UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Integração e Entrega Contínua para aplicações móveis desenvolvidas em React Native

Pedro José de Souza Neto

Trabalho de Graduação

Recife, Dezembro de 2018

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Integração e Entrega Contínua para aplicações móveis desenvolvidas em React Native

Trabalho apresentado ao programa de Graduação em Sistemas de Informação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Vinicius Cardoso Garcia

Recife, Dezembro de 2018

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

Pedro José de Souza Neto

Integração e Entrega Contínua para aplicações móveis desenvolvidas em React Native

Monografia submetida ao corpo discente da Universidade Federal de Pernambuco, defendida e aprovada em ___ de _____ de _____ .

Banca Examinadora:

Orientador: _____

Vinicius Cardoso Garcia

Examinador: _____

Kiev Santos da Gama

Recife, Dezembro de 2018

AGRADECIMENTOS

Como disse Richard Feynman: "Estude muito o que mais lhe interessa da maneira mais indisciplinada, irreverente e original possível."

Dito isso, agradeço a Deus pelo seus dons, proteção e livramentos concedidos até aqui.

Entre todas as pessoas que devo agradecimento, as que mais tenho motivos para agradecer são meus pais, Luciano Francisco (Branco) e Maria da Conceição (Bebel) que desde os meus 12 anos quando descobrir o que era programação e decidir que era isso que queria para minha vida, dentro de suas limitações financeiras, fizeram o possível para que eu conseguisse ter acesso aos meios necessários para desenvolver esse saber e me manter sempre ansioso em querer aprender mais sobre esse mundo que estava descobrindo.

À minha namorada, Juliana Amaral, que me acompanhou nessa trajetória com paciência nos dias que não podia sair para concluir algum projeto que precisava ser entregue antes das 23:59.

À minha tia, Maria Verônica, por me receber em sua casa e me tratar como um filho.

À minha prima, Beatriz Borges, pelo convívio diário nesses últimos anos e as minhas irmãs, Maria Cecília e Maria Vitória, por me suportar durante o fim de semana quando voltava para Limoeiro.

Ao professor, Vinicius Cardoso, por ser um cara que me inspirou a ir mais profundo na engenharia de software, me deu oportunidade para transmitir o conhecimento para outros estudantes e, principalmente, me fez ser um profissional muito melhor em todos os aspectos tanto técnico quanto ético.

E por fim, mas não menos importante, aos meus grandes amigos e parceiros de jornada, Augusto Lima, Carlos Melo e Rafael Jordão, os WildCats, que estiverem sempre junto comigo durante todo percurso e tornaram, sem nenhuma sombra de dúvidas, essa jornada muito edificante e divertida.

A todos, muito obrigado!

RESUMO

Integração e entrega contínua não é novidade para desenvolvedores que criam aplicações para web, todavia no desenvolvimento de aplicativos móveis essa prática ainda não é muito difundida principalmente pelo fato dos desafios durante o processo de distribuição da aplicação. Diante do crescente número de aplicativos, uma exigência maior em relação a qualidade e com os prazos de entrega cada vez menores, entregar um código saudável e com frequência é extremamente importante para acompanhar a concorrência. A proposta deste trabalho é implementar uma pipeline de integração e entrega contínua para aplicativos móveis desenvolvidos em React Native. Tendo como objetivo automatizar o processo de *build* e entrega de aplicações desenvolvidas com essa tecnologia.

Palavras-chave: pipeline, integração contínua, entrega contínua, react native, aplicativo móvel

ABSTRACT

Continuous integration and continuous delivery is not new for developers who create web applications, however in the development of mobile applications this practice is still not very common mainly because of the challenges during the process of distributing the application. In the face of the growing number of applications, a greater requirement for quality and ever-shorter delivery times, delivering a healthy code is often extremely important to keep up with the competition. The purpose of this work is to implement a integration and continuous delivery pipeline for mobile applications developed in React Native. Its intends to automate the process of build and delivery of applications developed with this technology.

Keywords: pipeline, continuous integration, continuous delivery, react native, mobile application

LISTA DE FIGURAS

2.1 Abstração da arquitetura de uma aplicação híbrida

2.2 Abstração da arquitetura de uma aplicação em React Native

2.3 Abstração de uma pipeline DevOps

2.4 Relacionamento entre integração contínua, entrega contínua e implantação contínua

2.5 Diferença entre Entrega contínua e Implantação contínua

3.1 Alteração na parcela de votos do Stack Overflow entre janeiro de 2015 e janeiro de 2016

3.2 Código do teste de aceitação

3.3 Código e a tela de listagem de professores

3.4 Código e a tela de detalhe de professor

4.1 Pipeline desenvolvida para este estudo

4.2 Estrutura de pipeline

4.3 Erro durante a execução da pipeline

4.4 Assinatura do aplicativo para distribuição na Google Play

4.5 Notificação para o time de desenvolvimento

4.6 Notificação para o grupo de usuários

4.7 Todas as etapas da pipeline após a execução.

SUMÁRIO

1. INTRODUÇÃO 9
1.1 Contexto e motivação 9
1.2 Objetivos 9
1.3 Estrutura do documento 10

2. CONCEITOS BÁSICOS 12
2.1 Frameworks de desenvolvimento em JavaScript 12
2.1.1 Angular 12
2.1.2 Vue 13
2.1.3 React 13
2.2 Desenvolvimento para aplicações móveis 13
2.2.1 React Native 15
2.3 DevOps 17
2.3.1 Integração contínua 18
2.3.2 Entrega contínua 18
2.3.3 Implantação contínua 19
2.4 Sumário 20

3. IMPLEMENTAÇÃO 21
3.1 Tecnologia adotada 21
3.2 Implementação dos testes 22
3.3 Implementação do aplicativo 24
3.4 Sumário 26

4. A PIPELINE DE INTEGRAÇÃO E ENTREGA CONTÍNUA 27
4.1 Configuração de ambiente 27
4.1.1 Jenkins 27
4.2 Estrutura da pipeline 28
4.2.1 Jenkinsfile 28
4.2.2 Poll SCM Trigger 29
4.2.3 Settings 29
4.2.4 Build Debug Mode 30
4.2.5 Acceptance Test 32
4.2.6 Build Release Mode 32
4.2.7 Delivery 34
4.3 Discussão 36

5. CONCLUSÕES 38
5.1 Dificuldades e lições aprendidas 38
5.2 Trabalhos futuros e limitações 38
REFERÊNCIAS 39

1. INTRODUÇÃO

O foco deste trabalho é a implementação de uma pipeline, uma sequência de operações automatizadas, de integração e entrega contínua [5,6,9] para aplicações móveis desenvolvidas em React Native [1]. Este capítulo descreve inicialmente o contexto e a motivação do trabalho. Em seguida, é descrito os objetivos traçados no início do estudo. A última Seção desta introdução descreve a estrutura deste trabalho.

1.1 Contexto e motivação

O crescente número de aplicações de software, uma exigência maior em relação a qualidade e com os prazos de entrega cada vez menores, acarretou em uma mudança que afetou o processo de desenvolvimento e a maneira que o software é entregue para o cliente.

Diante desse cenário, vimos o surgimento de metodologias como DevOps [2], que tem como base a prática de integração e entrega contínua no seu ciclo operacional com a finalidade de aumentar a capacidade de resposta às necessidades dos clientes por meio de lançamentos de software frequentes e automatizados [10].

Com a propagação dessas práticas supracitadas, muitas empresas desenvolveram diversas soluções aplicando integração contínua [1] e entrega contínua [2] para aplicações web, porém no mundo de aplicações móveis, a adoção dessas práticas ainda é pouco utilizada [19].

1.2 Objetivos

O objetivo deste trabalho é o estudo e a implementação de uma pipeline de integração e entrega contínua para aplicações móveis desenvolvidas em React Native [3] com fins de automatizar o processo de *build* e entrega destas aplicações.

- Criação da *pipeline* da aplicação
- Estudo de ferramenta de integração e entrega contínua
- Estudo de ferramenta para testar aplicações React Native
- Implementação de uma aplicação em React Native e dos testes baseados nos estudos para realizar as funções desejadas

Para o desenvolvimento da *pipeline*, foram utilizadas as ferramentas Jenkins[1], que é o principal servidor de automação *open source* que fornece centenas de plugins para apoiar os processos de *build*, implantação e automação para diversos tipos de projeto, Blue Ocean[2], que repensa a experiência do usuário do Jenkins, reduz a desordem e aumenta a clareza para todos os membros da equipe, TestFairy [3], que é uma plataforma de testes para aplicações móveis que fornece registros e relatórios de falhas além de ser um meio de centralizar a distribuição do aplicativo e o Slack[4], que é uma canal de comunicação presente em quase todas as empresas além de possuir ótimos meios de integração com as demais ferramentas utilizadas nesse trabalho.

