UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

BSI Sistemas de Informação UFRPE

Construção de uma solução para automatização de processos manuais de um assistente virtual

---

Relatório Técnico relativo ao Trabalho de Conclusão Curso do Bacharelado em Sistemas de Informação na modalidade Empresa

---

Aluno
Thales Gabriel dos Anjos Araujo

Orientador
Victor Medeiros
Departamento de Estatística e Informática

13 de maio de 2023

Thales Gabriel dos Anjos Araujo

Construção de uma solução para automatização de processos manuais de um assistente virtual

Relatório Técnico apresentado ao Curso de Bacharelado em Sistemas de Informação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada em: 27 de Abril de 2023.

Orientador: Victor Medeiros

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Estatística e Informática

Curso de Bacharelado em Sistemas de Informação

Banca: Gabriel Alves

Departamento de Estatística e Informática

Universidade Federal Rural de Pernambuco – UFRPE

Recife

13 de maio de 2023

Resumo

DevOps é uma abordagem de desenvolvimento de *software* que destaca a colaboração, comunicação e integração entre os desenvolvedores de *software* e os profissionais de operações de TI, com o objetivo de melhorar a velocidade, qualidade e confiabilidade da entrega de *software* valendo-se de práticas como a automação de processos que pode ajudar as empresas a reduzir custos, melhorar a eficiência, colaboração e satisfação do cliente. Uma vertente desta automação de processos é a orquestração, que auxilia no gerenciamento da implantação, coordenação e manipulação de diferentes partes de uma aplicação ou sistema. Envolve o uso de ferramentas e *frameworks* para simplificar todo o processo, automatizando tarefas, gerenciando dependências e reduzindo o risco de erros ou inconsistências. Os benefícios da orquestração podem ser observados em todas as empresas que adotam estas práticas para tornar seus processos mais eficientes, mesmo em situações com necessidades e contextos específicos. Um exemplo é a implantação de atualizações em seus sistemas, um processo comum e essencial para as empresas chamado de *deploy*, o objetivo dessa implantação é lançar uma nova versão de *software* ou atualização em um ambiente de produção, onde os usuários finais podem acessar e usar a aplicação. A implantação envolve uma série de etapas que podem variar entre as empresas, mas em todas inclui a preparação do código, teste da nova versão e a liberação para o ambiente de produção. Percebe-se que hoje o uso de ferramentas de automação vem se tornando cada vez mais comum, substituindo processos manuais a partir da implantação da cultura de *devops* nas empresas. Este projeto detalha a construção de uma solução que utiliza desenvolvimento de *software* aliado com a cultura *devops* para orquestração dos processos manuais de um assistente virtual. Ao término do desenvolvimento, o sistema alcançou o resultado desejado, abstraindo a execução de processos de forma automatizada e eliminando a necessidade do usuário de passar por todos os passos para realização do armazenamento de componentes ou implantação dos mesmos no assistente virtual, removendo erros ou inconsistências e tornando o desenvolvimento mais prático resultando em economia de recursos.

Sumário

1 Introdução

Uma solução de IA envolve um agrupamento de várias tecnologias, como redes neurais artificiais, algoritmos, sistemas de aprendizado, entre outros que conseguem simular capacidades humanas ligadas à inteligência. Por exemplo, o raciocínio, a percepção de ambiente e a habilidade de análise para a tomada de decisão (TOTVS, 2022). Tudo isso é necessário para resolver uma série de problemas, indo da grande complexidade da indústria ao corriqueiro cotidiano do homem moderno.

Para que a IA se desevolvesse foi necessário um passo antes: O processamento de linguagem natural (PLN) é uma tecnologia de *machine learning* que oferece aos computadores a capacidade de interpretar, manipular e compreender a linguagem humana. Isto é possível graças a combinação de técnicas da linguística computacional, inteligência artificial e ciência da computação (AMAZON, 2022).

Em essência, o PLN envolve o desenvolvimento de algoritmos e modelos que permitem aos computadores entender a estrutura e o significado da linguagem humana, bem como o contexto em que é usada, permitindo que aprendam padrões e façam previsões sobre como palavras, frases e sentenças provavelmente serão usadas em diferentes contextos.

Com todo esse aparato podemos dizer que a IA traz inúmeros benefícios a quem a usa entre eles:

- Redução do erro humano: Já que a partir de uma codificação correta padronizam e evitam erros ocasionais.
- Economia de recursos: Seja em tempo, dinheiro ou mão de obra;
- Assistência digital: Que interagem com os usuários eliminando a necessidade de recursos humanos.

Os *Chatbots* sintetizam bem o que foi escrito acima. Os *chatbots* permitem que as empresas atendam os clientes 24 horas por dia por meio de um sistema de comunicação automatizada (ZENDESK, 2022). Isto é possível por meio de tecnologias de PLN, além de várias tarefas como: Suporte ao cliente, agendamento de compromissos, resposta a perguntas frequentes ou reserva de serviços.

Apesar dos benefícios providos possuem certas limitações como:

- Falta de compreensão: Podem ter dificuldade em entender o que o usuário está tentando dizer, especialmente se a frase for complexa ou mal formulada.
- Capacidade limitada de resposta: *Chatbots* com base em regras têm respostas pré-programadas e não podem fornecer informações fora do escopo de suas regras. Eles podem não ter a capacidade de lidar com situações imprevistas ou fora do padrão.
- Falta de personalização: Os *chatbots* podem ter dificuldade em personalizar as respostas com base nas preferências e histórico do usuário, o que pode levar a respostas genéricas e pouco úteis.

Foi a partir desse contexto que o *IBM Watson Assistant*, também conhecido como Watson, se destacou de seus similares, pois é capaz de compreender clientes em diferentes contextos para fornecer respostas rápidas, precisas e consistentes em qualquer aplicativo, dispositivo ou canal.

Com muitas empresas e organizações mudando para operações remotas e online, principalmente durante a pandemia da covid-19, a necessidade de fornecer suporte ao cliente e serviços automatizados aumentou drasticamente o que acelerou ainda mais as necessidades de utilização de IA, de acordo com a pesquisa da *Global AI Adoption Index, 2022*, principalmente o foco no atendimento ao cliente.

Neste mesmo período de forte presença empresarial online houve também o aumento na demanda por profissionais da tecnologia. No Brasil, desde o início da pandemia da covid-19, houve 85 mil novas vagas. Para algumas funções, em São Paulo, a procura por profissionais em 2020 cresceu mais de 600%. **CNN**, São Paulo, 27 de outubro de 2021. Disponível em: cnnbrasil.com.

Diante deste contexto novos desafios surgiram para as grandes empresas de tecnologia, por exemplo: Muitas equipes trabalhando nas mesmas ferramentas utilizando processos diferentes, ou seja uma série de atividades executadas dentro de uma organização, transformando entradas em saídas com o propósito de atingir um objetivo específico, seja ele um produto ou serviço(Lisboa, 2018). trouxeram problemas de comunicação entre seus colaboradores o que impactou diretamente na eficiência de suas operações. Para lidar com isso foi, e continua sendo, necessária uma excelente comunicação, padronização e regras de governança para atingirem um objetivo em comum.

1.1 Problema abordado

Antes de comentar sobre o problema que deu origem a elaboração da solução é importante definir de forma mais específica como o Watson funciona. Ele é composto por:

- Habilidades;
- Nós de diálogo;
- Intenções;
- Entidades.

As habilidades são componentes mais gerais e tem a função principal de agrupar os nós de diálogo, elas também podem delimitar contextos segregando responsabilidades. Um exemplo seria definir uma habilidade relacionada a cartões de crédito, dentro dessa habilidade podemos construir diversos fluxos conversacionais que são representados por nós de diálogo como uma árvore binária que será exemplificada na figura 1.

Os nós de diálogo são representados por retângulos de cor branca. Podem ou não possuir um nome ou uma condição de entrada. Quando um usuário digita um texto e manda para o Watson o texto será interpretado a fim de identificar as intenções, representadas por uma # seguido do nome, e as entidades, representadas por uma @ seguido do nome. Quando a condição de determinado

nó de diálogo é positiva um conjunto de ações pode ser tomado a exemplo uma inserção no contexto de uma variável. Um conjunto de nós de diálogo encadeados pode ser chamado de fluxo de conversação.

