Log e Indexação

Realizar logs de informações das aplicações e infraestrutura é essencial para realizar o monitoramento e gerar alarmes. Ao capturar, indexar e analisar os logs gerados pelos aplicativos e pela infraestrutura, é possível entender como as alterações ou atualizações estão afetando o ambiente e seus usuários, o que proporciona mais facilidade na rastreabilidade, fornecendo maior esclarecimento sobre as causas raiz dos problemas. Com os logs indexados, é possível criar *dashboards* de acompanhamento real time e programar alarmes de acordo com determinada situação do ambiente.

Figura 1.16 – Práticas monitoração, alarme, log e indexação
Fonte: Google Imagens (2019)

1.16 Comunicação e Colaboração

O aumento da comunicação, colaboração e compartilhamento de experiência é um dos principais aspectos culturais do DevOps. O uso das práticas e ferramentas contribui para as equipes definirem normas culturais sólidas com relação ao compartilhamento de informações e processos de trabalho. Com a unificação das equipes, todos passam a trabalhar juntos, seguindo um objetivo comum.

Figura 1.17 – Práticas devops comunicação e colaboração
Fonte: Google Imagens (2019)

1.17 Estágios e Ferramentas DevOps

Apresentaremos, a seguir, os estágios de um processo de **DevOps** e as ferramentas utilizadas em cada um deles.

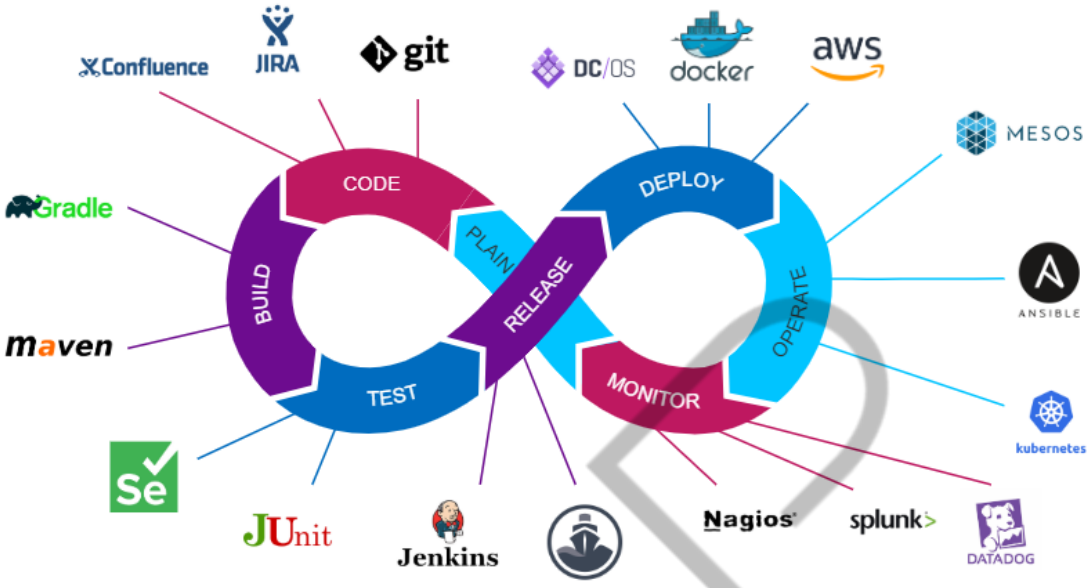

Figura 1.18 – Ilustração estágios e ferramentas devops
Fonte: Google Imagens (2019)

1.18 Planejamento (*Plan*)

Na fase de planejamento, quando os desenvolvedores e sysadmins estão interagindo para estimar e fatiar as atividades necessárias para a entrega, é fundamental a utilização das práticas de agilidade para melhor organização e fluxo das atividades.