1.3 Estrutura do documento

Esse documento é composto por 5 capítulos:

- Capítulo 1 - Introdução: Apresenta motivações e objetivos do estudo

- Capítulo 2 - Conceitos básicos: Apresenta um contexto teórico e tecnológico para a compreensão dos capítulos seguintes.

- Capítulo 3 - Implementação do aplicativo: Apresenta a escolha da tecnologia adotada, o desenvolvimento dos testes e do aplicativo.

[1] https://jenkins.io/
[2] https://jenkins.io/projects/blueocean/
[3] https://www.testfairy.com/
[4] https://slack.com/

- Capítulo 4 - A *pipeline* de integração e entrega contínua: Apresenta o passo a passo no processo de construção da pipeline.

- Capítulo 5 - Conclusões: Apresenta as conclusões obtidas durante o desenvolvimento do projeto, trabalhos futuros e limitações.

2. CONCEITOS BÁSICOS

2.1 Frameworks de desenvolvimento em JavaScript

No princípio, o JavaScript[5] era utilizado na construção de animações e para fornecer interatividade para algumas páginas da web. Contudo, não existia nenhum tipo de comunicação com o servidor, todas as ações eram voltadas apenas para exibição de artefatos.

Com o advento do AJAX[6], o JavaScript começou a se comunicar com o servidor e com isso passou a realizar tarefas mais importantes nas aplicações web. Atualmente é raro encontrar aplicações que não utilizem JavaScript para funcionar.

Devido a popularização da linguagem, algumas dificuldades dos desenvolvedores começaram a ficar mais evidentes, como a necessidade de manipulação do DOM[7], *Document Object Model*, e da interação do usuários através de eventos, por exemplo: o clique do mouse. As funções já existentes não possuíam uma solução eficiente para tais necessidades e começaram a surgir frameworks como Angular, Vue e React que fornecem um caminho para suprir essas dificuldades.

2.1.1 Angular

AngularJS[8], depois apenas Angular, foi criado em 2010 por uma equipe de desenvolvedores da Google e foi o pioneiro na popularização de frameworks em JavaScript. Ele surge focado no desenvolvimento de SPAs, *Single Page Applications*, com uma arquitetura MVC, *Model-View-Controller*, separando a lógica do aplicativo, lógica de exibição e modelos.

O Angular permite que o desenvolvedor possa estender o HTML criando novas *tags* que podem ser reutilizada em outra parte do código quando necessário.

[5] https://en.wikipedia.org/wiki/JavaScript
[6] https://en.wikipedia.org/wiki/Ajax_(programming)
[7] https://en.wikipedia.org/wiki/Document_Object_Model
[8] https://en.wikipedia.org/wiki/AngularJS

Outra característica importante que ficou conhecido por *two-way data-bindings*, vinculava o modelo com o objeto JavaScript permitindo a atualização automática quando acontecia mudança no HTML ou no próprio modelo.

2.1.2 Vue

Vue[9] é bastante parecido com Angular, foi criado por Evan You depois de trabalhar na Google utilizando Angular com o objetivo de ser um framework mais simples de se utilizar e na redução do número de arquivos usados.

Na documentação oficial, Vue é definido como um framework progressivo para construção de interfaces de usuário. Em suas últimas versões já apresenta uma abordagem baseada em componentes e um DOM virtual de maneira similar ao React.

O DOM virtual é uma representação de um objeto DOM. Um objeto DOM virtual possui as mesmas propriedades de um objeto DOM, no entanto manipulá-lo é muito mais rápido.

2.1.3 React

Na documentação oficial, React[10] é definido como uma biblioteca JavaScript declarativa, eficiente e flexível para criar interfaces com o usuário através de componentes.

React foi criado pelo Facebook e o primeiro framework a usar um DOM virtual. Em relação a sua estrutura, React não utiliza o conceito de MVC, sua visualização é representada como uma árvore de entidades chamadas componentes que podem ser compostos de outros componentes.

2.2 Desenvolvimento para aplicações móveis

Depois de dominar o mundo da web, o JavaScript começou a ser utilizado como uma alternativa para lidar com o desenvolvimento multiplataforma em dispositivos móveis. Aplicativos híbridos, como ficou conhecido, é a combinação de tecnologias

[9] https://vuejs.org/
[10] https://reactjs.org/

da web como HTML, o próprio JavaScript e CSS, *Cascading Style Sheets*, para construção de aplicativos móveis.

Os aplicativos híbridos são hospedados em uma aplicação nativa executando em cima de uma *WebView*, que é o nome dado ao *browser* encapsulado dentro da aplicação. Isso permite o acesso aos recursos do dispositivo como câmera, contatos, geolocalização etc. A Figura 2.1 apresenta uma abstração da arquitetura de um aplicativo híbrido.

Figura 2.1: Abstração da arquitetura de uma aplicação híbrida

Assim como ocorreu na web, começaram a surgir frameworks JavaScript focados no desenvolvimento de aplicações híbridas como o PhoneGap/Apache Cordova entregando um conjunto de *plugins* que auxiliam o acesso aos recursos do aparelho. O PhoneGap[11] foi criado em 2009 pela empresa Nitobi, porém em 2011 a Adobe comprou a empresa e doou o código para o projeto Apache. Depois desse acontecimento, nasceu o Apache Cordova[12] projeto *open-source* mantido por sua comunidade.

A principal limitação de uma aplicação híbrida é quando se fala em performance. Atualmente a abordagem híbrida não consegue ser tão performática

[11] https://phonegap.com/
[12] https://cordova.apache.org/

que uma aplicação nativa, justamente pelo fato do poder de renderização da *WebView*. Quando falo em aplicações nativas significa que o desenvolvimento da aplicação foi realizado com a linguagem de programação padrão do SDK, *Software Developer's Kit*, do dispositivo. No caso de Android temos o Java e, mais recentemente, Kotlin e para iOS temos o Objective-C e Swift.

Todavia, a aplicação híbrida possui como maior ponto positivo a capacidade de, através de um único código fonte, entregar a aplicação em múltiplas plataformas, trazendo consigo uma série de benefícios como: velocidade de desenvolvimento e facilidade de manutenção do código.

Diante desse cenário, temos o surgimento do React Native como uma solução que consegue melhorar o problema de performance da aplicação híbrida e mantém a prática de, através de um único código, entregar a aplicação nas principais plataformas móveis.

2.2.1 React Native

O React Native [1], assim como o React, foi criado e é mantido pelo Facebook. React Native utiliza o mesmo design do React, permitindo compor ricas interfaces de usuário a partir de componentes declarativos.

Com React Native, você não cria um aplicativo híbrido utilizando as ferramentas oriundas da web, no entanto é possível criar um aplicativo móvel real que é indistinguível de um aplicativo nativo, pois ele usa os mesmos blocos de construção fundamentais da interface do usuário dos aplicativos iOS e Android comuns. Diferente de ferramentas como PhoneGap/Apache Cordova que executa o JavaScript e permitir sua interação com recursos nativos, o React Native compila o JavaScript ao código nativo, explicando assim como ele consegue melhorar a performance da aplicação. A Figura 2.2 apresenta uma abstração da arquitetura do React Native. React Native vem crescendo junto com sua comunidade e sendo adotado por grandes empresas como: Instagram, Uber e o próprio Facebook.

Os principais problemas associados ao React Native são de gerenciamento de configuração, dependências e testes. Por ser apenas um código, na teoria se um teste passou no ambiente iOS, podemos assumir que o teste também passará no

ambiente Android. Na maioria dos casos o cenário descrito acima é verídico, que é testemunho do poder do React Native, porém existem exceções suficientes para causar problemas no ciclo de controle e garantia de qualidade do aplicativo, como por exemplo: bibliotecas que funcionam bem para iOS, porém não no Android.

Diante desse cenário, uma abordagem para solucionar tais problemas é o desenvolvimento e utilização de uma pipeline DevOps. DevOps se propõe a resolver esse problema combinando a mudança cultural com a automação de entrega de software com objetivo de garantir lançamentos de software frequentes, previsíveis e sem erros.