Fonte: Elaborada pelo autor

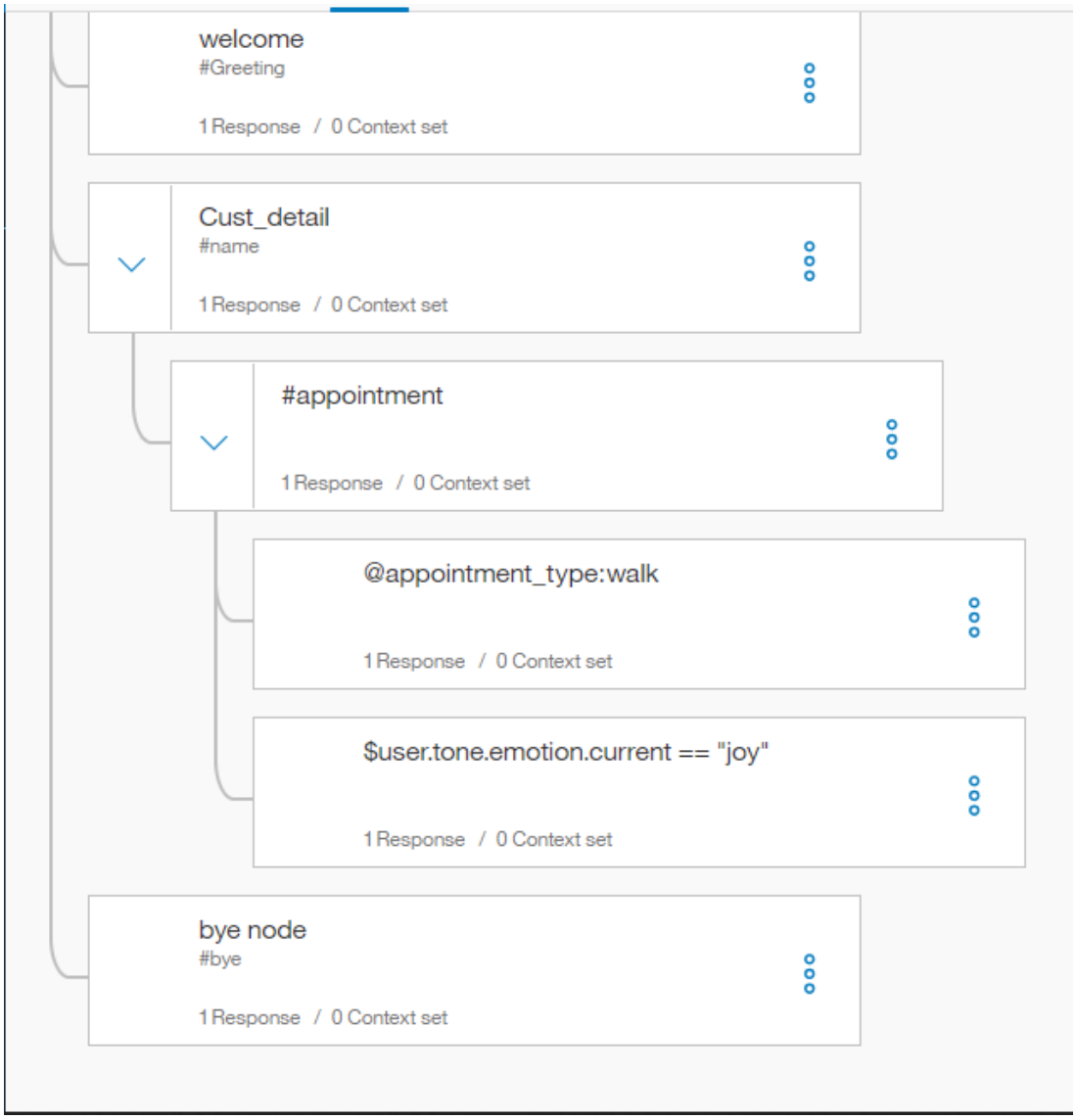

Figura 1: **Captura de tela que representa a estrutura em árvore mencionada do IBM Watson.**

Cada habilidade pode ser representada em um formato textual através de um JSON, essa que é uma estrutura para transmissão de dados baseada no conceito chave-valor. É desta forma que consegue-se armazenar as versões das habilidades e implantá-las em novos ambientes do Watson. Um ponto negativo é que o Watson, em sua primeira versão, limita esse armazenamento de habilidades em dez vezes, o que não é ideal já que ao longo do dia muitas modificações podem ser feitas ultrapassando o limite pré-estabelecido fazendo com que seja necessário abordar novas táticas para conseguir expandir o número de vezes em que pode-se armazenar as habilidades para contar com um histórico maior. É interessante ressaltar que este mecanismo de armazenamento é bastante simples, não contando com uma forma de identificar diferenças entre as versões, quem fez determinadas alterações, quando foram feitas, além de outras métricas de auditoria que são importantes para resolução de problemas e também do gerenciamento dos times em questão

Outro ponto importante é que não há a possibilidade de uma implantação de melhorias ou correções de forma automatizadas sendo necessário fazer o *download* das habilidades, armazenamento manual, e *upload* das habilidades no ambiente desejado para aplicar as modificações. Como este processo de implantação não é bem definido e a comunicação entre os times responsáveis por gerir o watson não é adequada tem-se problemas de inconsistências já que não se sabe onde cada equipe guarda os JSONs dessas habilidades. Esta falta de transparência no processo de implantação traz riscos como: Não saber se a versão que se encontra armazenada em determinado lugar de fato é a última mais recente.

A segunda versão do Watson é capaz de resolver o problema da falta de um versionamento eficiente, mas continua não possibilitando a implantação automatizada dos *JSONs* das habilidades entre diferentes ambientes, Entretanto a forma de precificação é diferente. As despesas não seriam interessante para empresa chegando a custar o dobro do que seria a primeira versão. Desta forma se tornou mais interessante investir no desenvolvimento de uma solução interna capaz de solucionar os problemas relacionados a utilização de processos manuais aliados a baixa comunicação entre as equipes gerando mais praticidade na implantação de melhorias e correções, menos possibilidade de erros durante as entregas de versões e segurança contra eventuais problemas já que se teriam habilidades recentes e funcionais armazenadas caso uma versão em ambiente produtivo apresentasse problemas.

Fonte: Elaborada pelo autor

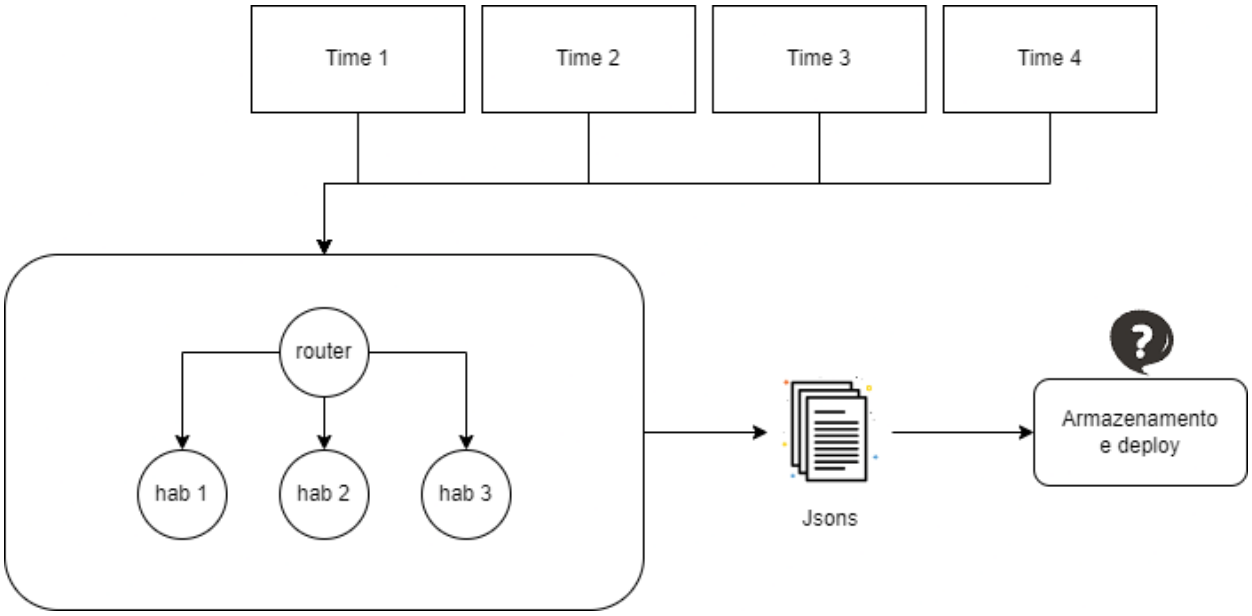

Figura 2: **Captura de tela que representa o funcionamento dos processos de armazenamento e implantação de forma manual.**

A figura 2 representa o funcionamento dos processos de armazenamento e implantação de forma manual que ocorriam antes da solução ser desenvolvida. Percebe-se que todos os times fazem o uso do Watson representado pela figura no formato de quadrado de maior lado, o produto são os JSONs das habilidades que serão armazenadas e implantadas, nota-se também a presença de um sinal de interrogação que representa a dúvida relacionada a como os processos de fato serão realizados por cada equipe.

1.2 Proposta de solução

Considerando os pontos expostos anteriormente houve uma reunião para desenhar um plano de ação que pudesse suprí-los. Inicialmente foram detalhados processos que serviriam como padrão a ser utilizado por todos os times envolvidos na manipulação das JSONs das habilidades do Watson. Entretanto isto não seria suficiente tendo em vista que a comunicação entre os times não é adequada para que se garanta que os processos estejam sendo seguidos, dessa forma a automatização se torna um recurso eficiente tendo em vista que abstrai todos os passos dos processos de armazenamento e implantação dando mais garantias de que tudo está sendo executado conforme esperado.

A solução mencionada conta com o auxílio de uma biblioteca, disponível em várias linguagens de programação, com uma documentação compreensível e pública, disponibilizada pela IBM. Esta biblioteca é a mesma que fornece a possibilidade de interação entre o usuário e o Watson então não demandou nenhum estudo a mais do que já se tinha. Para o armazenamento das habilidades, os participantes debateram a respeito de diversas ferramentas que já se fazem presentes no dia a dia de trabalho dos mesmos, o que é uma vantagem na questão de custos tendo em vista que não seria necessário contratar nenhum serviço novo para a solução em questão.