1.19 Desenvolvimento ou Codificação (*Code*)

Na etapa de desenvolvimento, com as atividades definidas, os times começam a codificação do software e a codificação da infraestrutura como código. Nessa fase, utilizamos o **Git**, sistema de controle de versões distribuído, capaz de registrar o histórico de edições de qualquer tipo de arquivo, facilitando que um time trabalhe no mesmo arquivo de código ao mesmo tempo. Para documentar nosso sistema de forma colaborativa, temos o **Confluence**, e utilizamos o **Jira** para organizar e acompanhar as atividades do time, garantindo o gerenciamento de todo o ciclo de desenvolvimento

em um único lugar, permitindo o link entre atividades, repositório **Git** e documentação do **Confluence**.

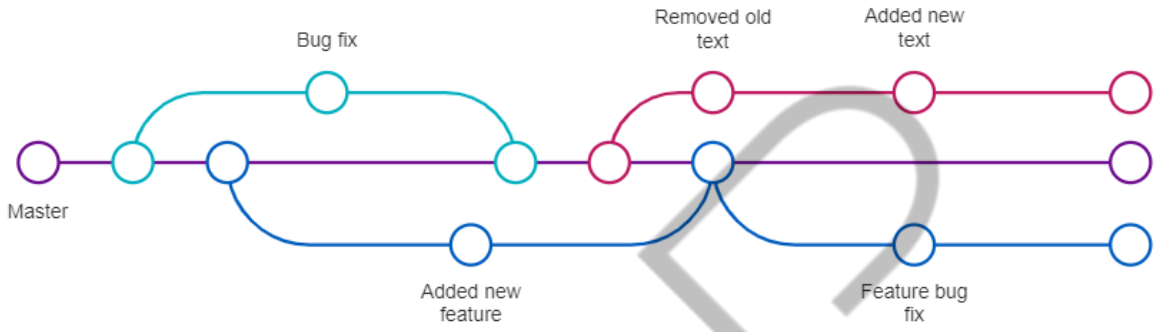

Figura 1.19 – Ilustração fluxo git
Fonte: Google Imagens (2019)