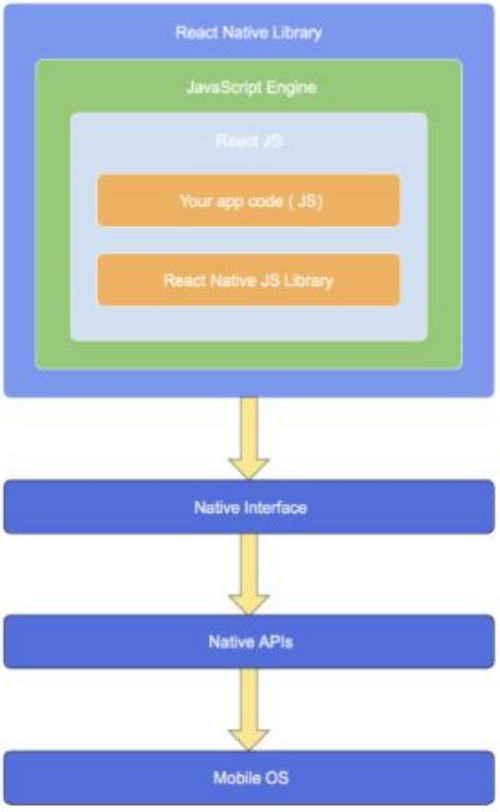

Figura 2.2: Abstração da arquitetura de uma aplicação em React Native

2.3 DevOps

DevOps é um conjunto de práticas destinadas a reduzir o tempo entre fazer uma alteração em um sistema e a mudança ser colocada em produção normal, garantindo alta qualidade [16].

DevOps surgiu na engenharia de software com ênfase em colaboração, automação e utilização de ferramentas como ponte de comunicação entre as atividades de desenvolvimento e operações, em resposta a crescente utilização das metodologias ágeis e ao aumento da demanda de produção de aplicativos de software, que necessitavam de entregas rápidas, contínuas e com mais qualidade. A Figura 2.3 apresenta uma abstração de uma pipeline DevOps.

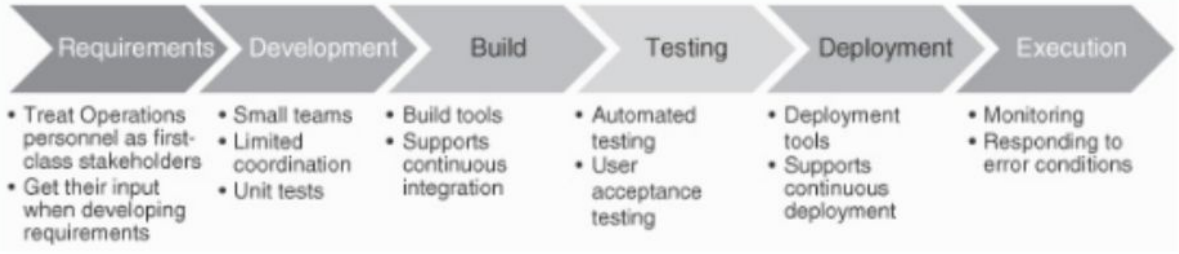

Figura 2.3: Abstração de uma pipeline DevOps

DevOps se inclui área emergente de pesquisa e prática chamada de Engenharia de Software Contínua. Essa área refere-se ao desenvolvimento, implantação e obtenção de feedback rápido do software e do cliente em um ciclo muito rápido [5,17]. Engenharia de Software Contínua envolve três fases: Estratégia de Negócio e Planejamento, Desenvolvimento e Operações. Para este estudo iremos focar nas atividades relacionadas a fase de Desenvolvimento: Integração contínua, Entrega contínua e Implantação contínua. A Figura 2.4 apresenta o relacionamento entre essas atividades.

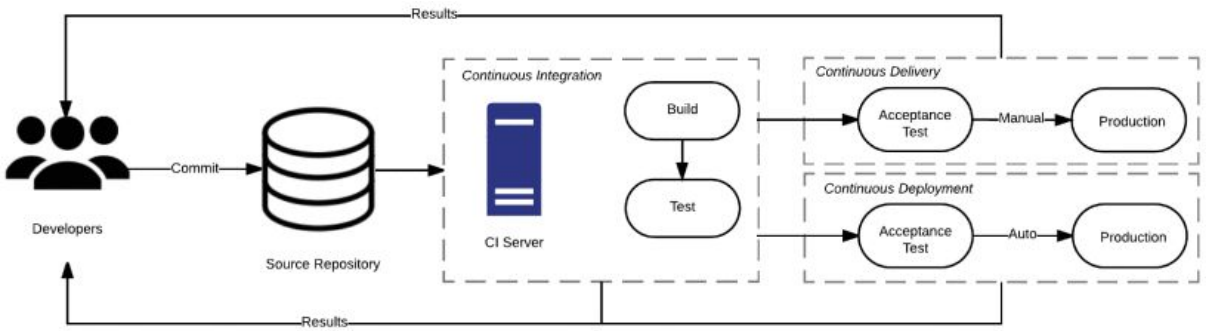

Figura 2.4: Relacionamento entre integração contínua, entrega contínua e implantação contínua [7]

2.3.1 Integração contínua

Integração contínua é um conjunto de práticas de desenvolvimento de software, no qual os desenvolvedores publicam regularmente as alterações em seus códigos com o propósito de reduzir o tempo para disponibilizar novas atualizações no software [3].

Martin Fowler define integração contínua como: "uma prática de desenvolvimento de software em que os membros de uma equipe integram seu trabalho com frequência, geralmente cada pessoa realiza pelo menos uma integração diariamente - levando a múltiplas integrações por dia. Cada integração é verificada por uma compilação automatizada (incluindo teste) para detectar erros de integração o mais rápido possível" [4].

A utilização dessa prática permite que as empresas de software tenham um ciclo de lançamento mais curto e frequente, melhorem a qualidade do software e aumentem a produtividade de suas equipes. Esta prática inclui a construção de um servidor com uma ferramenta de integração contínua como o Jenkins e implementação de testes automatizado.

2.3.2 Entrega contínua

Segundo Humble [8], entrega contínua é a capacidade de obter mudanças de todos os tipos - incluindo novos recursos, alterações de configuração, correções de erros e experimentos - na produção ou nas mãos dos usuários, com segurança e rapidez de maneira sustentável.

Entrega contínua nasce com o objetivo de substituir o processo de entrega de software tradicional e tal prática é possível garantindo um bom processo de integração contínua, deixando o software em um estado entregável após cada mudança efetivada [9] após passar com sucesso em testes automatizados e verificações de qualidade.

De acordo com Chen [11], esta prática oferece bastante benefícios como como redução do risco de implantação, redução de custos, aumento da satisfação do usuário, produtividade e eficiência. Conforme apresentado na Figura 2.4 esta prática tem como requisito a utilização de integração contínua.

2.3.3 Implantação contínua

Implantação Contínua é uma prática de desenvolvimento de software na qual cada alteração de código passa por toda a pipeline e é colocada em produção, automaticamente.

Não existe um consenso no meio acadêmico sobre a definição e diferença entre Entrega contínua e Implantação contínua [5], no entanto é importante ressaltar que a prática de Implantação contínua implica obrigatoriamente na prática de Entrega contínua, mas o inverso não é verdadeiro [18].

A principal diferença entre Entrega contínua e Implantação contínua está no momento da entrega do software. Na Entrega contínua, seu software estará sempre em um estado entregável após as mudanças efetivadas, porém o momento de distribuição para o cliente final é um passo manual, ou seja, uma decisão de negócio. No entanto, conforme apresentado na Figura 2.5, na Implantação contínua o processo é totalmente automatizado.