Como as habilidades podem ser representadas de forma textual foi pensado em uma ferramenta chamada git, um sistema de gerenciamento de versões de dados não estruturados que fornece esse suporte. Desta forma, além de se conseguir armazenar as habilidades no formato textual, consegue-se ver pontos de destaque referentes ao que mudou de uma versão para outra, quem fez a modificação, quando foi feita e diversas outras métricas interessantes que servem como base de auditoria e mantém um longo histórico de alterações.

Para implementação das resposabilidades de armazenamento e implantação se pensou na criação de uma aplicação responsável por interagir com a biblioteca fornecida pela IBM, através de código, de forma tal que se pudesse baixar os JSONs das habilidades de um ambiente específico e implantar em outro ambiente. Já para realizar o registro das habilidades no repositório do projeto os participantes decidiram que essa deveria ser uma responsabilidade da própria esteira do gitlab. A esteira de processamento é composta por diversos passos que definem um processo como: Construção de um arquivo executável, execução dos testes unitários da aplicação, execução da aplicação em si e principalmente a utilização de comandos do git para o armazenamento das habilidades no repositório em questão.

Os participantes então pensaram como a solução final seria executada tendo em vista não é um sistema que necessita estar funcionando constantemente já que a demanda é interna e possui um público alvo reduzido Para isso pensaram em criar uma solução que se aproximasse de um *command line interface (CLI)*, ou seja, a aplicação seria executada por meio de comandos reconhecidos pelo terminal de seus computadores. Porém não seria o suficiente já que os usuários finais poderiam executar a solução ao mesmo tempo e causar alguma inconsistência, então era necessário limitar o uso.

Para que este limite seja colocado de eficaz era necessário centralizar a solução. Isto fez com que fosse cogitado que o sistema fosse *RESTful*, uma interface que dois sistemas de computador usam para trocar informações de forma segura pela internet. Mas isso iria totalmente contra a questão de

manter os custos baixos, já que não tinhamos acesso a muitas formas de fazer a aplicação ser executada sem que fosse no cluster fornecido pela própria empresa e isso traria um custo desnecessário. A titulo de comparação cada instância presente no cluster custa cerca de 20 dólares.

Além de que traria uma desvantagem relacionada a como seria visto que essa solução estaria sendo utilizada? Haveria a necessidade de desenvolver uma interface amigável para o usuário final, que trouxesse a visão de informações básicas como o histórico de execuções, quem executou e o que foi executado. Teria-se então mais esse custo, fora todo esforço que haveria para elaboração em relação ao valor agregado da solução, o que não era interessante para os participantes ou para as lideranças deles, o que rapidamente fez com que descartassem a idéia.

Foi nesse momento em que foi atribuída mais uma responsabilidade a esteira do gitlab que seria a execução do projeto em si. Neste ponto os participantes já tinham uma proposta simples de como a solução poderia funcionar em relação ao armazenamento e a implantação permitindo que ela pudesse ser utilizada apenas quando necessário, entregando o valor pretendido com o mínimo de recursos e sem causar inconsistências durante a execução, mas ainda tinham que entender como realizar o armazenamento diário e automático. Para isso os participantes decidiram criar uma tarefa cronológica que automaticamente executaria uma função dado um horário pré-definido.

Para construção da tarefa cronológica foi utilizado o kubernetes já que a empresa disponibiliza de um cluster. Neste momento não havia muita discussão tendo em vista que os participantes já tinham um conhecimento prévio relacionado a utilização dessa ferramenta então não foram consideradas outras opções de execução de forma automática.

Um problema que acabou surgindo foi o fato de que nem todos os times possuem acesso a ferramenta conhecida como Gitlab e sendo assim não teriam a autonomia para executar a solução. Foi sugerida uma solução que é utilizada por todas as equipes, o Jira, uma ferramenta de controle de tarefas que é capaz de criá-las de forma intuitiva e personalizada de forma a auxiliar os colaboradores no desenvolvimento dessas atividades, ferramenta essa na qual seria utilizado o modo de automação para se comunicar com a esteira do projeto e assim dar a possibilidade de implantação a quem não possui acesso ao Gitlab.

De forma bem rudimentar e genérica a utilização da biblioteca fornecida pela IBM e a utilização do git para versionamento foram os componentes fundamentais para elaboração do produto de mínimo valor para que pudesse solucionar os problemas apresentados e aproveitado por todos os times que tem de alguma forma que interagir com melhorias dos fluxos no Watson padronizando e automatizando um processo importante do ciclo de vida de desenvolvimento de software o que na visão de Alexander Bartie, 2002, permite que um número maior de defeitos sejam descobertos antecipadamente a sua implantação em produção

1.3 Motivação

A solução foi desenvolvida dentro da equipe responsável principalmente por manter a consistência e criar novos fluxos no watson que realizam integração com APIs externas. Tendo nascido como uma forma de otimizar processos e custos facilitando, dessa forma, o dia a dia da empresa tendo em vista que a utilização de processos manuais se tornou uma dor entre as equipes diretamente envolvidas com o Watson.

Foi uma iniciativa para melhorar as consequências da utilização de processos mal utilizados dos times envolvidos na manipulação do Watson, fornecer uma opção de armazenamento das habilidades para momentos de emergência e facilitar o processo de implantação de atualizações nessas habilidades em ambiente produtivo, ou seja o ambiente utilizado pelos clientes. Também foi uma forma que os envolvidos encontraram para dar apoio ao momento da empresa de instigar a eficiência operacional entregando o mesmo valor, ou até mais, com o mínimo de recursos possível.

Após o desenho inicial da elaboração da solução e posterior apresentação aos demais membros da empresa em questão ocorreu um erro que acarretou a inicialização imediata do projeto detalhado pelo documento. Tem-se em mente que as habilidades de produção são defasadas em relação as habilidades em homologação tendo em vista que o desenvolvimento de novas funcionalidades e correções acontece neste último ambiente, dessa forma o erro que ocorreu foi a substituição das versões de homologação das habilidades pelas versões de produção ocasionando em perda de trabalho de todos os times. A solução para evitar este problema seria baixar manualmente cada habilidade em formato de *JSON* e savá-la em algum lugar para se prevenir de problemas como esse.

Previamente ao fato ocorrido já estava sendo investigada a primeira forma de automatização desses processos em um script simples. Em vez de cada habilidade ser baixada individualmente e salva em algum repositório foi utilizado este script desenvolvido para baixar todas as habilidades de uma vez e realizar o armazenamento desses *JSONs* em um repositório na nuvem. Apesar de ser manual esta foi a primeira vez que utilizamos uma automação como parte do fluxo de trabalho diário o que foi visto com louvor e priorizado o processo de implantação daquela solução.

A partir deste acontecimento um novo atributo de qualidade foi adicionado no projeto que foi o armazenamento diário de todas as habilidades de forma a não permitir a repetição do erro, ou seja, ao menos a versão do dia inicial já estaria salva para que todos pudessem ter esse respaudo. Este armazenamento diário também serviria para evitar que os participantes tivessem que ficar utilizando o script mencionado anteriormente para fazer o *download* das habilidades.

De forma a intensificar a elaboração da versão funcional da solução as dicussões acerca do tema passaram a ter mais frequência até que uma possível solução foi construída visando um produto mínimo viável, que pudesse ser implantado na infraestrutura da empresa, de forma barata, rápida e que pudesse ser utilizada por todos os times envolvidos na manipulação do Watson. A estratégia escolhida se mostrou mais adequada para a natureza das situações exploradas visto que todas as ferramentas que seriam utilizadas já eram conhecidas, o que evitou gastos desnecessários com novas ferramentas e possibilitou que todas as equipes pudessem ter uma participação efetiva no processo garantindo a autonomia de cada uma.

1.4 Objetivo

O objetivo deste trabalho foi criar uma automação que soluciona a limitação padrão do Watson relacionado ao armazenamento das habilidades, podendo este armazenamento ser diário e automático. Também contempla a implantação das atualizações das habilidades em ambiente produtivo, que é o utilizado na empresa para entregar seu produto aos usuários finais, bem como desfazer essa implantação. Este sistema final deve ser auditável por interesse da equipe de infraestrutura e segurança da empresa a partir da geração de relatórios de utilização.

1.4.1 Objetivos específicos

- Desenvolvimento de um sistema que abstraia toda lógica de armazenamento e implantação dos fluxos;
- Manipulação de uma interface responsável pela execução da solução;
- Armazenamento das versões dos fluxos conversacionais;
- Armazenamento diário de forma automatizada.

1.5 Participação na implementação do sistema

O sistema foi desenvolvido por uma equipe composta por profissionais de engenharia de *software*, *DevOps*, processos e experiência do cliente. A concepção da idéia foi realizada pelos engenheiros de *software* com experiência na diagramação de soluções, as pipelines do gitlab ci tiveram o auxílio do time de *DevOps* e a equipe de processos foi responsável pela criação do usabilidade da solução de forma comum entre todos os times responsáveis pela manipulação do watson. O time de experiência do cliente foi responsável pelos testes e alterações no fluxo.

