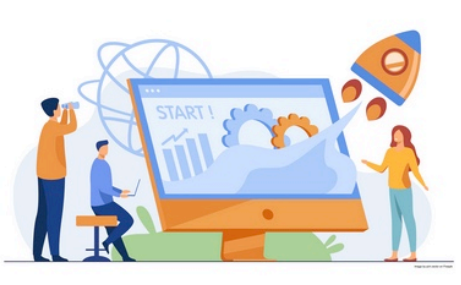


INTRODUÇÃO: DevOps, CI, CD, IaC

DevOps é um conjunto de práticas e ferramentas para a entrega rápida de aplicações com qualidade. Mas esse conjunto de práticas forma uma verdadeira sopa de letras: CI, CD, IaC, VCS, build, etc. Esta primeira aula introduz conceitos vistos durante todo esse curso. Também apresentamos algumas ferramentas que implementam conceitos e são úteis em diversos momentos do desenvolvimento de software. Por fim, explicamos os principais benefícios e os desafios envolvidos quando se adota DevOps.

INTRODUÇÃO

DevOps é um termo usado para representar o agrupamento de um conjunto de práticas e ferramentas para a *entrega rápida de aplicações*. *Entrega rápida de aplicações* significa que o tempo entre o *ciclo* de desenvolvimento e a experiência do usuário no uso do aplicativo é reduzido. Entender esse processo de desenvolvimento e experiência do usuário como um ciclo é chave em DevOps. Em DevOps, esse ciclo entre desenvolvedores e usuários significa que o desenvolvimento é *alimentado* pela experiência do usuário. Com isso, a aplicação evolui para atender as expectativas do usuário.

DevOps usa massivamente ferramentas de automatização para reduzir o tempo no ciclo de entrega de uma aplicação. Mas DevOps **não é** só sobre o uso de ferramentas. DevOps é também um conjunto de práticas. Versionamento de código, integração contínua (CI), entrega contínua (CD), infraestrutura como código (IaC), e monitoramento constante, são algumas das práticas mais conhecidas. Tanto as ferramentas quanto as práticas vão além de funções específicas como *desenvolvedor* ou *testador*. DevOps defende a integração entre diferentes funções. Isso **não** significa que não tem pessoas com funções diferentes em uma equipe, mas sim, que a equipe trabalha de forma integrada.

Essa integração entre funções e equipes é justamente um dos grandes desafios em adotar DevOps. Essa integração requer uma *cultura*. Mudanças culturais não são triviais. Apenas adotar ferramentas e práticas de DevOps em uma empresa que não tem a *cultura* de DevOps trará poucos resultados. Empresas com estrutura rígida e altamente hierárquica *tendem* a ter dificuldades na adoção de DevOps.

Benefícios

O benefício mais visível na adoção de DevOps é a entrega rápida de aplicações. Mas a entrega rápida leva a vários outros benefícios, como qualidade, escala e confiabilidade. *Qualidade* significa que a aplicação atende seus requisitos funcionais e não-funcionais. Isso é assegurado pela execução de testes e monitoramento constante. *Escala* significa que o aplicativo pode ser implantado em um único servidor ou em vários sem degradação de desempenho. Isso é assegurado por práticas como IaC, CI e CD. Finalmente, *confiabilidade* significa que, mesmo em meio a mudanças constantes, existe a certeza de que o processo funciona. Isso é assegurado pela automatização e monitoramento da aplicação e da infraestrutura em todos os passos do ciclo de entrega da aplicação.

Esses benefícios **não** são triviais. Ao contrário, eles são impactantes para garantir a competitividade de empresas. Resultados consolidados de vários anos de pesquisa mostram que empresas que adotam a cultura, práticas e ferramentas DevOps tendem a ter melhor desempenho organizacional, melhor gerenciamento dos seus produtos, e até menor nível de *burnout* [1].

[1] Nicole Forsgren, Jez Humble, and Gene Kim. 2018. Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations (1st. ed.). IT Revolution Press.

Desafios

Apesar dos benefícios, adotar a cultura, as práticas e as ferramentas de DevOps apresenta vários desafios. Resultados recentes mostram que, mesmo com a adoção crescente em todo o mundo, a adoção de práticas DevOps variam bastante, tanto em quantidade quanto em maturidade de adoção. Isso mostra claramente que os desafios têm atrapalhado a adoção de DevOps. Mudanças culturais são um grande desafio, mas não o único. Por exemplo, adotar testes automatizados requer disciplina, treinamento, e pode aumentar o esforço de desenvolvimento em um primeiro momento.