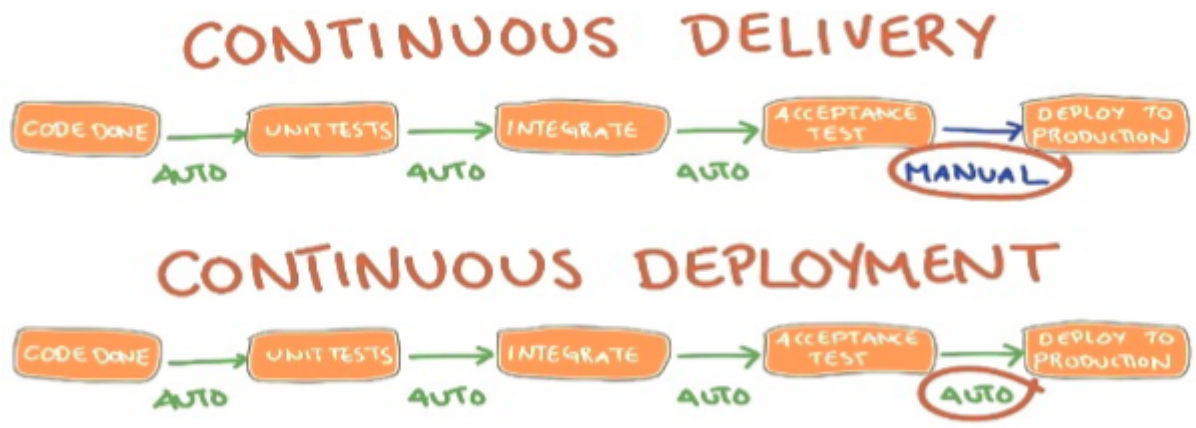

Figura 2.5: Diferença entre Entrega contínua e Implantação contínua [18]

2.4 Sumário

Neste capítulo foi apresentado todo um contexto teórico e tecnológico para termos uma noção sobre os benefícios e desafios associados à implantação de uma pipeline DevOps.

O próximo capítulo apresenta como foi realizado o desenvolvimento do aplicativo bem como a implementação de seus testes.

3. IMPLEMENTAÇÃO

Neste capítulo apresenta a escolha pela tecnologia adotada para o desenvolvimento do aplicativo, bem como sua implementação e os *scripts* de testes elaborados para o propósito deste estudo.

3.1 Tecnologia adotada

Conforme apresentado na Seção 2.1, no capítulo de conceitos básicos, React Native é uma biblioteca para construção de aplicativos móveis. Pelo fato desta tecnologia ter JavaScript em seu *core*, é muito mais fácil encontrar desenvolvedores em comparação a outras linguagens como Swift ou Kotlin.

A StackOverflow[13], maior comunidade *online* de desenvolvedores do mundo, publica todo ano um *survey* apresentando um consolidado das respostas de seus usuários sobre vários temas, desde tecnologias favoritas até preferências de trabalho. No seu *survey* [12] publicado neste ano, JavaScript atingiu a marca de linguagem de programação mais popular do mundo pelo sexto ano consecutivo. O *survey* publicado em 2016 [13] apontou React como a principal tendência tecnológica no contexto de desenvolvimento como ilustrado na Figura 3.1, a confirmação desta tendência está presente no *survey* deste ano no qual React aparece na terceira colocação na categoria de *Frameworks, Libraries and Tools* mais utilizada atualmente.

[13] https://stackoverflow.com/

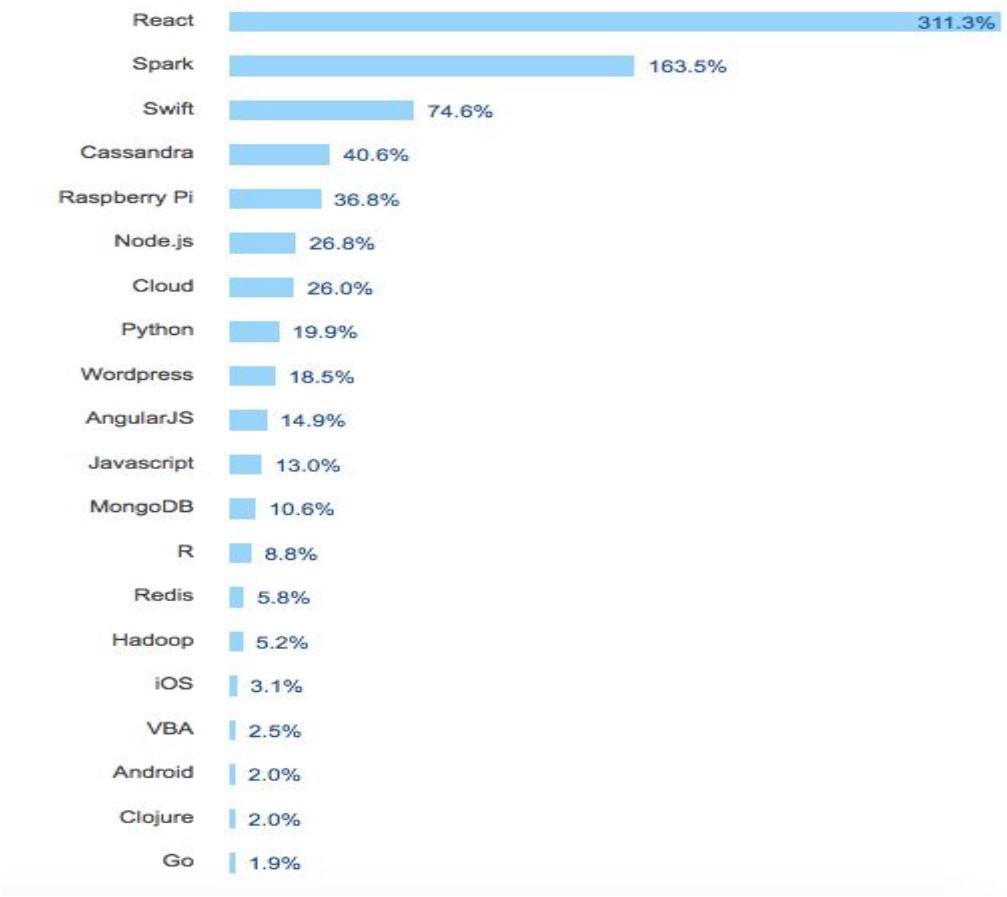

Figura 3.1: Alteração na parcela de votos do Stack Overflow entre janeiro de 2015 e janeiro de 2016 [13]

3.2 Implementação dos testes

Brian Marick [14], em uma série de *posts*, fez uma classificação de testes de *software* em uma forma chamada *The Marick Test Matrix*. De acordo com Marick, podemos dividir a área de testes em quatro tipos, são elas: testes de aceitação, unitário, exploratório e não funcional.

Neste estudo, para atender o objetivo, focamos na implementação apenas dos testes de aceitação que representa requisitos funcionais vistos da perspectiva de negócio. Geralmente, esse tipo de teste, é escrito na forma de histórias de usuários exemplificando como o *software* deve funcionar. O aplicativo desenvolvido para este estudo, consiste em uma listagem de professores e para o mesmo foram definidos duas histórias de usuários:

- Como usuário, eu quero visualizar a lista de professores
- Como usuário, eu quero visualizar os detalhes de um professor

Os testes presentes na Figura 3.2 foram implementados utilizando a ferramenta *open-source* Calabash[14] em sua versão 0.9.7. Calabash permite a escrita e execução de testes de aceitação automatizados de aplicativos móveis. Ele consiste em uma biblioteca que permite o código de teste interagir com o aplicativo simulando ações do usuário final.

```
require 'calabash-android/calabash_steps'

Given(/^the app has launched$/) do
 wait_for do
   !query("*").empty?
 end
end

Then("I see home page") do
 wait_for_element_exists("* marked:'Teachers List'")
 puts "Home Page"
 sleep(5)
end

When("I press some teacher") do
 touch("* marked:'TeacherItem' index:0")
 sleep(2)
end

Then("I go to teacher details") do
   wait_for_element_exists("all * marked:'Teacher Details'")
   puts "Teacher Details Page"
   sleep(2)
end
```

Figura 3.2: Código do teste de aceitação

As palavras reservadas *Given*, *When* e *Then* são expressões do Cucumber[15], ferramenta que permite a utilização de linguagem natural para expressar o comportamento do usuário ao utilizar o aplicativo. No trecho de código abaixo estamos descrevendo a ação do usuário de abrir a aplicação.

[14] https://calaba.sh/
[15] https://cucumber.io/

```
Given(/^the app has launched$/) do
  wait_for do
    !query("*").empty?
  end
end
```

No entanto, as expressões *wait_for*, *query*, *wait_for_element_exists*, *sleep* e *touch* são palavras reservadas do Calabash responsáveis por simular as ações do usuário interagindo com a aplicação. No trecho de código abaixo estamos simulando a ação de clique do usuário em um item da listagem de professores.

```
When("I press some teacher") do
  touch("* marked:'TeacherItem' index:0")
  sleep(2)
end
```

3.3 Implementação do aplicativo

Após a definição das histórias de usuários e a elaboração dos casos de teste foi iniciado a fase de implementação do aplicativo em React Native na versão 0.55.2. A Figura 3.3 apresenta o código e a tela de listagem dos professores e a Figura 3.4 apresenta o código e a tela de detalhe de professor. Todo código desenvolvido está disponível em um repositório público[16] no Github.