Com fins de detalhar a visão em que este documento foi escrito me envolvi especificamente na construção da arquitetura de *software* e no desenvolvimento dos serviços do sistema. Durante a implementação da arquitetura, houve colaboração no desenvolvimento de quase todos os complementos, contribuindo com a implementação de pelo menos uma funcionalidade.

Para que o sistema funcionasse apenas quando solicitado houve o desenvolvimento da pipeline no Gitlab CI. O desenvolvimento dessa fase ocorreu após o estudo de como essa ferramenta funciona aplicado no contexto da empresa. Posteriormente a implementação do código faria a solução funcionar em conjunto com a equipe de *DevOps*.

Para executar a solução final o usuário deve utilizar as *pipelines* do gitlab dentro do repositório do projeto em questão e preencher os campos ilustrados na figura 3:

Fonte: Elaborada pelo autor

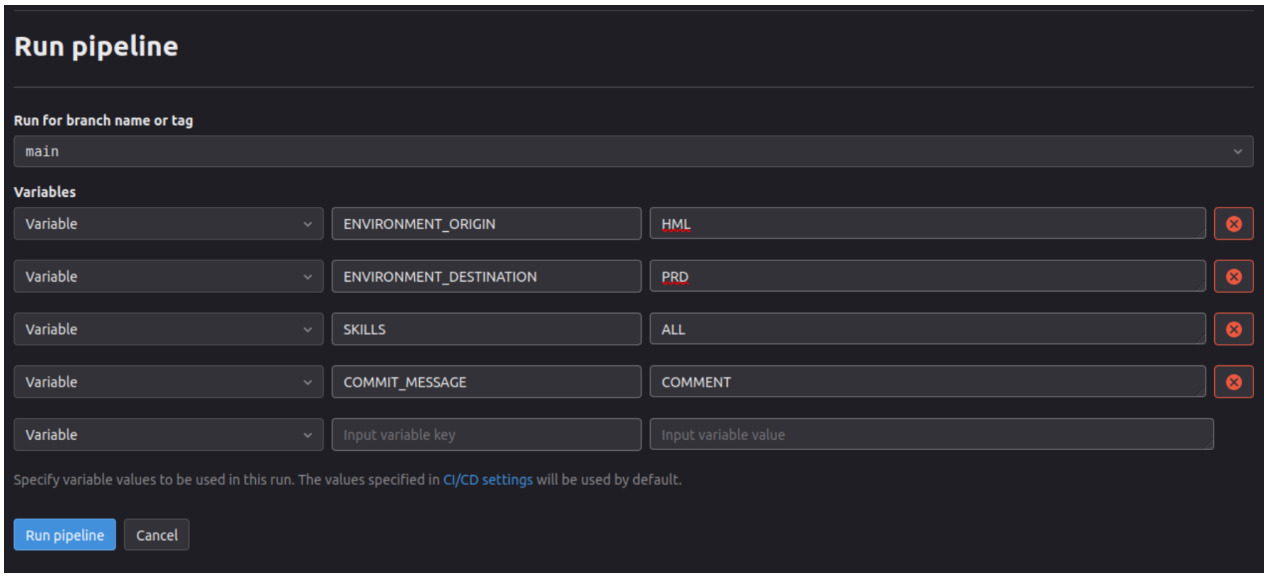

Figura 3: **Captura de tela encontrada representando como as variáveis que o sistema precisa são resgatadas pela pipeline**

A figura 3 representa como o usuário vê a tela inicial da *pipeline* no Gitlab do projeto. Nestes campos o usuário passa essas variáveis que serão consideradas durante a execução do projeto. De forma específica as intenções das pipelines são:

- Resgatar informações de cada execução da pipeline como: Ambiente de origem e destino, as habilidades que serão implantadas no ambiente de destino e a mensagem descritiva a respeito da motivação daquela implantação;
- Realizar procedimentos relacionados a qualidade de código e testes automatizados;
- Executar o sistema com as variáveis propostas pelo usuário.

Uma variável que é preenchida durante o processamento da *pipeline* é a versão da implantação. Isto é importante porque na descrição da habilidade no ambiente produtivo nós teremos exatamente qual é a versão atual, característica importante para que se possa fazer a busca no repositório do projeto a respeito de mais informações a cerca da versão em questão como: Quem fez, o que foi alterado, quando foi a implantação e a tarefa que gerou a implantação. É importante ressaltar que a figura 3 é mais específica para implantação de atualizações nas habilidades. Para utilizar apenas o armazenamento das habilidades é necessário apenas o ambiente de origem de onde as habilidades, em formato *JSON*, serão baixadas.

Para que a *pipeline* pudesse ser construída foi preciso seguir uma série de boas práticas construídas pelo time de *DevOps* para que se possa manter dentro dos padrões de qualidade estabelecidos pela empresa. Dessa forma foi preciso haver uma comunicação frequente, e incomum, com esse time para que fosse possível avançar no processo. Entre as boas práticas existe um repositório específico que deveria ser construído a pipeline, este que serve como um centralizador de código para que a manutenção se torne mais fácil.

Antes do projeto de fato ter sido colocado em prática um *script*, escrito na linguagem python, foi construído contemplando todos os objetivos finais do projeto a ser desenvolvido. Esta ação foi importante devido ao erro mencionado anteriormente no qual as versões das habilidates foram apagadas gerando em um prejuízo para os membros dos times. Este *script* foi importante já que ele ajudou a construir a lógica do sistema final o que deu mais agilidade na implantação da solução e também garantiu uma certa segurança aos membros do time de que o erro que gerou a deleção das habilidades fosse reparado caso se repetisse. Posteriormente o sistema foi refeito de forma estruturada, com boas práticas e uma melhor arquitetura.

Também foi necessário uma conversa constante entre diversas pessoas da empresa entre líderes, desenvolvedores, pessoas do time de negócio entre outras. E a motivação foi sempre esclarecer a importância do projeto e o porque era necessário atuar nele da forma que os participantes estavam o fazendo.

2 Referencial teórico

Para o melhor entendimento do problema e desenvolvimento da solução, alguns conceitos, ferramentas e técnicas foram utilizados. Eles serão apresentados a seguir de forma a detalhar como a solução funciona nos passos subsequentes. É importante notar que essa seção foi dividida em duas contemplando: Conceitos e Ferramentas

2.1 Conceitos

2.1.1 *Containers*

Container é, em português claro, o agrupamento de uma aplicação junto com suas dependências, que compartilham o kernel do sistema operacional do host, ou seja, da máquina (virtual ou física) onde está rodando. São bem similares às máquinas virtuais, porém mais leves e mais integrados ao sistema operacional da máquina host, uma vez que compartilha o seu kernel, o que proporciona melhor desempenho por conta do gerenciamento único dos recursos(Descomplicando o docker, 2018).

Os *containers* são projetados para resolver a dificuldade de gerenciar aplicativos de *software* complexos com inúmeras dependências, onde as alterações em uma parte do sistema podem ter efeitos inesperados em outras partes. Ao encapsular um aplicativo e suas dependências em um *container*, os desenvolvedores podem garantir que o aplicativo seja executado consistentemente independentemente do sistema.

Também oferecem outros benefícios para o desenvolvimento, como ciclos de desenvolvimento mais rápidos, colaboração aprimorada e maior escalabilidade, tornando-os uma ferramenta popular no desenvolvimento de *software*, especialmente com o aumento das arquiteturas de microsserviços e a necessidade de opções de implantação mais flexíveis e escaláveis.

2.1.2 API

Uma API (Interface de Programação de Aplicativos) é um conjunto de regras, protocolos e ferramentas para a construção de aplicativos de *software*. Essencialmente, uma API especifica como diferentes componentes ou serviços de *software* devem interagir entre si, tornando mais fácil para os desenvolvedores a construção de aplicativos complexos, aproveitando serviços e recursos existentes. Conectam soluções e serviços, sem a necessidade de saber como esses elementos foram implementados (REDHAT, 2022).

As APIs podem assumir muitas formas diferentes, mas geralmente são usadas para permitir a comunicação entre diferentes componentes ou serviços de *software*. Por exemplo, uma API pode permitir que um aplicativo móvel recupere dados de um servidor web ou interaja com um serviço baseado em nuvem. As APIs são geralmente acessadas por meio de uma rede, usando protocolos padronizados como HTTP ou HTTPS.

2.1.3 Integração contínua (CI)

Na visão de Martin Fowler 2006, é uma prática de desenvolvimento de software que visa melhorar a colaboração e reduzir problemas de integração dentro de uma equipe de desenvolvimento. É um método de desenvolvimento de *software* em que os desenvolvedores, regularmente, submetem suas alterações a um repositório de códigos de forma a juntá-las. Após isso são aplicadas diversas automações responsáveis por garantir a qualidade do código que os desenvolvedores estão entregando. Estas automações variam de empresa para empresa, mas em geral é importante que sejam verificados:

- Os testes unitários e de integração daquele sistema;
- A qualidade do código com ferramentas como Sonarqube;
- Vulnerabilidades e afins com ferramentas tais quais o Veracode
- O *build* que a aplicação gera;
- E o restante de steps que a empresa julgar necessário.

Este método é importante quando comparamos o processo feito antes dele. No passado, o desenvolvimento era feito por diversas equipes de forma isolada até que decidissem juntar suas alterações em uma versão nova. O problema é que muito código estocado tornava o processo de junção difícil, demorado, custoso e manual o que sem uma comunicação ineficiente se tornava impossível de manter.