Mesmo com a adoção de algumas práticas, parece não existir um consenso claro em como uma ou outra prática deve ser implementada. Um exemplo clássico é o uso de versionamento de código. *Versionamento de código* é a prática de registrar as mudanças de código a cada edição. Isso permite identificar em que momento um erro foi introduzido no aplicativo, por exemplo. Alguns defendem que os benefícios no uso de versionamento só podem ser alcançados usando uma única *branch* (ramificação da estrutura do código), enquanto outros defendem o uso de várias *branches*. Adotar uma prática pode ser um grande desafio se as discussões ficarem centradas em como implentar a prática, em vez de tentar alcançar o benefício que a prática pode trazer.

Capacitação da equipe é um dos entraves para a adoção de DevOps. Isso porque DevOps **é** um conjunto de práticas e ferramentas. Isso envolve tempo para aprendizado e especialização. *Tempo* é um fator chave. O tempo é necessário não só para aprendizado, mas também para a aplicação das práticas. Como veremos neste curso, criar uma *pipeline* de CI/CD, criar testes, configurar ferramentas de monitoramento, leva tempo. *Pipeline* significa que existe otimização dos recursos usados em uma fila, de modo que o recurso é sempre aproveitado. Gastar mais tempo inicialmente pode parecer contra-produtivo em um primeiro momento. Contudo, quando tudo está em funcionamento, é confortante ter a segurança de que a qualidade do aplicativo está garantida.

Quer entender melhor os benefícios e desafios de DevOps? Que tal acompanhar o relatório anual ***State of DevOps Report*** .

INTEGRAÇÃO CONTÍNUA

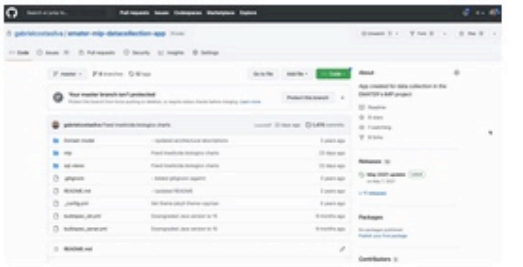
Integração Contínua (CI) é uma das práticas mais comuns de DevOps. Historicamente, CI não surgiu como parte de DevOps, mas foi integrada como uma de suas práticas. CI é um conjunto de atividades que integram código, bibliotecas, e testes, para formar um (ou vários) arquivo(s) de *deploy* de uma aplicação. *Integração* é a palavra-chave aqui. Com a integração, cada atividade pode ser executada automaticamente. Isso gera agilidade, e rápido *feedback*. Por exemplo, se um teste falhar, já fica claro que existe um problema no código. Isso pode ser corrigido antes do aplicativo chegar na fase de *deploy* (implantação no servidor).

O vídeo ao lado mostra um exemplo prático de CI do *manejo.app*. O manejo.app é um aplicativo desenvolvido como parte de um projeto que coordenei na UTFPR em parceria com o Serviço Nacional de Desenvolvimento Rural - PR (SENAR/PR) e o Instituto de Desenvolvimento Rural do Paraná (IDR). O projeto visa aumentar a eficiência do Manejo Integrado de Pragas e Doenças (MIP/MID) na cultura da soja. O manejo.app automatiza a coleta e acompanhamento de dados do MIP/MID, que têm reduzido em mais da metade a aplicação de inseticidas nas lavouras de soja que adotam MIP/MID. Esses resultados têm um enorme impacto financeiro e ambiental.

A primeira parte do vídeo mostra na prática como uma *pipeline* de CI pode ser implementada usando ferramentas em nuvem. No vídeo, são usadas ferramentas da Amazon Web Services (AWS), um dos líderes mundiais de computação em nuvem. No decorrer desta página, explicamos melhor cada passo do processo.