O trecho de código abaixo ilustra a classe responsável pela tela de listagem dos professores.

```
.....
class ListTeachers extends Component {
 static navigationOptions = { title: 'Teachers List',};
 render() { return (
   <FlatList data={[...]} renderItem={ ({item}) =>
    <TouchableOpacity onPress={()=> this.props.navigation.navigate('Detail', {
item: item })} accessibilityLabel= {"TeacherItem"}>
           <View style={styles.itemView}>
              <Image source={{ uri: item.photo}} style={styles.photo}/>
                 <Text style={styles.itemList}>{item.name}</Text>
           </View>
.....
```

[16] https://github.com/pedrojsn96/intro-react/tree/master/TeachersProject

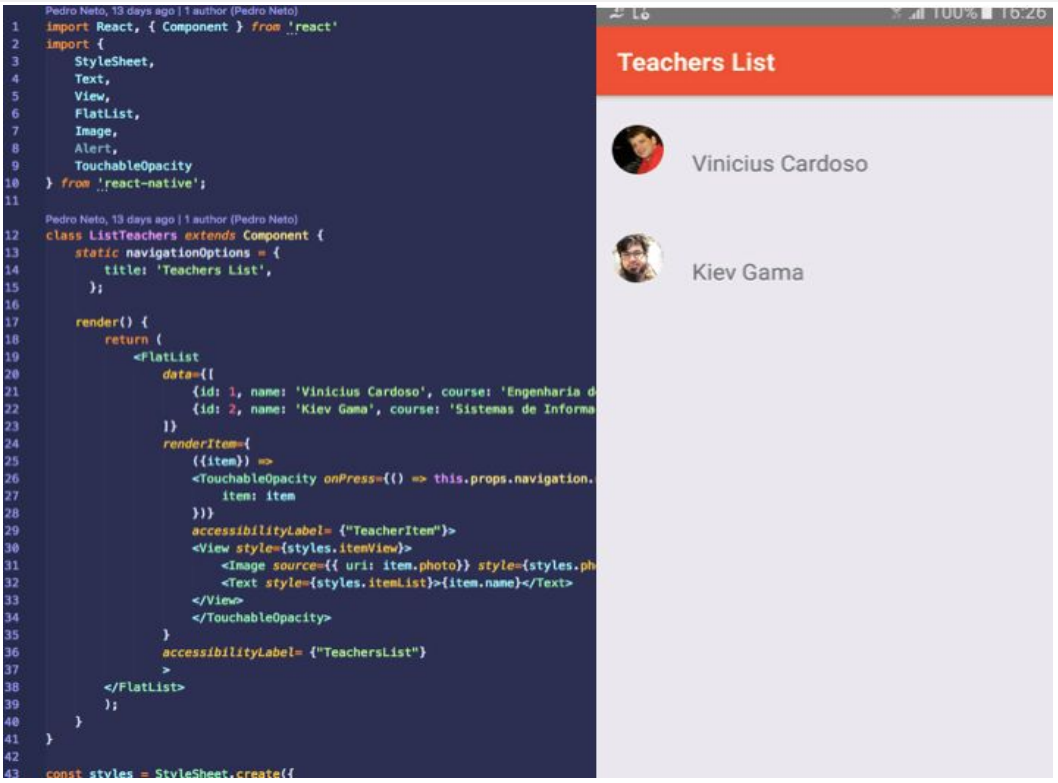

Figura 3.3: Código e a tela de listagem de professores

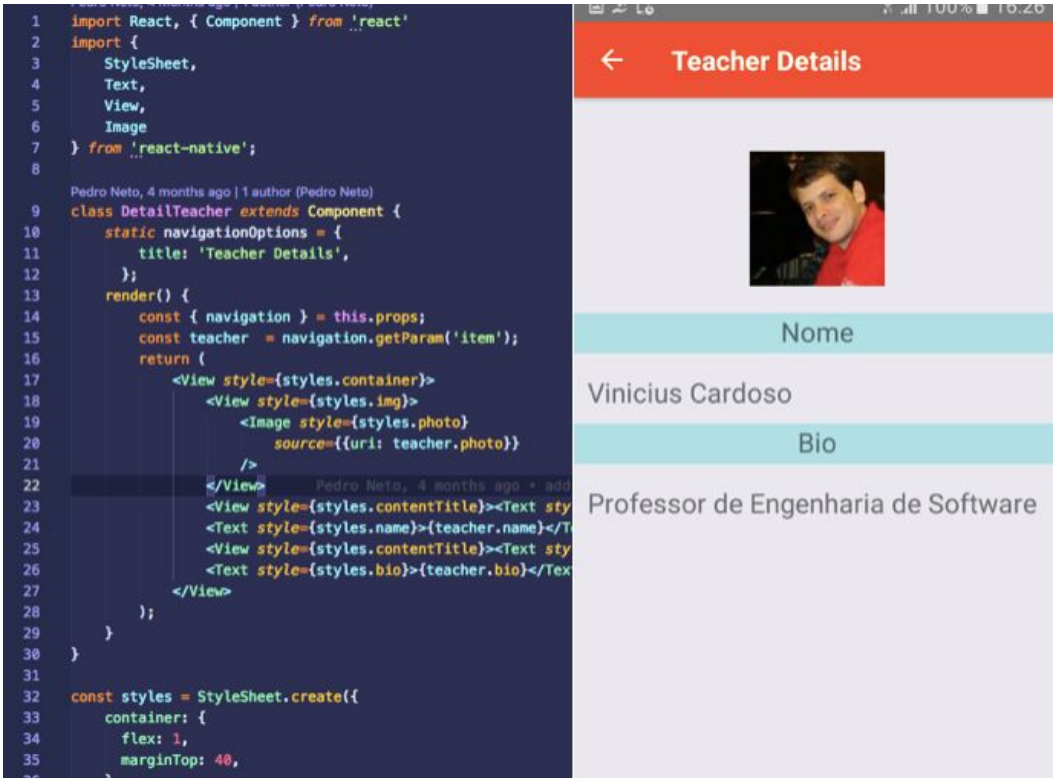

Figura 3.4: Código e tela do detalhe de professor

3.4 Sumário

Neste capítulo foi apresentado a justificativa da tecnologia adotada, a implementação dos testes e o desenvolvimento do aplicativo em React Native que será utilizado na pipeline DevOps.

A implementação do aplicativo foi importante para sentir as dificuldades de um desenvolvedor React Native com o gerenciamento de versões de dependências, compreender o motivo pelo qual React é o framework JavaScript mais utilizado atualmente e entender as dificuldades e limitações das ferramentas disponíveis para testar esse tipo de aplicação.

O próximo capítulo apresenta como foi o passo a passo do processo de construção da pipeline de integração e entrega contínua.

4. A PIPELINE DE INTEGRAÇÃO E ENTREGA CONTÍNUA

Rafal Leszko [15] apresenta o conceito de pipeline como: "uma sequência de operações automatizadas que geralmente representa uma parte da entrega de software e o processo de garantia de qualidade".

O sucesso da implantação de uma pipeline de integração e entrega contínua não depende exclusivamente das ferramentas que vão ser utilizadas, mas principalmente da estrutura da organização, equipe de desenvolvimento e as suas práticas.

A pipeline pode variar de acordo com o aplicativo em questão, no sentido de possuir mais ou menos passos para atender a necessidade do mesmo. A Figura 4.1 apresenta a pipeline para o aplicativo desenvolvido neste estudo.