A comunicação excessiva em um ambiente sem integração contínua pode se tornar uma tarefa de sincronização complexa e intricada, que acrescenta custos burocráticos desnecessários ao projeto. Assim as versões de códigos ficam mais lentas e com maiores taxas de falha, pois requer que os desenvolvedores sejam sensíveis e cuidadosos com relação às integrações. Esses riscos crescem muito conforme o tamanho da equipe de engenharia e da base de código aumenta (ATLASSIAN, 2022).

A Integração Contínua (CI), na visão de Martin Fowler 2006, é uma prática de desenvolvimento de software que visa melhorar a colaboração e reduzir problemas de integração dentro de uma equipe de desenvolvimento. Por meio da CI, os desenvolvedores integram frequentemente as mudanças de código em um repositório compartilhado, permitindo a detecção precoce de problemas de integração, feedback rápido sobre a qualidade do código, maior colaboração entre os desenvolvedores, melhoria da estabilidade do software e preparação para a entrega contínua. Através da automação de testes e integração, a CI proporciona uma abordagem ágil e eficiente no ciclo de desenvolvimento, ajudando a garantir um software de qualidade e confiável em todas as etapas do processo.

2.1.4 Implantação contínua (CD)

Como o próprio nome sugere, pode ser pensado como a extensão da integração contínua responsável por automatizar a entrega propriamente dita. Seu maior objetivo é minimizar o tempo que o código colocado na *branch* principal leva para ser realmente aproveitado pelos clientes em produção.

Na figura 4 temos um exemplo de uma pipeline que une tanto a integração quanto a implantação contínua. Fica bem evidente que cada passo pode ter uma denominação própria, mas em termos gerais nós aplicamos termos apenas a integração que une todos os passos da automação e a implantação que é o *deploy* propriamente dito.

Fonte: Elaborada pelo autor, 2022.

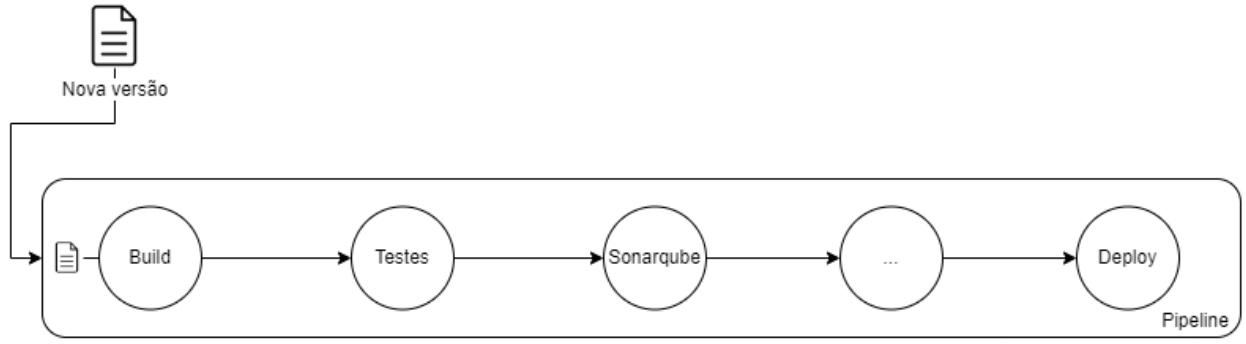

Figura 4: **Diagrama representando um exemplo de pipeline**

Note que essas fases são mais lógicas do que físicas e são criadas apenas para quebrar um problema grande em vários subproblemas menores. Você pode ter menos ou mais fases, dependendo de sua arquitetura e de seus requisitos.(ATLASSIAN, 2022).

2.1.5 *DevOps*

DevOps é uma abordagem colaborativa que visa integrar as equipes de desenvolvimento (Dev) e operações (Ops) para melhorar a eficiência, agilidade e qualidade dos processos de desenvolvimento e entrega de software. A metodologia DevOps enfatiza a automação, a comunicação e a colaboração contínua entre os desenvolvedores de software e os profissionais de operações, visando superar as barreiras tradicionais que existem entre essas equipes. Segundo o artigo DevOps Literature Review de 2014, É uma grande mudança no desenvolvimento de Sistemas de Informação porque reduz

a lacuna entre desenvolvedores, operações e o usuário final, permitindo a detecção precoce de problemas de forma mais eficiente e eficaz.

2.1.6 *Frameworks*

São estruturas que permitem, auxiliam e tornam mais prático o processo de desenvolvimento de sistemas e aplicações. Funcionam como uma espécie de molde que quando utilizados oferecem benefícios estruturais para construção da solução. São comumente utilizados pelas ”bibliotecas”que ele possui. Essas bibliotecas são outros projetos já construídos pela própria comunidade que tem um objetivo específico como economizar código e são integradas ao projeto em desenvolvimento.

O principal objetivo dos *frameworks* é resolver problemas recorrentes com uma abordagem genérica. Com isso, o desenvolvedor não precisa ficar reescrevendo *softwares*, podendo focar seus esforços em resolver os problemas em si. Pode-se dizer que no desenvolvimento além de precisar de muita criatividade, existem muitas tarefas mecânicas e repetitivas. Portanto, é aí que entra o *framework*, como uma espécie de automação ou um conjunto de códigos que já foram testados e que estão disponíveis para facilitar a vida dos programadores (HOSTGATOR, 2020).

2.2 Ferramentas

2.2.1 *Spring Framework*

Spring é um *framework* Java criado com o objetivo de facilitar o desenvolvimento de aplicações, explorando, para isso, os conceitos de Inversão de Controle e Injeção de Dependências. Dessa forma, ao adotá-lo, tem-se à disposição uma tecnologia que fornece não apenas recursos necessários à grande parte das aplicações, como módulos para persistência de dados, integração, segurança, testes, desenvolvimento *web*, como também um conceito a seguir que permite a criação soluções menos acopladas, mais coesas e, consequentemente, mais fáceis de compreender e manter.

Já o *Spring Boot* é um desses recursos do *Spring* que visa simplificar a configuração e a inicialização das aplicações. Fornece uma abordagem ”opinião sobre configuração”, em que padrões e convenções predefinidos são usados para configurar o aplicativo, minimizando a necessidade de configurações detalhadas.

2.2.2 *Docker*

O *Docker* é uma plataforma *open source* que facilita a criação e administração de ambientes isolados, os containers. Ele possibilita o empacotamento de uma aplicação ou ambiente dentro de um container, se tornando portátil para qualquer outro host que contenha o Docker instalado. Então, você consegue criar, implantar, copiar e migrar de um ambiente para outro com maior flexibilidade. A ideia do Docker é subir apenas uma máquina, ao invés de várias. E, nessa única máquina, você pode rodar várias aplicações sem que haja conflitos entre elas.(TREINAWEB, 2021).

O Docker Engine tornou-se um padrão do setor para o processo de *containers* com uma abordagem de embalagem universal e ferramentas de desenvolvedor simples. Ele não é estritamente necessário, mas com o auxílio das *Dockerfiles* é possível criar imagens personalizadas para que possemos referenciá-las e impantá-las em praticamente qualquer grande provedor hoje em dia distribuindo tudo de forma muito rápida aos clientes.

Além da utilização da técnica para o desenvolvimento também é importante devido a economia de recursos que este proporciona com conceito utilizado no docker denominado *Multi-stage building* na qual com auxílio de algumas palavras reservadas da própria ferramenta nós conseguimos extrair apenas os artefatos necessários de cada passo utilizado, dessa forma nós conseguimos gerar uma imagem final mais leve. Logo abaixo temos um exemplo utilizando a linguagem Go

```
1 package main
2
3 import "fmt"
4
5 func main () {
6   fmt.Println("Hello, world!")
7 }
```

Quando executado o código acima simplesmente retorna a frase dentro dos parêntesis, mas em contrapartida temos duas formas de construir a imagem referente a este código: Uma sem utilizar técnica alguma para economizar recursos e outra que utiliza o *Multi-stage build*.

- Sem *Multi-stage build*

```
1 FROM golang:1.19 AS build
2 WORKDIR /app
3 COPY . /app
4 RUN CGO_ENABLED=0 GOOS=linux go build -o api main.go
5 CMD [ "go", "run", "main.go" ]
```

- Com *Multi-stage build*

```
1 FROM golang:1.19 AS build
2 WORKDIR /app
3 COPY . /app
4 RUN CGO_ENABLED=0 GOOS=linux go build -o api main.go
5
6 FROM scratch
7 WORKDIR /app
8 COPY --from=build /app .
9 CMD ["/app"]
```

Com esta definição de uma *Dockerfile* nós precisamos apenas:

1. Buildar a imagem utilizando o comando ”docker build -t app .”;
2. Executar o container utilizando o comando ”docker run app”.

Dessa forma receberemos como saída o resultado esperado em ambas, mas como é possível ver na figura 5 a utilização de recursos é maior em um dos exemplos o que ilustra os ganhos que tivemos com relação a economia de recursos além de entendermos o quanto realmente é necessário para que nossa aplicação consiga operar de forma eficiente. Explicar esta situação também é importante para que possamos detalhar mais quando estivermos comentando sobre kubernetes.