Versionamento de Código

O primeiro passo para criar uma *pipeline* CI é adotar o versionamento de código. Versionamento significa que *versões* de um artefato são salvas. Isso permite (i) identificar quando problemas ocorreram, (ii) retornar modificações anteriores, e (iii) acompanhar a evolução do artefato. *Artefato* é uma palavra-chave aqui. Isso porque versionamento **não** se aplica apenas ao código, mas também a qualquer artefato que evolui no processo de desenvolvimento, incluindo arquivos de configuração, *scripts* de banco de dados e, inclusive, a descrição da infraestrutura.



Git é atualmente a ferramenta de versionamento de código mais usada no mundo. Apesar de não ser a mesma coisa, o Github ajudou muito na popularização do Git. O Github é um repositório de software que adota o Git. Com o Github é possível não só controlar a versão por meio do Git, mas também armazenar os artefatos de forma centralizada na nuvem.

No vídeo, usamos o Github para armazenar os artefatos do manejo.app. Isso inclui código, mas também outros artefatos que fazem parte do projeto, como configuração de ferramentas e *schema* do banco de dados. Nossa *pipeline* busca o código no repositório e inicia o processo de *build*.

Build

Build é um termo usado para representar a *construção de uma aplicação*. Esse é o processo no qual o código de uma aplicação é compilado, (opcionalmente) testado, as dependências são baixadas localmente e ligadas ao código, e um arquivo executável é gerado. Esse processo existe mesmo para linguagens não compiladas, como Python, por exemplo. Isso porque o *build* é mais do que apenas a compilação da aplicação. Esse costuma ser o segundo passo em uma *pipeline* de CI.

Diferente do que acontece com o versionamento de código, o *build* depende de ferramentas específicas de linguagem. Por exemplo, o Apache Maven é a ferramente de *build* mais utilizada no ecosistema Java.

No vídeo, nosso processo de *build* é simplificado - ele consiste em baixar as dependências, executar testes unitários e gerar o arquivo de *deploy*. O arquivo de *deploy* é então entregue a um serviço de armazenamento (Amazon S3), para ser posteriormente usado pelas máquinas virtuais onde o aplicativo é executado.

CI ou CD?

Frequentemente, esses dois termos são usados como se fossem sinônimos. Mas, na verdade, não são. *Entrega contínua* (CD) estende CI levando o arquivo de *deploy* até seu destino final - o servidor. Com isso, a aplicação é implantada em um servidor e disponibilizada para os clientes.

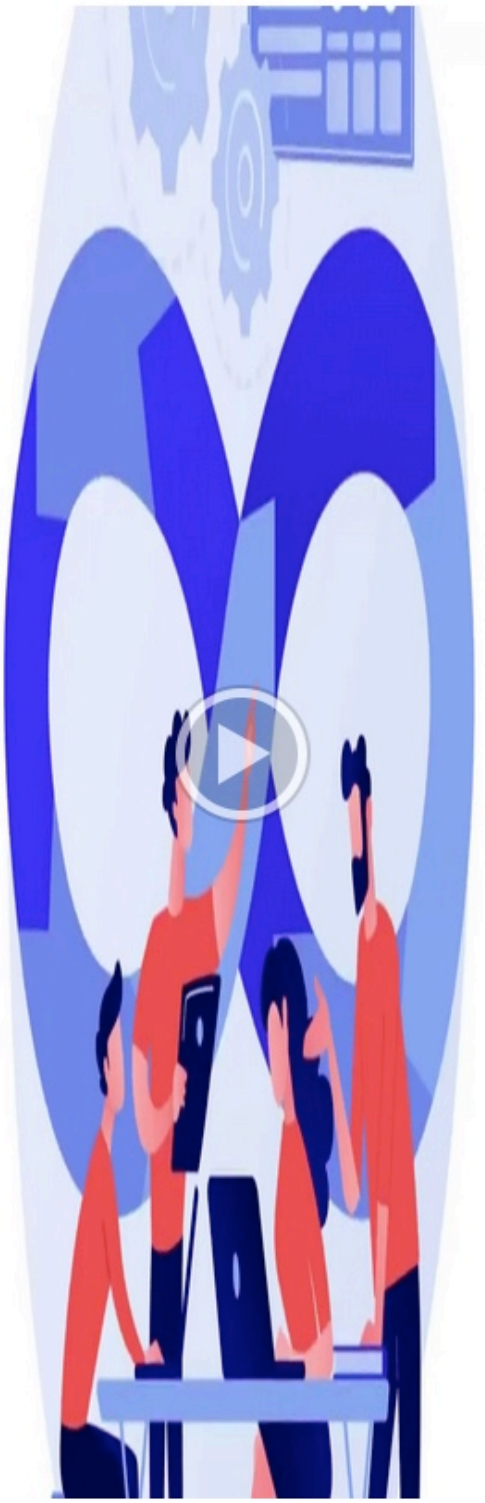
Em Inglês, *continous delivery* e *continous deployment* podem ser usados para diferenciar a entrega contínua entre implantação manual (*continous delivery*) e automatizada (*continous deployment*). Neste curso, nós não fazemos essa diferenciação. Por isso, quando o aplicativo é implantado e disponibilizado para o cliente, usamos o termo CD - independente se isso aconteceu de forma manual ou automatizada.