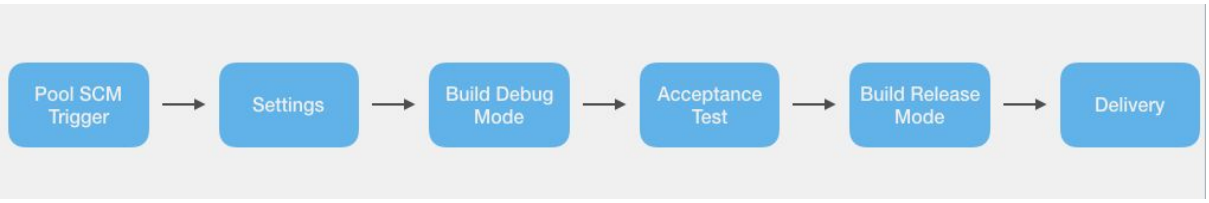

Figura 4.1: Pipeline desenvolvida para este estudo

4.1 Configuração de ambiente

No processo de implementação de uma pipeline de integração e entrega contínua é necessário fazer uso de ferramentas para dar suporte ao controle de versionamento de código, automação de *build*, automação de testes etc. Neste estudo utilizamos o Github[17] como ferramenta para versionamento de código, Jenkins na versão 2.138.2 como servidor de integração contínua e o Gradle[18] para realização do *build*.

4.1.1 Jenkins

Na configuração do servidor Jenkins foram instalados os plugins *Blue Ocean* na versão 1.9.0, *Slack Notification Plugin* na versão 2.3, *TestFairy Plugin* na versão 4.16

[17] https://github.com/
[18] https://gradle.org/

e *Github Integration Plugin* na versão 0.2.4 para auxiliar na integração e comunicação entre o Jenkins e as demais aplicações utilizadas no desenvolvimento da pipeline.

4.2 Estrutura da pipeline

Ao lidar com pipeline no Jenkins é preciso ter em mente dois elementos: etapa e passo.

- Passo: É a operação mais básica, uma tarefa única onde se diz o que o Jenkins vai fazer. Por exemplo: executar um *script*.
- Etapa: É um conjunto de passos, dentro do mesmo contexto, que serão executados de maneira sequencial ou paralela. Por exemplo: sequência de *scripts* de configuração.

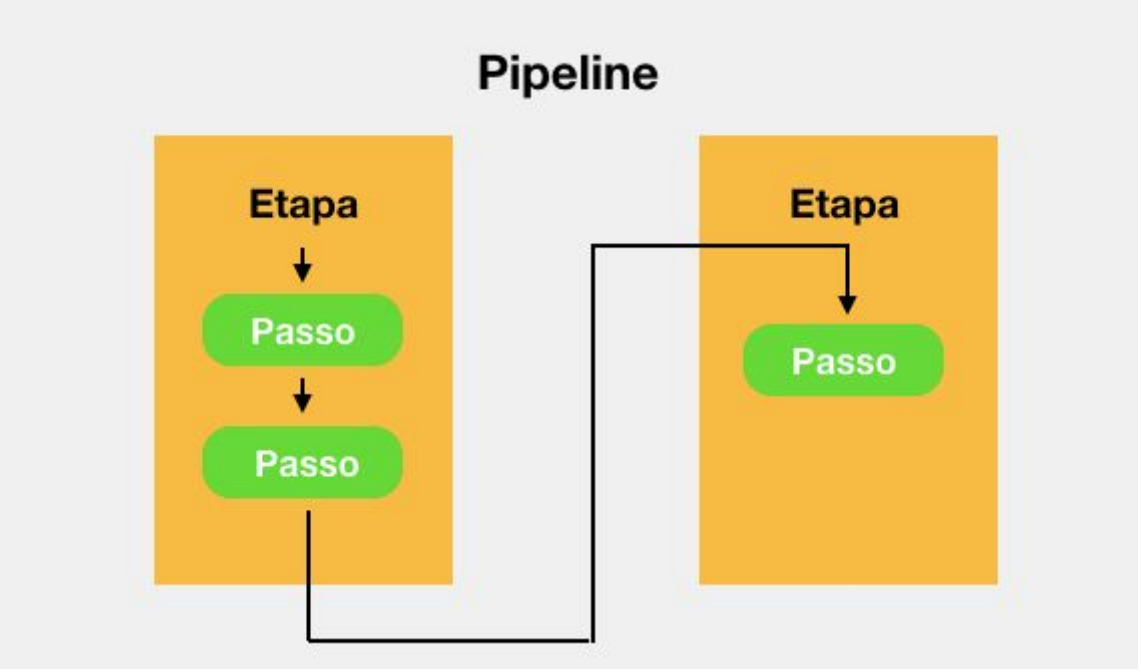

Figura 4.2: Estrutura de pipeline

4.2.1 Jenkinsfile

Para construção de pipeline no Jenkins é necessário escrever um arquivo de texto chamado de Jenkinsfile. Neste arquivo contém a declaração de todas as etapas e passos presentes na sua pipeline.

O Jenkinsfile fica na raiz do seu projeto sendo considerado como parte do desenvolvimento. Esta prática deu origem ao termo "pipeline-as-code" que trata a pipeline de integração e entrega contínua como parte do aplicativo que precisa ser versionado e revisado.

4.2.2 Poll SCM Trigger

A primeira etapa é responsável por dar início a execução da pipeline. Ela é acionada automaticamente, através de *cron job*[19] que executa a cada doze horas de segunda a sexta verificando a existência de alterações no repositório. Caso haja de novos *commits* no repositório, o processo de pipeline é iniciado.

O bloco de código a seguir apresenta a parte do Jenkinsfile responsável por esta etapa.

```
.........
triggers {
   pollSCM('H */12 * * 1-5')
 }
}
```

4.2.3 Settings

A etapa de Settings contém duas sub-etapas chamadas de BUNDLE INSTALL e NPM INSTALL. Esta etapa é responsável pelo gerenciamento das dependências do projeto. Durante a execução da pipeline as duas sub-etapas são executadas em paralelo.

A sub-etapa de BUNDLE INSTALL garante que todas as dependências necessárias para testar a aplicação estejam devidamente instaladas e versionadas. Enquanto a sub-etapa de NPM INSTALL realiza a instalação de todas as dependência, devidamente versionadas, que a aplicação necessita para funcionar corretamente.

[19] https://en.wikipedia.org/wiki/Cron

O bloco de código a seguir apresenta a parte do Jenkinsfile responsável por esta etapa.

```
.........
stage('SETTINGS') {
     parallel {
       stage('BUNDLE INSTALL') {
         steps {
           sh '''#!/usr/local/bin/zsh
             source $HOME/.zshrc
             source "$HOME/.rvm/scripts/rvm"
             bundle install'''
         }
       }
       stage('NPM INSTALL') {
         steps {
           sh '''#!/usr/local/bin/zsh
             source $HOME/.zshrc
             cd TeachersProject/
             npm install'''
         }
       }
     }
   }
.........
```

4.2.4 Build Debug Mode

Esta etapa é responsável por criar o build no modo de debug, neste modo, por exemplo, é possível visualizar todos os erros e alertas gerados pela aplicação em tempo de execução.

Ainda nesta etapa, um arquivo apk[20], Android Application Package, é gerado sendo possível executar os cenários de testes na próxima etapa da pipeline. Para esse passo executamos um *script* que aciona o Gradle, ferramenta responsável pelo *build* da aplicação conforme apresentado na Seção 4.1. Se alguma coisa der errado, a execução da pipeline é interrompida apresentando visualmente em qual passo o erro aconteceu, como ilustrado na Figura 4.3, caso contrário, seguirá para a próxima etapa.

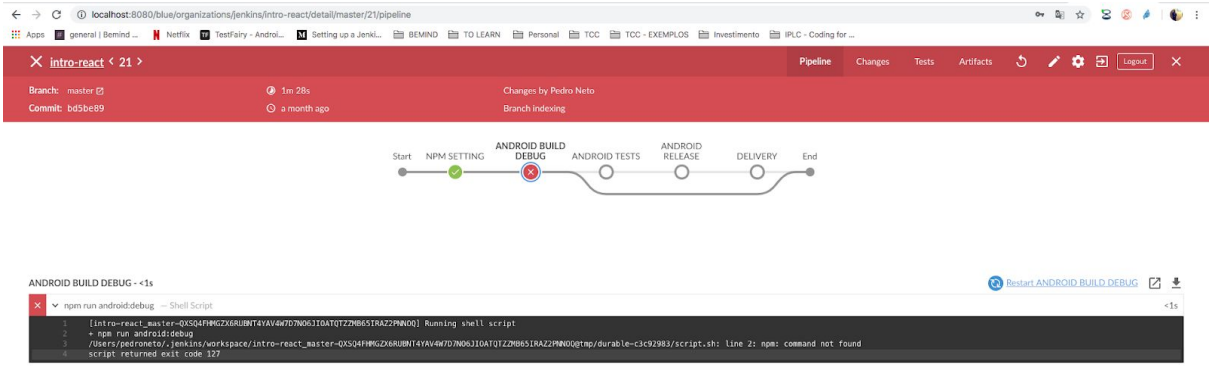

Figura 4.3: Erro durante a execução da pipeline

O bloco de código a seguir apresenta a parte do Jenkinsfile responsável por esta etapa.

```
.........
stage('ANDROID BUILD DEBUG') {
     steps {
       sh '''#!/usr/local/bin/zsh
         source $HOME/.zshrc
         cd TeachersProject
         npm run android:debug'''
     }
   }
.........
```

[20] https://en.wikipedia.org/wiki/Android_application_package

4.2.5 Acceptance Test

A etapa de Acceptance Test é a responsável por executar os testes do aplicativo, apresentados na Seção 3.2. Esta etapa é importante, pois é ela que vai garantir a qualidade do *software* e decidirá se o aplicativo está pronto ou não para ser entregue para os usuários. Se acontecer erro em algum dos testes, a execução da pipeline é interrompida, caso contrário, o aplicativo seguirá para próxima etapa.