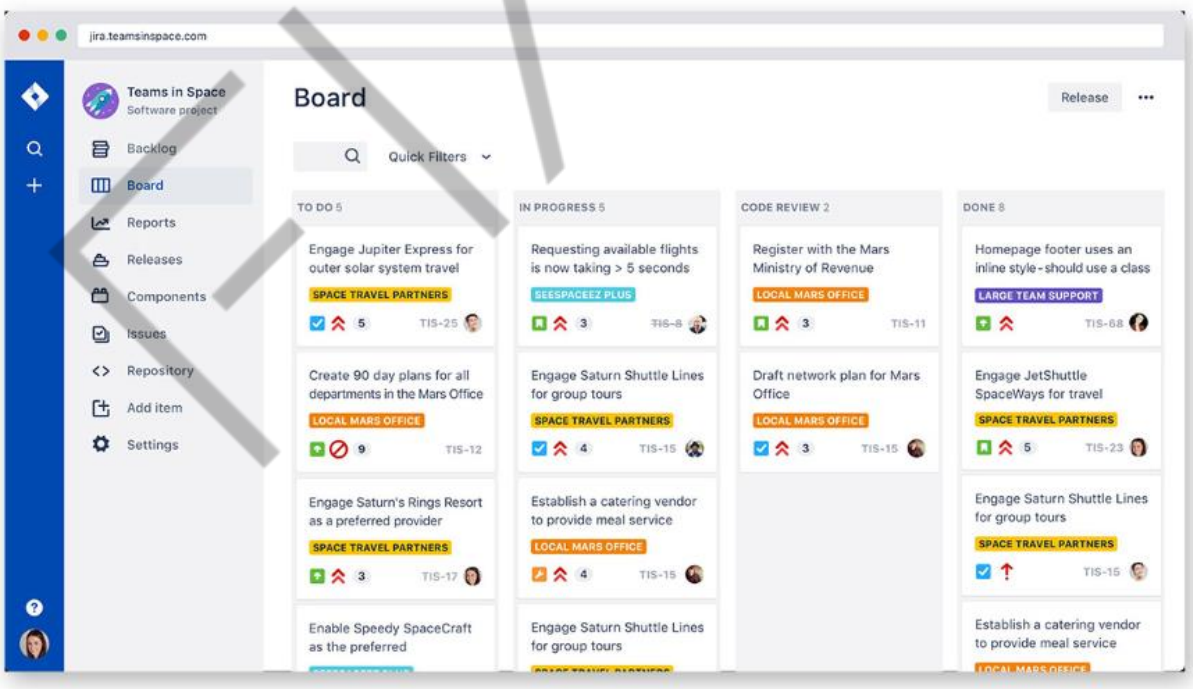

Figura 1.20 – Tela exemplo do jira software
Fonte: Google Imagens (2019)

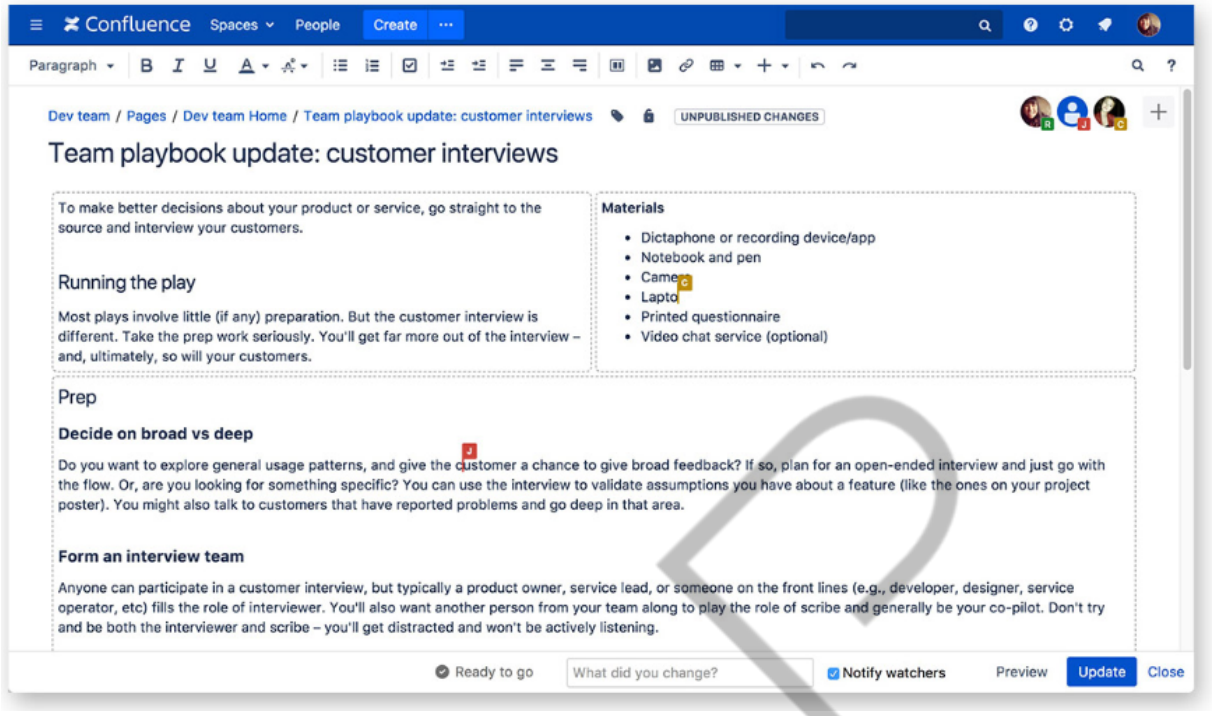

Figura 1.21 – Tela exemplo do jira software
Fonte: Google Imagens (2019)

1.20 Construção (*Build*)

Com o código desenvolvido, é necessário realizar o processo de construção também conhecido como build, no qual o código e demais dependências do software e da infraestrutura são baixados do repositório central, compilados e fechados em uma versão, que será disponibilizada para os testes. Também é comum, nessa fase, alguns testes básicos já serem realizados e, se não forem aprovados, o processo de *build* é cancelado. Para essa fase, podemos utilizar ferramentas como Apache Maven, que realiza a automação da compilação do código e gerenciamento das dependências por meio de XML, ou o Gradle, que se baseia nos conceitos do Apache Maven, porém, introduz uma linguagem de domínio específico, baseada em Groovy em vez de XML, permitindo maiores funções programaticamente.