O vídeo mostra o CD do manejo.app. O CD é realizado de maneira completamente automatizada. Quando o *build* termina, o CodePipeline, serviço de CD da AWS, envia o arquivo para o Amazon S3 - um serviço de armazenamento da AWS. Na sequência, o vídeo mostra a configuração usada pelas máquinas virtuais (VM) do EC2 - o serviço de VMs da AWS. Essa configuração é responsável por conectar no serviço de armazenamento (S3) e baixar o arquivo de *deploy*, que é implantado na VM onde o aplicativo é executado.

Apesar do vídeo mostrar todas as ferramentas em uma mesma plataforma de nuvem (AWS), isso **não é** um requisito para se implementar CI/CD. Contudo, pode ser mais simples de configurar e integrar serviços quando tudo está em um único lugar.

Observe que essa semana ainda estamos introduzindo os assuntos. Por isso, nada nesta semana é profundo ou detalhado. Nas próximas semanas vamos entrar em detalhes em cada um dos tópicos abordados nessa semana, incluindo o uso das ferramentas.

INFRAESTRUTURA COMO CÓDIGO



Infraestrutura como código (IaC) é uma prática na qual a infraestrutura é definida como código, assim como acontece com uma aplicação. Se você não está acostumado com computação em nuvem, talvez isso pareça estranho. Isso porque, na computação em nuvem, o usuário de nuvem gerencia recursos virtuais. Por exemplo, uma máquina virtual (VM) é apenas um conjunto de configurações que é usado por um aplicativo de virtualização (*hypervisor*) para criar um espaço isolado de memória, CPU, rede e outros recursos, para a execução de aplicativos (*sandbox*). Dessa forma, é possível tratar a infraestrutura como se fosse código.

Tratar a infraestrutura como código traz uma série de benefícios, a começar pela possibilidade de colocar o código da infraestrutura em um repositório. Com isso, é possível compartilhar o código com a equipe, acompanhar sua evolução e identificar erros de maneira automatizada.

O vídeo ao lado apresenta o *manejo.infra*. O manejo.infra é o nome do aplicativo que reflete a infraestrutura usada no manejo.app. Assim como um aplicativo tradicional, o manejo.infra consiste de código-fonte que é executado para criar a infraestrutura do manejo.app. O manejo.infra usa a sintaxe (*linguagem de programação*) do AWS CloudFormation. Isso significa que ele é *executado* pelo CloudFormation. O CloudFormation é um serviço da AWS que permite o uso de código para criar infraestrutura. Essa **não é** a única forma de criar infraestrutura usando código.

A aula da próxima semana introduz os principais conceitos de computação em nuvem.

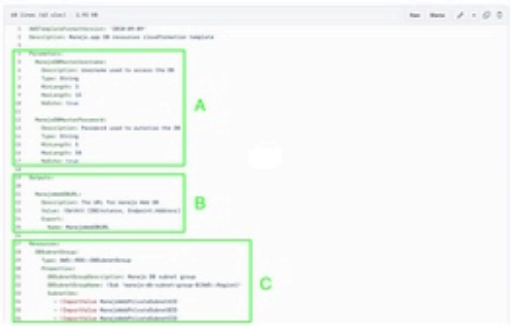
CD da Infraestrutura

Outro benefício direto da IaC é a possibilidade de criar uma CD da infraestrutura. Isso significa que o código da infraestrutura é recuperado em um repositório e implantado em uma plataforma. Assim como o manejo.app, o manejo.infra usa o CodePipeline para criar uma CD da infraestrutura. A principal diferença é que a infraestrutura não passa por uma processo de *build* - o código da infraestrutura não precisa ser compilado ou linkado à biblitecas. Por isso, a *pipeline* consiste de apenas duas atividades.