Para esta etapa obter sucesso, existe a necessidade de possuir um dispositivo Android, configurado em modo de desenvolvimento, conectado via *usb* na máquina ou a utilização de um simulador Android. O aplicativo será instalado automaticamente no dispositivo e os testes serão executados no mesmo.

O bloco de código a seguir apresenta a parte do Jenkinsfile responsável por esta etapa.

```
.........
stage('ANDROID TESTS') {
     steps {
       sh '''#!/usr/local/bin/zsh
         source $HOME/.zshrc
         source "$HOME/.rvm/scripts/rvm"
         calabash-android resign
TeachersProject/android/app/build/outputs/apk/app-debug.apk
         cd TeachersProject
         npm run android:tests'''
     }
   }
.........
```

4.2.6 Build Release Mode

Esta etapa tem a função de preparar o aplicativo para ser enviado para Google Play Store[21]. O Android exige que todos os aplicativos sejam assinados digitalmente com um certificado antes de poderem ser distribuídos e instalados.

[21] https://play.google.com/store

Para assinar digitalmente é necessário seguir a documentação oficial da plataforma[22] e adicionar o certificado gerado ao seu código, como pode ser visto na Figura 4.4. Dado que o certificado já foi devidamente inserido ao código, agora é possível gerar um build no modo de release.

```
/**
 * Run Proguard to shrink the Java bytecode in release builds.
 */
def enableProguardInReleaseBuilds = false

android {
    compileSdkVersion 26
    buildToolsVersion "26.0.3"

    defaultConfig {
        applicationId "com.teachersproject"
        minSdkVersion 16
        targetSdkVersion 26
        versionCode 1
        versionName "1.0"
        ndk {
            abiFilters "armeabi-v7a", "x86"
        }
    }
    signingConfigs {
        release {
            if (project.hasProperty('MYAPP_RELEASE_STORE_FILE')) {
                storeFile file(MYAPP_RELEASE_STORE_FILE)
                storePassword MYAPP_RELEASE_STORE_PASSWORD
                keyAlias MYAPP_RELEASE_KEY_ALIAS
                keyPassword MYAPP_RELEASE_KEY_PASSWORD
            }
        }
    }
    splits {
        abi {
            reset()
            enable enableSeparateBuildPerCPUArchitecture
            universalApk false  // If true, also generate a universal AP
            include "armeabi-v7a", "x86"
        }
    }
    buildTypes {
        release {
            minifyEnabled enableProguardInReleaseBuilds
            proguardFiles getDefaultProguardFile("proguard-android.txt")
            signingConfig signingConfigs.release
        }
    }
```

[22] https://facebook.github.io/react-native/docs/signed-apk-android

Figura 4.4: Assinatura do aplicativo para distribuição na Google Play

Para gerar o build no modo de release, executamos um *script* que aciona novamente o Gradle. Se ocorrer erro durante a execução deste passo, a pipeline é interrompida, caso contrário, seguirá para a próxima etapa.

O bloco de código a seguir apresenta a parte do Jenkinsfile responsável por esta etapa.

```
.........
stage('ANDROID RELEASE') {
     steps {
       sh '''#!/usr/local/bin/zsh
         source $HOME/.zshrc
         cd TeachersProject
         npm run android:release'''
     }
   }
.........
```

4.2.7 Delivery

A última etapa é responsável basicamente por uma tarefa: notificação. No nosso caso, esta etapa notifica dois públicos distintos: o time de desenvolvimento e grupo de usuários que receberam o aplicativo para testar.

O time de desenvolvimento recebe o aviso através de um bot do Jenkins em um canal no Slack indicando que a execução da pipeline foi finalizada, conforme a Figura 4.5.

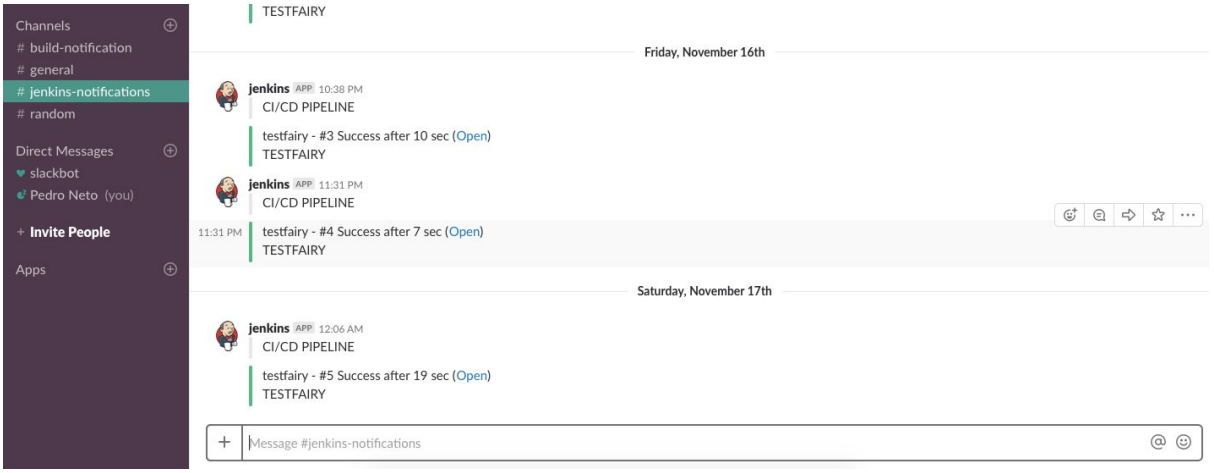

Figura 4.5: Notificação para o time de desenvolvimento

Por sua vez, o grupo de usuários recebe o aviso através de um email contendo os devidos links para realização do download do aplicativo, conforme a Figura 4.6.

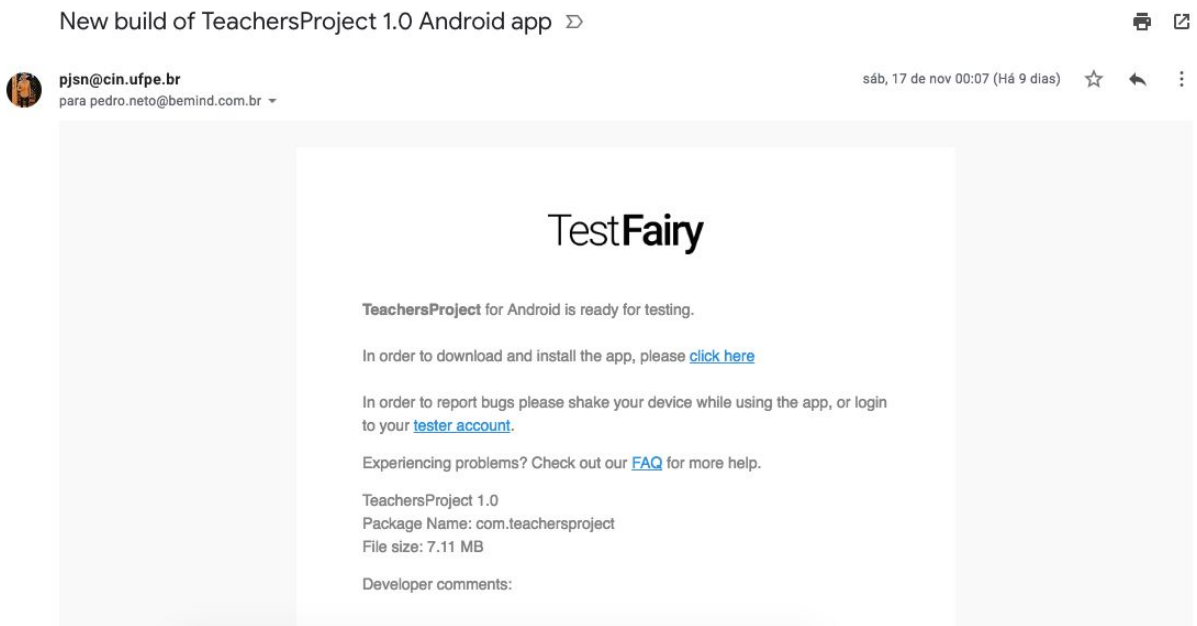

Figura 4.6: Notificação para o grupo de usuários

O bloco de código a seguir apresenta a parte do Jenkinsfile responsável por esta etapa.

```
.........
stage('DELIVERY') {
     steps {
       slackSend(baseUrl:
'https://pedrocompany.slack.com/services/hooks/jenkins-ci/', channel:
'jenkins-notifications', token: '2LpDh91ZUFjnvG6NdwYHAQsx', message: 'CI/CD
PIPELINE')
       build 'testfairy'
     }
   }
 }
.........
```