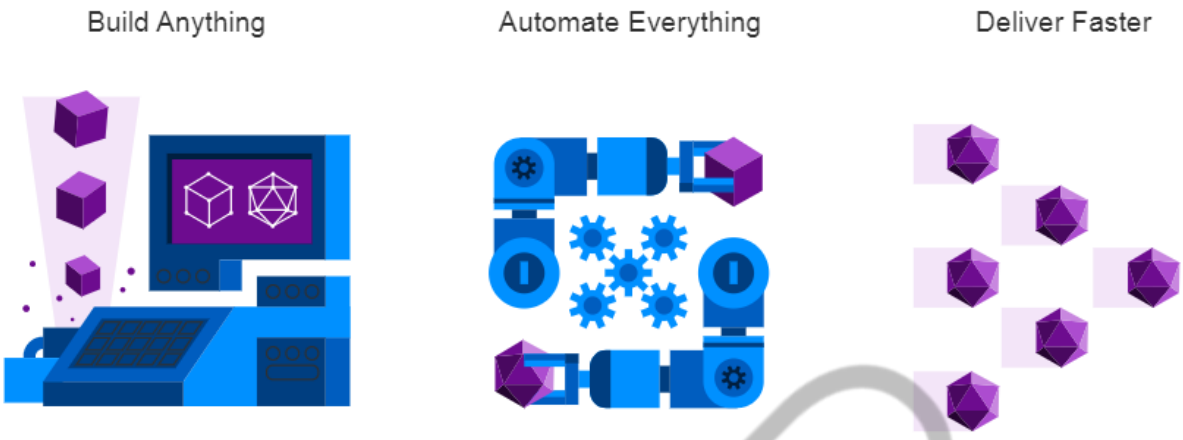

Figura 1.22 – Ilustração build, automação e delivery
Fonte: Google Imagens (2019)

1.21 Teste (*Test*)

Com a aplicação do **DevOps** na fase de testes, além dos testes no software, que vão desde testes unitários, a testes de fumaça (*smoke test*) e integração, regressão e ponta a ponta, também são realizados os testes de infraestrutura. Para o software, é muito comum a utilização do JUnit para testes unitários de código, e o Selenium para testes que envolvam telas ou fluxos ponta a ponta. Para infraestrutura, temos o TestInfra para realizar a validação.

Figura 1.23 – Logos do test infra, junit e selenium
Fonte: Google Imagens (2019)

1.22 Lançamento / Entrega (Release)

Após o desenvolvimento, testes e empacotamento, é o momento de fazer o lançamento da versão. Com a cultura DevOps, esse processo é automatizado por meio de ferramentas de pipeline de entrega, que suportam integração e entrega contínua, como Jenkins e CodeShip. No *pipeline* de entrega, é possível configurar diversas etapas da entrega, que podem ir desde testes simples até entregas em diversos níveis de ambientes (homologação, produção, pós-produção), além de permitir um passo de aprovação para seguir com a entrega.