Observe que a possibilidade de criar e, consequentemente, destruir, a infraestrutura automaticamente, permite economia de recursos em nuvem. Isso porque, em nuvem, os recursos são cobrados por uso - assim como água ou eletricidade. Quanto mais você usa, mais você paga. Por isso, se seu aplicativo é usado apenas em horário comercial, você pode criar toda infraestrutura no início da manhã, alguns minutos antes do início do horário comercial e, ao final da tarde, destruir toda a infraestrutura. Com isso, os recursos alocados são liberados durante a noite, gerando uma economia significativa.

Código da Infraestrutura

IaC é uma prática completamente dependente da ferramenta que está sendo utilizada. Isso significa que a sintaxe pode variar muito. Porém, a ideia é sempre a mesma: representar um conjunto de recursos virtualizados em código. No vídeo, usamos o CloudFormation. A figura ao lado exemplifica o código do CloudFormation para a criação de um banco de dados.



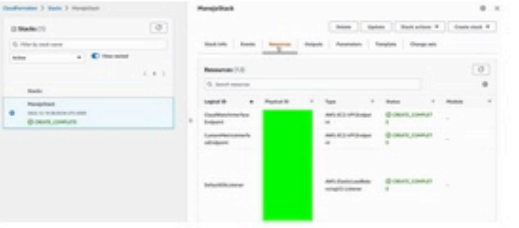
O bloco A mostra os parâmetros de entrada para a criação do banco. Assim como acontece com um aplicativo tradicional, a infraestrutura também pode receber dados de entrada do usuário. Nesse caso, os parâmetros são usados para definir o usuário e senha do banco de dados.

O bloco B mostra os resultados da execução do código. Assim como acontece com um aplicativo tradicional, a infraestrutura também pode emitir resultados para o usuário. No exemplo da figura, o resultado é a URL usada para acessar o banco. Essa URL é usada pelo manejo.app para salvar e recuperar dados persistentes.

Por fim, o bloco C mostra os recursos computacionais que devem ser criados. Esses recursos variam de acordo com o tipo (*Type*, linha 29). O CloudFormation suporta quase todos os recursos da plataforma da AWS. Adicionalmente, cada recurso tem um conjunto de propriedades (*Properties*, linha 30). Essas propriedades restringem ou definem como o recurso deve funcionar.

Resultado

No vídeo, o CloudFormation é acessado para mostrar o resultado da CD - a criação da infraestrutura. A figura ao lado ilustra os recursos que foram criados na execução da CD. Cada recurso pode ser acessado diretamente por meio dessa lista. Isso permite o acompanhamento de cada elemento da infraestrutura. Caso aconteça algum erro durante o processo de criação, esse erro é apresentado na coluna *Status*. Uma fonte comum de erros é a falta de algum elemento que funciona como pré-requisito para outro. Por exemplo, não é possível criar um banco de dados sem definir o tipo de banco (MySQL, PostgreSQL, etc).



Observe que o acesso a um painel geral que reúne todos os elementos da infraestrutura permite o fácil monitoramento e configuração dos recursos. Esse é outro benefício direto da IaC.

MONITORAMENTO

Monitoramento é uma característica fundamental para fechar o ciclo DevOps. Graças ao monitoramento constante, a aplicação pode continuar evoluindo para gerar valor para seus usuários. Nas seções anteriores podemos ver sempre um mecanismo de monitoramento associado aos serviços em uso. Porém, uma visão geral é fundamental para entender o todo. O vídeo ao lado mostra como usamos o Amazon Cloudwatch, um serviço de monitoramento da AWS, para acompanhar os recursos de Infraestrutura e ter uma visão geral da aplicação em uso. Observe que o benefício do monitoramento é facilitado pelo uso de toda a infraestrutura, código, *pipelines*, e monitoramento serem gerenciadas em uma mesma plataforma.