4.3 Discussão

Neste capítulo foi apresentado o conceito, a estrutura e a pipeline desenvolvida para o estudo. Bem como as ferramentas utilizadas para dar suporte a integração contínua, ao versionamento de código, ao build e a entrega do aplicativo. É apresentado o detalhe de cada etapa da pipeline explicando suas responsabilidades e funcionamento.

A etapa que apresentou maior dificuldade de ser implementada foi a de Acceptance Tests, pelo fato da mesma ser responsável por garantir a qualidade do software que será entregue para os usuários e talvez seja um dos principais motivos para quebrar o processo de automação do pipeline por completo. A Figura 4.7 apresenta todas as etapas da pipeline após a execução.

O próximo capítulo apresenta as conclusões, limitações e os trabalhos futuros necessários para cobrir toda a fase de Desenvolvimento da Engenharia de Software Contínua.

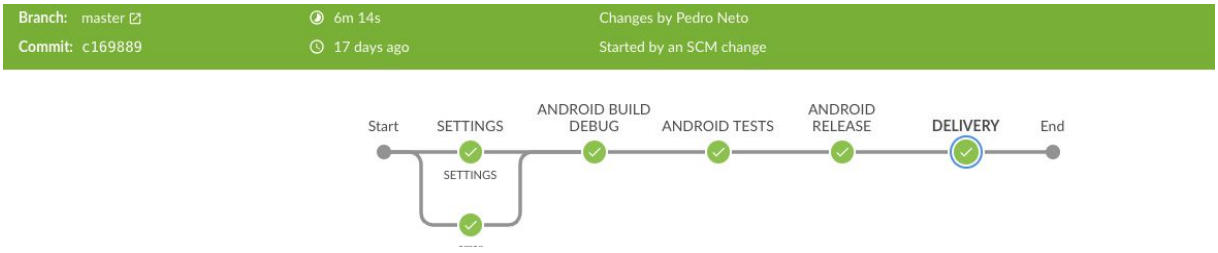

Figura 4.7: Todas as etapas da pipeline após a execução.

5. CONCLUSÕES

A construção de uma pipeline fornece um mecanismo para organização do processo de desenvolvimento de software como um todo, automatizando build e a maneira como o software é entregue para o cliente. Com a popularização no uso de integração e entrega contínua, a construção de pipelines estará presente no cotidiano de toda empresa de desenvolvimento nos próximos anos.

Com o desenvolvimento deste trabalho, foi possível provar com meios práticos e utilização de ferramentas testadas no mercado e amplamente conhecidas no ambiente organizacional que é tangível a utilização de integração e entrega contínua, altamente difundida no desenvolvimento web, agregando assim todos os benefícios já conhecidos atrelados a utilização desta prática, para o ambiente de desenvolvimento de aplicativos móveis.

5.1 Dificuldades e lições aprendidas

Neste estudo a maior dificuldade encontrada foi que o React Native possui um conjunto limitado de ferramentas para realizar testes. Calabash, a ferramenta escolhida, possui uma pequena comunidade de usuários o que dificulta para encontrar soluções de possíveis problemas e apresenta dificuldades para inspecionar e identificar os elementos na tela, para contornar esse problema é necessário adicionar a propriedade do React Native *accessibilityLabel*, que é uma propriedade que torna os aplicativos acessíveis para pessoas com deficiências indicando que o elemento está visível para ser manipulado. Esta propriedade auxilia na busca por elementos pelas ferramentas de testes, além de ser um boa prática de desenvolvimento.

5.2 Trabalhos futuros e limitações

Neste estudo não foi realizado a implementação de testes unitários e não foi aplicado uma ferramenta de Test Coverage, na qual se exige uma porcentagem

mínima de código coberto por teste para um commit ser aceito e dar início a execução da pipeline. Ficamos limitado ao ambiente Android, não foi possível incluir o processo de build para iOS, bem como não foi implementado o passo da implantação contínua responsável, no contexto de aplicação móvel, por publicar o aplicativo nas lojas.

Como principais atividades futuras esta implementação dos testes unitários utilizando Jest[23], ferramenta utilizada para testar aplicações desenvolvidas em React, integrar ao pipeline a ferramenta Fastlane[24] para auxiliar na etapa de build da aplicação para o ambiente iOS e no processo de publicação tanto na Google Play Store quanto na Apple Store. Com essas atividades será possível cobrir toda a fase de Desenvolvimento da Engenharia de Software Contínua.

REFERÊNCIAS

[1] FACEBOOK. React Native

[2] AMAZON. What is devops?

[3] "Continuous integration", http://en.wikipedia.org/wiki/ Continuous-Integration

[4] "Continuous integration", Martin Fowler, http://www.martinfowler.com/articles/continuousIntegration.html

[5] B. Fitzgerald, K.-J. Stol. Continuous software engineering: a roadmap and agenda. J. Syst. Software, 123 (2017).

[6] Marko Leppanen, Simo Makinen, Max Pagels, Veli-Pekka Eloranta, Juha Itkonen, Mika V Mantyla, Tomi Mannisto. The highways and country roads to continuous deployment.IEEE Software, 2015.

[7] M. Shahin, M. A. Babar, L. Zhu, "Continuous Integration Delivery and Deployment: A Systematic Review on Approaches Tools Challenges and Practices", IEEE Access, vol. 5, pp. 3909-3943, 2017.

[8] J. Humble, D. Farley, Continuous Delivery: Reliable Software Releases through Build Test and Deployment Automation, Addison-Wesley, 2010.

[23] https://jestjs.io/

[24] https://fastlane.tools/

[9] I. Weber, S. Nepal, L. Zhu, "Developing dependable and secure cloud applications", IEEE Internet Comput., vol. 20, no. 3, pp. 74-79, May/Jun. 2016.
[10] Mojtaba Shahin, Muhammad Ali Babar, Mansooreh Zahedi, Liming Zhu, "Beyond Continuous Delivery: An Empirical Investigation of Continuous Deployment Challenges", Empirical Software Engineering and Measurement (ESEM) 2017 ACM/IEEE International Symposium on, pp. 111-120, 2017
[11] L. Chen, "Continuous delivery: Huge benefits but challenges too", IEEE Softw., vol. 32, no. 2, pp. 50-54, Mar. 2015.
[12] STACK OVERFLOW. Developer Survey 2018
[13] STACK OVERFLOW. Developer Survey 2016
[14] Marick, Brian. Agile Software Development. Disponível em <http://codegardener.com/tag/Brian%20Marick >. Acesso em: 12 nov. 2018.
[15] LESZKO, Rafal. "Continuous Delivery with Docker and Jenkins."
[16] Bass, Len. "DevOps: A Software Architect's Perspective."
[17] J. Bosch, J. Bosch, "Continuous software engineering: An introduction" in Continuous Software Engineering, New York, NY, USA:Springer, pp. 3-13, 2014.
[18] J. Humble, Continuous Delivery vs Continuous Deployment, Mar. 2016, [online] Available:https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/.
[19] Jacksman, Marie. Continuous Integration Tools for Mobile vs Web. What’s the Difference?. Disponível em <https://code-maze.com/ci-tools-for-mobile-apps/>. Acesso em: 15 dez. 2018.