Fonte: Elaborada pelo autor, 2022.

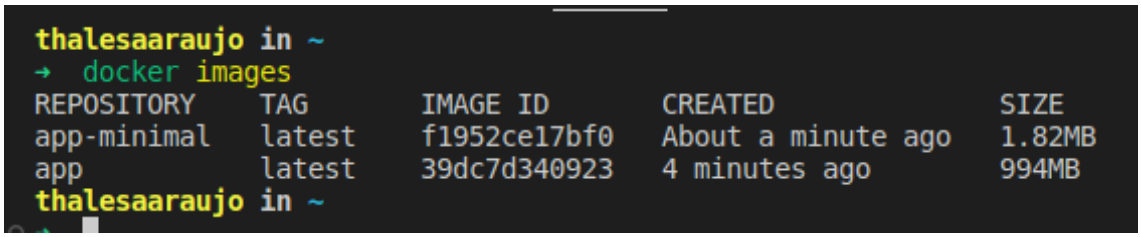

Figura 5: **Ilustração representando a diferença de tamanho entre as imagens com e sem MSB**

Esta prática nos ajudou a economizar reduzindo o tamanho das imagens de contêineres, o que requer menos espaço de armazenamento, resultando em economias significativas, especialmente para organizações que implantam muitos contêineres. Imagens de contêineres menores também resultam em tempos de implantação mais rápidos, reduzindo os custos de infraestrutura e melhorando a eficiência geral.

Ao otimizar os *Dockerfiles* para incluir apenas os arquivos e dependências necessários, o *multi-stage building* pode reduzir ainda mais o tamanho da imagem, melhorando o desempenho e reduzindo a utilização de recursos. Dessa forma conseguimos continuar implantando toda solução com o mínimo de custos. Essa questão foi bastante discutida por um dos participantes que já tinha mais experiência na empresa e já foi muito impactado por decisões que envolviam custos, inclusive ter que parar um outro projeto por conta da falta de economia.

2.2.3 Git e Gitlab

De longe, o sistema de controle de versão moderno mais usado no mundo hoje é o Git. É um **Sistema de Controle de Versões Distribuído** o qual simplifica muito o dia a dia de trabalho de projetos que envolvem mais de um programador. Estes sistemas registram as alterações feitas em cima de um código armazenando essas informações e permitindo que, caso seja necessário, um(a) programador(a) possa regredir a versões anteriores de uma aplicação de modo simples e rápido.

Já o Gitlab serve para facilitar o controle de versões de um software ou aplicação, ele armazena todos estes dados em uma nuvem e você pode acessá-los de qualquer lugar. Funciona também como uma ferramenta gráfica na qual você pode interagir da forma que quiser com o seu código e projeto.

O Git é baseado em alguns conceitos como **repositórios,** *commits, branches e merges* que interagem facilmente com ferramentas como o Gitlab, mas que funcionam, fundamentalmente via linha de comando. Um *commit* é um registro de mudanças feitas em um arquivo ou conjunto de arquivos. Este commit inclui uma mensagem que descreve as mudanças, bem como uma referência ao estado anterior do código, permitindo que outros desenvolvedores acompanhem as mudanças ao

longo do tempo e colaborem no projeto.

Esses códigos geralmente podem possuir ramificações, chamadas de *Branches*. Em um repositório a *branch* principal é chamada de main e a cada nova funcionalidade você pode criar mais *branches* que depois serão integradas a *branch* main, este processo é denominado de *Merge*. O *merge* quando criado não ocorre imediatamente, é criado um *Merge request* no qual os outros desenvolvedores podem visualizar as mudanças que um desenvolvedor fez.

Este último processo é denominado *code review* e após concordarem entre si se alterações são necessárias para ajustar o código que está sendo requisitado para integrar a branch principal finalmente ocorre o *merge*.

2.2.4 Interface de Linhas de Comando e Shell

Um command-line interface, ou CLI, processa os comandos que serão enviados para um programa de computador na forma de linhas de texto. Já o Shell é responsável por processar os comandos do CLI. O principal benefício da utilização de ambos, principalmente num projeto como esse, são os baixos custos de utilização.

A necessidade de seu uso veio ao percebermos que a ideia central do sistema é ser executado e finalizado, ou seja, de forma a evitar gastos desnecessários pensamos em criar um CLI para que sua utilização fosse tão simples quanto executar um comando de versionamento ou implantação.

2.2.5 Kubernetes

O Kubernetes é uma plataforma de código aberto que permite a automação de operações com contêineres, eliminando processos manuais de implantação, gerenciamento, escala e controle de recursos (HENRY, 2017). O Kubernetes permite agrupar múltiplas máquinas, físicas ou virtuais, para a execução de contêineres, ficando responsável por gerenciar todos os recursos. A este agrupamento damos o nome de *Cluster*.

Da mesma forma que o docker, o kubernetes também possui um linguajar próprio que é construído com auxílio dos manifestos com a extensão *yaml*. Cada manifesto pode referenciar um recurso em específico do kubernetes. Antes de ser apresentado um exemplo é importante deixar claro que existem inúmeros recursos, entre eles o *deployment* é nele onde se descreve um estado desejado em uma implantação e o controlador de implantação altera o estado real para o estado desejado em uma taxa controlada. (Kubernetes.io, 2022).

Ao final do processo de implantação do *deployment* no *cluster* desejado o resultado é a obtenção de um pod este que é puramente um *container*. A exemplo tem-se um manifesto que define um recurso do tipo *deployment* que executa o nginx, que em resumo é um servidor *web*.

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: nginx-deployment
5   labels:
6     app: nginx
```

```
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

O *cluster* kubernetes é essencial para exercer um maior controle de gerenciamento em infraestruturas com muitos serviços e todos operam em *containers*, dessa forma é imprescidível que se tenha essa ferramenta ao alcance para que se possa visualizar todas as aplicações bem como seus dados, se ainda estão sendo executadas, se estão com algum problema e afins.

A *cronjob* é outro recurso do kubernetes com características próprias. Em linhas gerais uma *cronjob* é capaz de criar uma tarefa que se repete sempre em determinado horário escolhido, tarefa esta que tem início, meio e fim. Este recurso funciona de forma similar ao deployment. A *cronjob* vai controlar quando os *jobs* que surgem serão executados.

Tal funcionamento é bastante importante visto que existem horários fixos de *deploy* e armazenamento das habilidades que ocorrem todos os dias de forma manual o que se torna interessante automatizar esses horários.

Uma parte importante do processo de criação dos *jobs* automáticos foi a compreensão das regras de criação de recursos no *cluster*. Foi observado que existem regras de segurança que impedem a implantação de um recurso de forma simples, é necessário utilizar uma ferramenta interna para seja possível criar os recursos no *cluster*. Recursos esses que eram transcritos como *deployments*.

Durante o desenvolvimento foi utilizada uma técnica para simular um *cluster* na máquina de desenvolvimento para que fosse possível testar os recursos no kubernetes sem necessidade colocá-los diretamente no *cluster* da empresa. Este processo foi realizado com o auxílio de uma ferramenta chamada *kind* que separa uma certa quantidade de recursos em sua máquina e simula um *cluster* local.

Logo abaixo está a descrição do manifesto escrito para criação da tarefa que executa sempre duas vezes ao dia em horários fixos, meia noite e meio dia, destinada a realizar o armazenamento automático das habilidades do watson:

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: automate-watson-job
spec:
  schedule: "12 * * * *"
  jobTemplate:
    spec:
```