Uso da Infraestrutura

O painel na figura ao lado mostra um painel de controle (*dashboard*) que centraliza o monitoramento de alguns recursos críticos da Infraestrutura. Por exemplo, o primeiro painel (*DB CPUUtilization*) mostra o uso de CPU do banco de dados nas últimas três horas. O gráfico mostra que o uso de CPU esteve abaixo de 3% nas últimas horas. Isso *pode ser* um indicativo de que a CPU alocada para o banco de dados está superdimensionada. Nesse caso, *talvez* o código da infraestrutura precise ser ajustado. Com isso, o custo da infraestrutura pode ser reduzido.

Mas o contexto importa. Por isso, não podemos fazer julgamentos precipitados. O painel mostra baixo uso de CPU do banco de dados, isso pode significar que o aplicativo esta fora do ar e, portanto, o banco não está sendo usado. Por isso que o monitoramento é tão importante. Os demais painéis mostram que o uso de rede está estável, *sugerindo* que não existe problema de conexão da aplicação com o banco de dados.

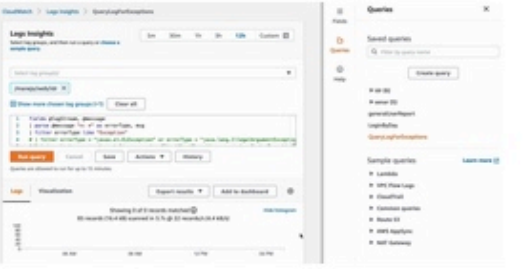
Alarmes

Um benefício direto do monitoramento é a possibilidade de criar alarmes. Esses alarmes servem como um aviso para quando algo não vai bem. Alarmes eliminam a necessidade de um analista humano acompanhar o monitoramento. É claro, o contexto importa, mas não precisa de alguém olhando um painel 24 horas para dizer que o uso de CPU está baixo. Nesse caso, um alarme pode informar alguém que o uso de CPU está baixo e, essa pessoa, analisa outros painéis para determinar o que pode estar acontecendo.

Como exemplo, a figura ao lado mostra um alarme de cobrança. Esse alarme coleta dados de uso e envia um e-mail para o administrador caso as despesas estimadas com a infraestrutura ultrapassem US$ 50,00. Da mesma forma, podemos criar alarmes para diversas métricas que incluem não só a infraestrutura, mas também a própria *pipeline* de CI/CD. Dessa forma, você pode ver o problema antes que ele seja visível para o cliente.

Logs

Coletar e analisar logs de aplicações é uma prática antiga, mas extremamente efetiva para identificar gargalos, problemas e oportunidades em uma *pipeline* de CI/CD. O Cloudwatch é um serviço que também centraliza logs. Isso significa que logs podem ser coletados e enviados para o Cloudwatch. Com isso, podem ser geradas métricas, alarmes e painéis de acompanhamento baseado nesses logs.



A figura ao lado mostra um exemplo de coleta e análise de logs. Os logs são gerados pela aplicação (manejo.app), e entregues ao Cloudwatch para armazenamento. A consulta que você vê na figura busca por erros (*Exceptions*) que tenham sido disparados pela aplicação. Isso não significa, necessariamente, que o usuário viu uma página de erro. O que o usuário vê depende de como a aplicação trata a ocorrência da exceção. De qualquer forma, o fato de um administrador conseguir visualizar a ocorrência de erros em tempo real permite que ações imediatas sejam tomadas para evitar que o erro se repita. Em caso de um bug, por exemplo, o desenvolvedor pode ser informado automaticamente, corrigir o bug, e já fazer o *deploy* da nova versão da aplicação. Dessa forma, talvez o usuário nem perceba que o erro aconteceu.

FERRAMENTAS

Por toda esta lição, falamos de práticas DevOps e demonstramos essas práticas usando um conjunto de ferramentas - a maioria delas é parte da AWS, a plataforma de nuvem da Amazon. Apesar dos vídeos mostrarem um cenário real de aplicação de práticas e ferramentas DevOps, não devemos utilizar apenas os serviços da AWS no restante desse curso. Isso porque a AWS é uma plataforma paga. Por isso, usar seus serviços iria incorrer em custos, apesar da AWS fornecer alguns serviços de forma gratuita nos primeiros 12 meses de uso.

Neste curso, usamos Git e Github para aprender e praticar versionamento de código. Usaremos o Apache Maven para o *build* da aplicação. Usamos JUnit para testes unitários, e Docker, para conteinerizar nossa aplicação. Por fim, vamos utilizar AWS CodePipeline para integrar as demais ferramentas e formar nossa CI/CD.