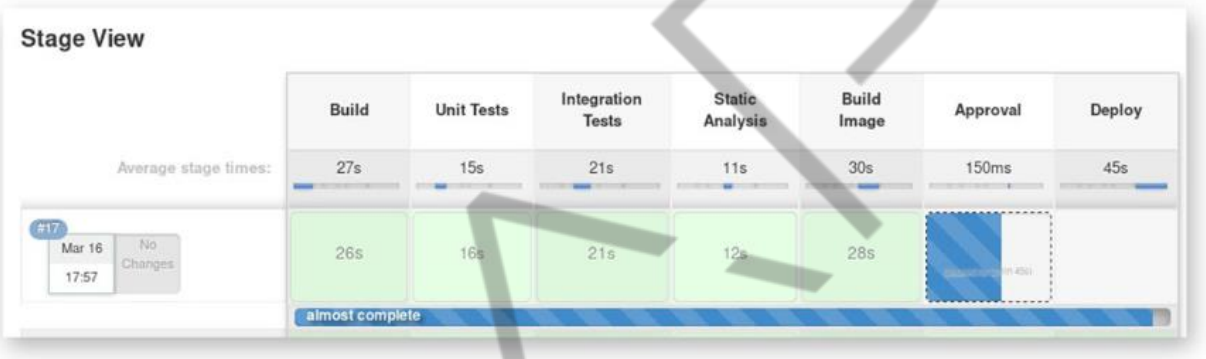

Figura 1.24 – Exemplo de pipeline de entrega do jenkins
Fonte: Google Imagens (2019)

1.23 Implantação (Deploy)

A fase de implantação está muito ligada à fase do Lançamento (Release). Na maioria das vezes, a implantação é acionada pelo pipeline de entrega, e nesse momento, existe a instalação de forma automatizada do nosso software e infraestrutura. Diferentemente da instalação manual, com DevOps o processo é automatizado, e por meio de ferramentas como Docker, que fornece uma camada adicional de abstração e automação de virtualização em nível de sistema operacional, ele encapsula o seu software em um contêiner com tudo o que é necessário para ser executado. Uma dúvida comum que existe, quando falamos sobre contêineres, é qual a diferença do Docker comparado a uma máquina virtual (VM) − basicamente, o Docker não necessita de um Sistema Operacional (OS) dentro da sua virtualização para funcionar, ele utiliza o sistema operacional do próprio host para trabalhar. A figura “Diferença entre virtualização e contêiner” exemplifica a diferença:

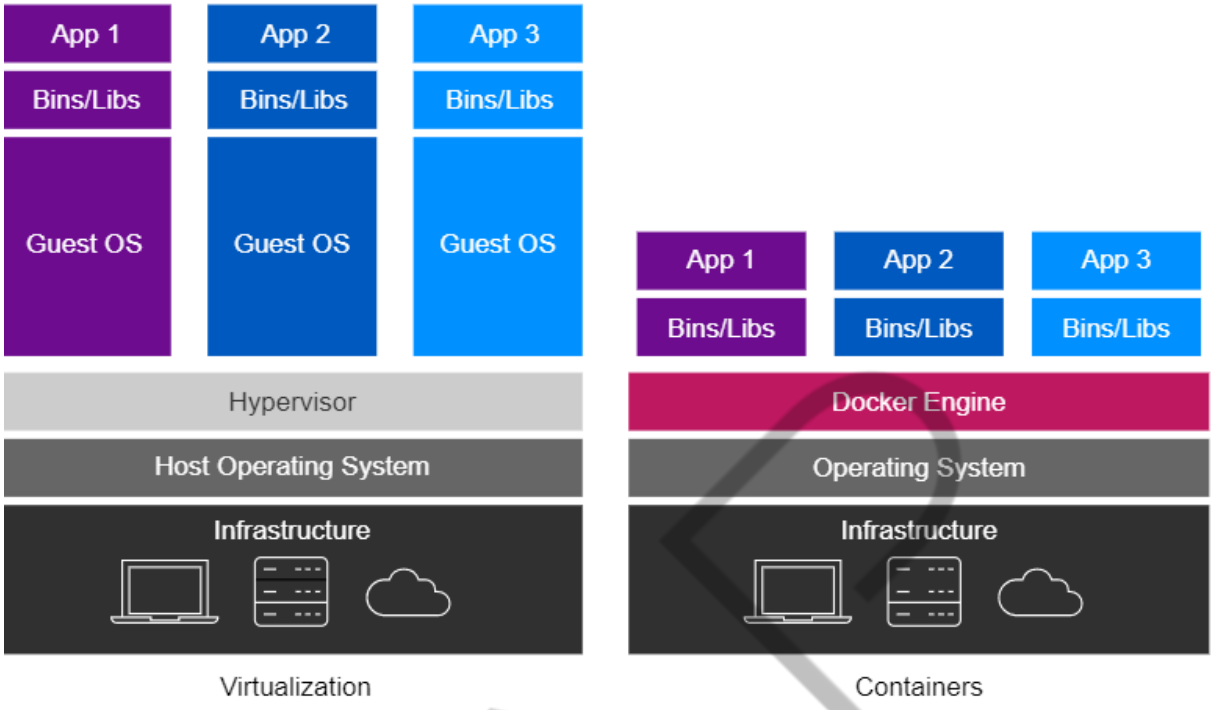

Figura 1.25 – Diferença entre virtualização e contêiner
Fonte: Google Imagens (2019)

Para controlar a distribuição desses contêineres em sistemas na nuvem (*cloud*), podemos utilizar os serviços da Amazon Web Service (AWS) e, para contribuir com a implantação independente da infraestrutura (*cloud* ou *on-premisse*), utilizamos Kubernetes ou Apache Mesos, que veremos a seguir.

1.24 Operação (*Operate*)

Depois de implantado o sistema, podemos operá-lo para modificar comportamentos, escalar ou realizar manutenções e, para contribuir tanto na parte de implantação como na operação do ambiente, temos o Kubernetes e o Apache Mesos, sistemas de orquestração de contêineres *open source* que automatizam a implantação, o dimensionamento e a gestão de aplicações, além de criarem uma camada de abstração em cima da infraestrutura, ou seja, você pode trabalhar com diversos tipos de configuração de hosts na sua infraestrutura, que o Kubernetes ou Apache Mesos irá identificar os recursos e disponibilizar a infraestrutura como um único cluster.

Ambas as ferramentas também possuem funcionalidades de balanceamento de carga (*load balance*),para distribuir a carga de trabalho uniformemente entre os

contêineres, o descobrimento de serviços (*service discovery*), para sua detecção automática, a autorrecuperação (*self healing*), para recuperar contêineres que tiveram problemas ou que não estão respondendo, por meio de uma verificação de saúde no serviço, entre outros.

Figura 1.26 – Logos docker, kubernetes e apache mesos
Fonte: Google Imagens (2019)

Também temos o Ansible, uma ferramenta de automatização para gerenciar múltiplas máquinas de uma vez, que possui uma linguagem bastante simples, sendo possível começar a criar serviços de automação de forma fácil e rápida, além de utilizar SSH para se conectar com os servidores e executar as atividades. O Ansible não utiliza agentes (*agentles*) nas máquinas que operacionaliza, tornando o processo de automação mais eficiente e leve.

Um assunto muito comum na operação é o escalonamento da infraestrutura, que diz respeito à estratégia da sua expansão, em que temos dois modelos, o escalonamento vertical e o horizontal. O escalonamento vertical tem como objetivo aumentar os recursos *host*, aumentando a capacidade de memória, processamento, disco, entre outros. Já o escalonamento horizontal tem como objetivo aumentar a quantidade de *hosts* e dividir o trabalho entre eles. Na arquitetura de microsserviços e contêineres, o escalonamento horizontal é a prática mais adequada, e os orquestradores já estão preparados para isso. Quando utilizamos o escalonamento vertical, aumentar os recursos acaba se tornando uma estratégia mais cara, e pode se tornar limitada, visto que os recursos de infraestrutura maiores custam mais e são limitados até certo ponto. A figura “Exemplificação da estratégia de escalonamento horizontal e vertical” exemplifica as estratégias:

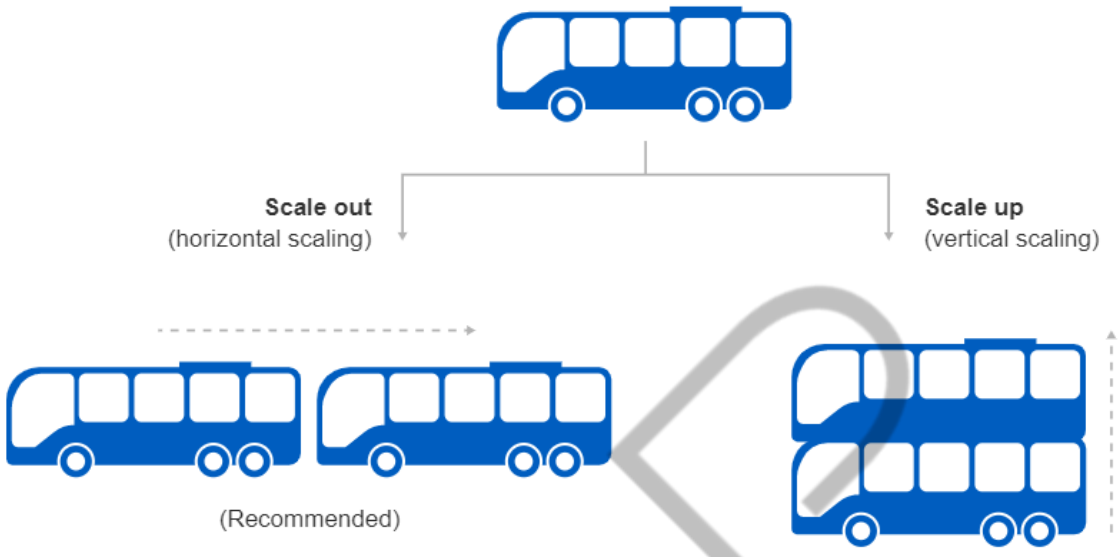

Figura 1.27 – Exemplificação da estratégia de escalonamento horizontal e vertical
Fonte: Google Imagens (2019)

1.25 Monitoração (Monitor)

Embora seja uma fase muito importante no ciclo **DevOps**, a fase de monitoração, por vezes, é esquecida. Com as monitorações, é possível acompanhar o ambiente em tempo real para análise de performance, comportamentos da aplicação e usuário, *troubleshooting* (análise de problemas), entre outros. Ferramentas como o Splunk**,** Datadog e Nagios permitem a indexação de informações geradas pelas aplicações, para realizarmos pesquisa, monitoramento e análise de grandes volumes de dados gerados, por meio de uma interface Web, possibilitando a criação de dashboards iterativos para consolidar e expor as informações.

Figura 1.28 – *Dashboard* de monitoração no splunk
Fonte: Google Imagens (2019)