```
    template:
      spec:
        containers:
        - name: automate-watson-job
          image: curlimages/curl
          imagePullPolicy: IfNotPresent
          command:
          - /bin/sh
          - -c
          - curl --request POST \
            --form token=<trigger-token> \
            --form ref=main \
            https://gitlab.example.com/api/v4/projects/1/trigger/pipeline
        restartPolicy: OnFailure
```

Percebe-se aqui que não foi utilizada a imagem da própria aplicação para a execução da solução. Um dos participantes notou que não era necessária já que tudo ocorreria dentro do próprio gitlab então criamos uma *cronjob* que utiliza uma requisição curl para iniciar a pipeline com as variáveis *default* do projeto.

No ambiente real ao final do processo teremos um *container* que utilizando uma simples imagem linux contendo o curl, que são destinados para funcionar como uma forma de verificar a conectividade da URL, irá disparar uma requisição via gitlab api que irá disparar a pipeline referente ao processo de versionamento nas duas vezes ao dia. Isto funcionará como um armazenamento automático.

3 Desenvolvimento realizado na empresa

Este capítulo visa detalhar a problemática abordada e a solução desenvolvida principalmente com relação a arquitetura construída. As tecnologias que já foram detalhadas serão citadas aqui. E também a escolha de cada tecnologia, método e abordagem. Os resultados do projeto também serão compartilhados bem como o impacto da contribuição.

3.1 A problemática e a solução proposta

Percebe-se que cada time interage com o Watson representado pelo retângulo arredondado da figura 6. Cada time é responsável pelo *JSON* da habilidade que gera, bem como onde irá guardá-lo representando bem a falta de transparência no processo e bem como quem realmente o fez já que só se poderia ver uma alteração feita depois que já estava em produção.Outro problema é a falta de percepção de onde estão as versões referentes a cada implantação feita, o que traz um problema na de recuperação contra desastres. Caso as habilidades de homologação fossem deletedas a forma de reaver o conteúdo perdido ainda sim seria problemático.

A solução proposta está diagramada na figura 7 dessa forma o processo de interação com as habilidades do watson bem como o processo de armazenamento dos jsons e por fim o deploy desses se encontra representado na figura. Isso traz uma grande transparência do processo pois garante como tudo está sendo feito além de possibilitar auditoria a partir de um relatório de quem fez, o que

Fonte: Elaborada pelo autor, 2022.

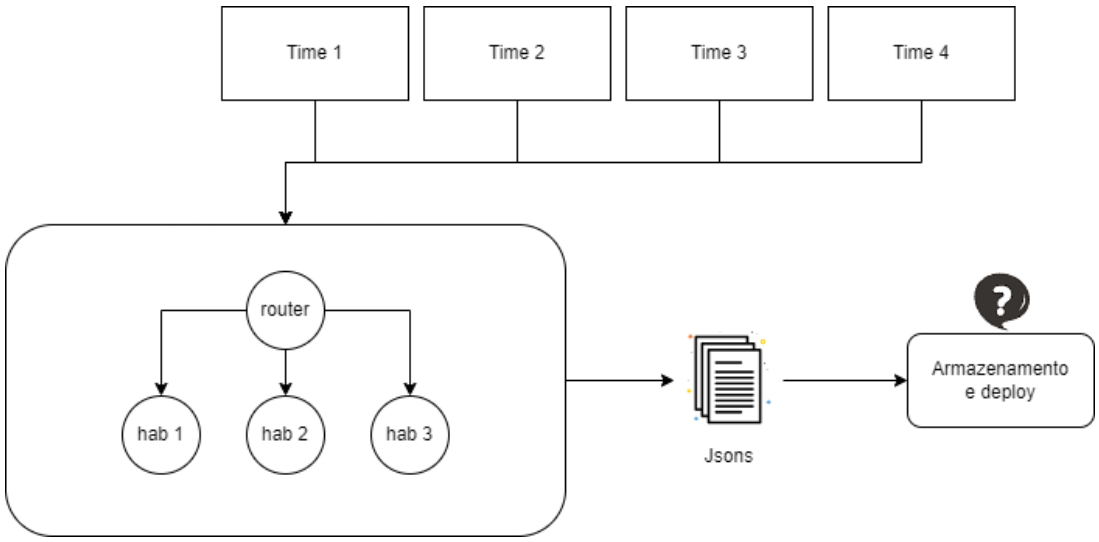

Figura 6: **Diagrama representando o processo de implantação e versionamento antigos.**

foi modificado e quando foi feito.

Fonte: Elaborada pelo autor, 2022.

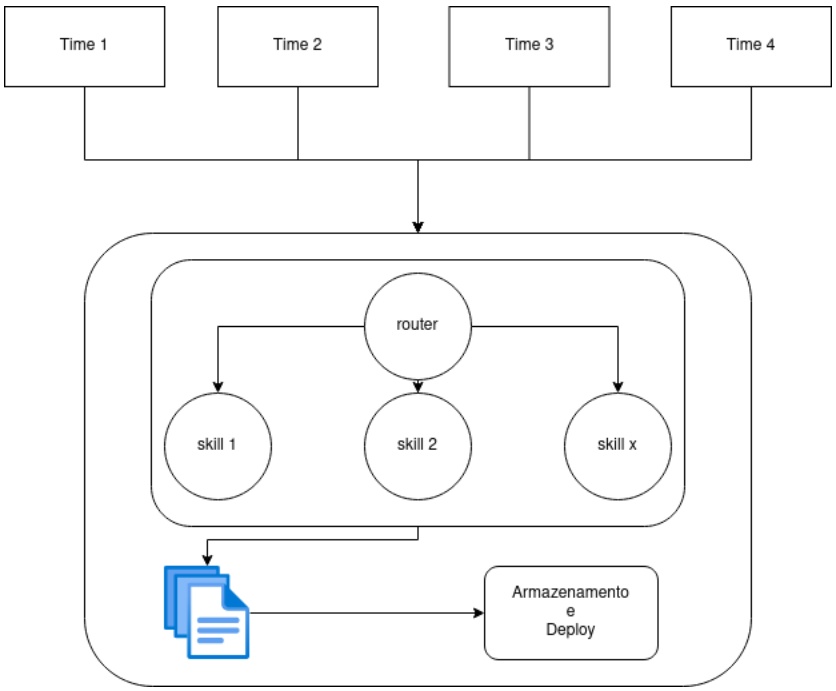

Figura 7: **Diagrama representando o processo de implantação e versionamento novos.**

3.2 Metodologia

Após a análise dos dados e entendimento do problema, resultando na proposta de solução, e priorização desta demanda na sprint dos participantes, a metodologia abordada para o desenvolvimento da arquitetura do sistema foi estruturada nas seis etapas descritas a seguir:

1. Estudo das características de infraestrutura e definição das tecnologias;
2. Desenho da arquitetura;
3. Validação da arquitetura proposta com os consultores de software;
4. Validação da escolha do ferramental proposto com os mesmos consultores;
5. Priorização da demanda;
6. Implementação dos serviços do sistema;
7. Testes de componente, integração e de sistema;
8. Compreensão da forma de implantação no ambiente produtivo;
9. Implantação do sistema em ambiente de produção.

O primeiro passo realizado foi relativamente simples a medida em que foi decidido utilizar uma parte do ferramental já conhecido de forma ampla na empresa a qual se trabalha dessa forma já se sabia de sua qualidade. Em se tratando das ferramentas que não conheciam um levantamento foi realizado através de pesquisas na internet. Através desse levantamento foram definidas as ferramentas e tecnologias utilizadas no desenvolvimento do sistema.

O segundo passo então foi desenhar como o sistema de fato iria funcionar iniciando então o passo de desenho da arquitetura que foi realizado pelo autor e depois validado pelos participantes e uma equipe de consultores de *software* presentes na própria empresa onde foi discutido o porque da solução ser a mais adequada no quesito custos e funcionamento, captado e absorvido *feedback*. A arquitetura será detalhada na seção 3.3.

A partir daí foi necessário esperar que houvesse tempo para que a demanda fosse priorizada e houvesse a estimativa do custo de tempo e esforço para o desenvolvimento, para que se desse início à fase de implementação. Na etapa de implementação foi iniciado o desenvolvimento do serviço, planejando a sequência de implementação com base na experiência dos participante sobre o que seria mais simples de ser desenvolvido e foi do armazenamento(*Backup*), implantação(*Deploy*) e desfazer a implantação (*Rollback*).

A etapa de testes do sistema foi realizada de forma manual e muito semelhante a um teste de integração onde era verificado se de fato as habilidades no formato *JSON* estavam sendo baixadas na etapa do versionamento e consequentemente armazenadas com auxílio das *pipelines* do gitlab. Após isso foi iniciada a automatização dos testes para garantir a consistência ao longo do tempo iniciando com os testes unitários e depois alguns poucos de integração já que estes são mais custosos.

Ao fim da implementação, foi realizada a etapa de teste de sistema, onde o versionamento foi utilizado como uma forma de armazenamento manual das habilidades acionando as *pipelines* do gitlab.

Após isso criada a *cronjob* para manter a execução automática e então criadas as funcionalidades de implantação automática e a ação de desfazer a implantação.

3.3 A abordagem da arquitetura de *software*

Para que a automação proposta fosse possível, foi necessário, além da concepção e do debate entre os participantes a implementação de uma arquitetura de *software* adequada o suficiente para que os objetivos propostos fossem alcançados. A arquitetura do sistema atende a quatro requisitos principais:

- Executada apenas quando solicitada;
- A execução ocorre considerando parametros;
- Atualização do ambiente produtivo a partir do ambiente de homologação;
- Possibilidade de construção de uma *cronjob*;

Para atender aos requisitos a solução é composta de serviços:

- Watson;
- Gitlab;
- Aplicação;
- Kubernetes cronjob.

O Watson é o ponto central da solução onde a API provida pela IBM é utilizada tanto para o resgate das habilidades quanto implantação das mesmas. É o maior motivador do esforço para elaboração da solução. O Gitlab além de executar a aplicação que contém a lógica de *backup*, *deploy* e *rollback* a partir dos parametros solicitados pelo usuário também é o responsável por armazenar cada versão das habilidades. Ainda no gitlab temos uma auditoria moderada na qual podemos ver quem iniciou a automação, o que foi alterado e quando foi alterado. A aplicação contém a lógica dos processos de automação e interage com a API provida pela IBM utilizando boas práticas de programação. Esta aplicação foi desenvolvida utilizando o Spring. O docker foi um ponto que não foi pontuado na composição da solução porque ele foi utilizado indiretamente. Inicialmente os participantes estavam gerando imagens da aplicação para que pudessem ser utilizadas pelas tarefas temporais(*cronjobs*), mas foi percebido que isso não era necessário pelo fato de conseguirmos utilizar a API do gitlab para iniciar a execução da aplicação então o docker acabou sendo utilizado indiretamente. Já o kubernetes auxiliou na construção da *cronjob* diária para que o *backup* pudesse ser realizado ao menos duas vezes por dia evitando assim erros humanos e inconsistências.

3.3.1 Automação Watson

A Automação Watson é um sistema que permite uma rápida interação com a API fornecida pela IBM com o seu assistente virtual auxiliando os desenvolvedores em tarefas do dia a dia de trabalho.

Este sistema é baseado em três características citadas anteriormente:

- *Backup*: Este é o processo de armazenamento das habilidades em um repositório dentro do Gitlab da empresa.
- *Deploy*: Este é o processo de implantação que envolve o *download* das habilidades dos ambientes de origem e destino, armazenamento em memória e *upload* no ambiente de destino.