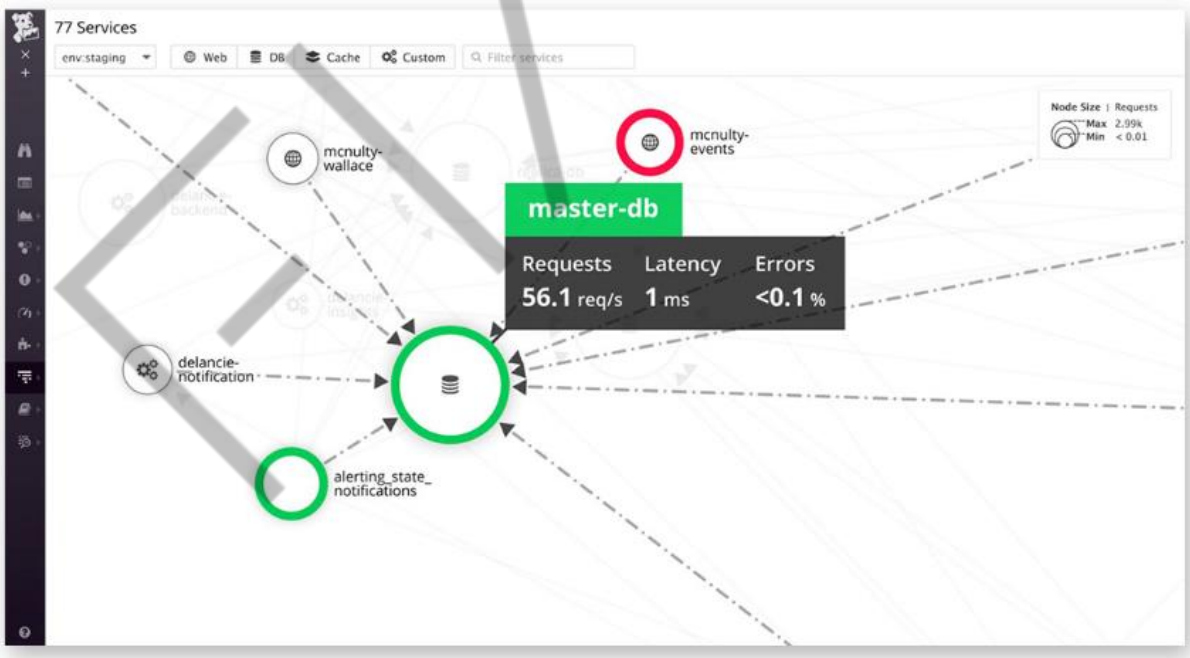

Figura 1.29 – *Dashboard* de monitoração no datadog
Fonte: Google Imagens (2019)

1.26 DEVSECOPS

Durante todo o conteúdo, abordamos diversos pontos relacionados a desenvolvimento, testes, implantação, operação e monitoração, porém, algo que não podemos deixar de citar quando falamos sobre **DevOps** é a segurança.

Muitas vezes, a abordagem sobre segurança só é colocada para discussão após o software chegar à produção, ou depois que ocorre algum problema relacionado à segurança, como vazamento de informações, ataques e demais brechas. Para evitar que o assunto de segurança só seja abordado no final de todo o ciclo, existe um movimento no mercado, conhecido como *Shifting Security Left* (movendo a segurança para a esquerda, se considerarmos que o ciclo de desenvolvimento começa na esquerda e termina na direita). Para que o assunto “segurança” seja abordado no início de todo o ciclo, temos os DevSecOps. As principais vantagens ao se trabalhar com DevSecOps são:

- Segurança distribuída dentro da organização;
- Prevenção e endereçamento de vulnerabilidades encontradas antes da entrega;
- Disseminação da consciência de segurança dentro dos times;
- Softwares mais seguros e com maior qualidade;
- Redução de custo para identificar e resolver um problema de segurança.

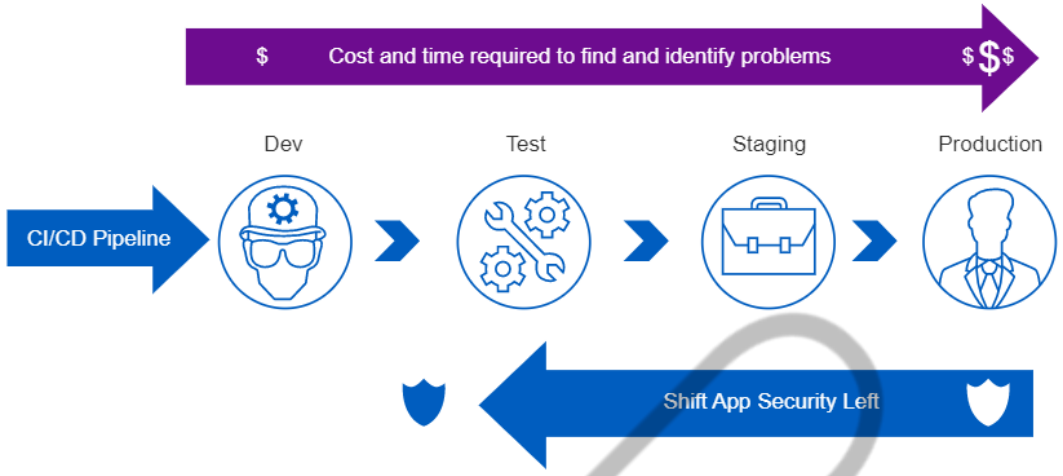

Figura 1.30 – Ilustração do shifiting secutiry left
Fonte: Google Imagens (2019)

REFERÊNCIAS

Amazon Web Services. **O que é DevOps?** Disponível em: https://aws.amazon.com/pt/devops/what-is-devops. Acesso em: 25 fev. 2019.