- *Rollback*: Este é o processo que desfaz o último *deploy* fazendo com que a versão antes do *deploy* volte a ser a do ambiente de destino.

Para que toda essa automação proposta fosse implementada, foi necessário, além da idealização, a implementação de uma arquitetura de software contemplando os serviços necessários para que o objetivo geral fosse alcançado. A arquitetura do sistema atende a quatro requisitos principais:

- Todos os times conseguiram se beneficiar com o projeto;
- Poderemos abstrair o processo de versionamento e *deploy*;
- Pôde-se automatizar o *backup* diário e futuramente o *deploy*.

Fonte: Elaborada pelo autor, 2022.

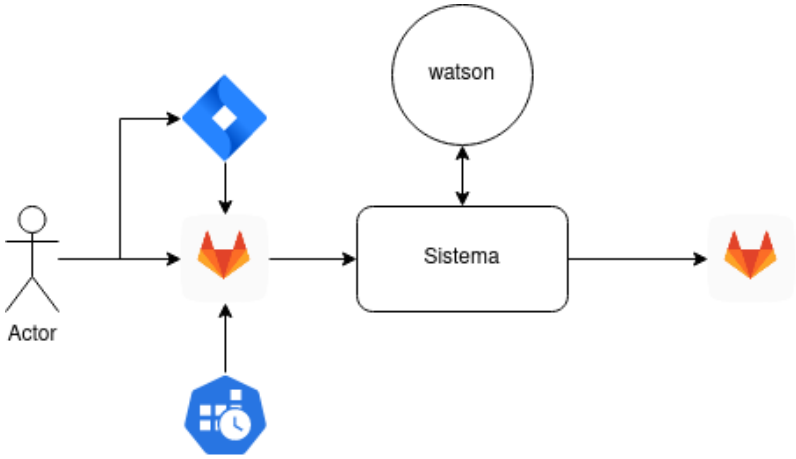

Figura 8: **Diagrama representando o processo de implantação e versionamento novos.**

Para atender a solução proposta foi necessário o desenvolvimento de apenas um serviço baseado no *Spring boot* contendo mais de uma responsabilidade. Também foi necessária a criação de *cronjobs* para automatizar o horário de armazenamento diário e futuramente realizar implantações de forma automática que já ocorrem hoje.

Como é um sistema relativamente simples nenhum conceito de monitoramento foi levado tão profundamente como o conceito de observabilidade, mas ainda sim é interessante que tenhamos um

relatório de auditoria do uso da solução que é enviado diretamente para o sistema de comunicação da empresa, o Slack, dessa forma conseguimos saber sem precisarmos acessar o cluster se a execução foi bem sucedida ou não. É importante lembrar também do *Jira automate* que tem a mesma função de executar a pipeline de forma remota porém por um sistema que é mais frequente no dia a dia dos participantes e também devido ao fato de um dos times não ter acesso ao gitlab. Desta forma todos os times conseguem utilizar e se beneficiar do projeto de abstração promovendo mais transparência no processo.

3.4 Contribuição e justificativa das escolhas

Este capítulo visa detalhar a contribuição do autor na solução desenvolvida bem como as escolhas que foram feitas de forma mais objetiva explicando as vantagens que foram privilegiadas em detrimento das desvantagens.

A linguagem Java é a de maior especialidade entre todos os participantes da solução proposta. Esta que é usada no dia a dia, o que passou a ser um fator determinante tendo em vista que precisávamos de agilidade no desenvolvimento e na manutenção. Como a solução tem um escopo fechado com pouco espaço de extensão não foram analisadas os prós e contras de outras linguagens tendo em vista as necessidades de implementação que tínhamos que também não são complexas. De forma análoga também foram utilizadas boas práticas que são comuns no dia a dia da empresa de forma a manter-se um padrão relacionado as aplicações já existentes de forma a facilitar a manutenção. De forma a contemplar a agilidade necessária para realização da solução foi decidido utilizar o *Spring*. Cada membro participante já tinha experiência com o *framework* java e chegaram a cogitar outros, porém o *Spring* é amplamente utilizado e fornece um suporte maior a bibliotecas diversas que ajudaria a completar a solução final.

Pela solução ter um escopo relativamente pequeno e simples construímos a solução em apenas uma aplicação que é condizente com uma maior facilidade de manutenção, desenvolvimento e implantação já que tem-se apenas uma base de código para gerenciar. Também é interessante comentar a respeito dos custos da arquitetura que foram baixos já que não foi utilizado nenhum serviço pago. Uma característica dessa solução é que não temos uma necessidade latente de escalabilidade já que é uma ferramenta interna com público alvo fixo e que varia pouco, o que traz mais um ponto positivo a respeito da utilização dessa estratégia para esse contexto. Outro ponto que confirmou a escolha da utilização de uma aplicação única em relação a serviços distribuídos é que para uma aplicação se comunicar com outra é necessário que ao menos uma delas esteja disponível a todo momento. De forma a evitar esses custos é mais interessante que o sistema possua as duas responsabilidades para que ela seja utilizada apenas no momento em que é exigida na *pipeline* CI/CD.

Outro ponto que destacou a estratégia escolhida é que a equipe participante tem um escopo muito fechado no desenvolvimento de novos fluxos do watson então para que a solução de fato pudesse ser completamente desenvolvida se tornou muito necessário o desenvolvimento ágil, além de que possibilita que todo o projeto seja executável por meio de um arquivo com extensão jar o que abre um leque de possiblidades sobre como o projeto poderia ser distribuído entre os times. Por questões de custos em relação a frequência de utilização não faria sentido manter a aplicação executando

de forma contínua. Levando este último quesito em consideração a aplicação foi desenvolvida levando em consideração o uso do arquivo executável, o jar, de forma a ser executada apenas quando fosse solicitada e para que isso pudesse ser visualizado e auditado sem que fosse preciso construir uma ferramenta gráfica desnecessáriamente. Para isso foram utilizadas as *pipelines* do gitlab. De certa forma não foram considerados os pontos negativos desta utilização já que seria útil mesmo se não pudesse ser executada com auxílio das pipelines do gitlab. Bastaria executar o arquivo jar no computador do usuário que o processo seria realizado com sucesso.

Além da utilização da *pipeline* e histórico de alterações nas habilidades o gitlab também foi utilizado para armazenamento desses *JSONs* em detrimento de outras sistemas de gerenciamento não estruturados como é o caso da Amazon que disponibiliza o Amazon S3 como uma forma de armazenar grandes arquivos como fotos, pdfs e etc. Já que o custo dessas soluções costuma ser grande e não chega a possuir um sistema de gerenciamento de versões próprio. O Jira, sistema com muitas funções entre elas o de gerenciamento de tarefas, também foi uma ferramenta usada para facilitar a auditoria e sistema de aprovações já que diversos times utilizariam a solução final e um desses times não possui acesso ao gitlab dificultando o processo inicial de desenvolvimento da solução o que fez com que o Jira pudesse ser uma ótima saída já que se poderia abrir cards de solicitação de implantação no ambiente produtivo.

A containerização foi utilizada por questões de agilidade custos e também por facilitar a execução da solução em momentos sem manipulação humana como o caso do *backup* automático que ocorre todos os dias de meia noite e meio dia. Isto fez com que o docker e o kubernetes fossem utilizados levando em consideração que era a forma mais conhecida entre os participantes de criar uma tarefa temporal.

4 Dificuldades encontradas

A dificuldade na comunicação entre os participantes também foi frequente visto que é uma solução interna que resolve as questões estabelecidas, embora as demandas externas estivessem chegando sempre em peso o que fez com que os horários de cada participante fosse sempre reduzido. Por outro lado, deu a possibilidade de exercer o papel de líder e conduzir o desenvolvimento da arquitetura, trazendo para a equipe discussões acerca do planejamento da implementação e arquitetura.

O erro que gerou a deleção das habilidades de homologação e consequentemente perda do conteúdo trabalhado apagou cerca de duas semanas de desenvolvimento o que fomentou a construção da solução, mas atrapalhou de forma direta já que os participantes tiveram de se envolver na reelaboração do que havia sido perdido. Ao final do processo tornou-se necessário o armazenamento manual por meio do *script* de cada habilidade do watson.

Outro problema é o permissionamento dos sistemas da empresa que é um tanto quanto burocrático, a utilização de docker e do ambiente kubernetes e aws comprometeu bastante o desenvolvimento de uma cronjob que facilitasse o processo de armazenamento automático. Este processo inicialmente foi feito de forma manual utilizando o *script* todos os dias.

Relacionado a isso um dos problemas mais marcantes que aconteceu foi após a conclusão do estado final da solução que foi apresentada neste documento. Ao finalizar a implementação da pipe-

line CI/CD no chamado *framework* da empresa, que consiste na padronização e automação dentro das pipelines construídas pelo time de *DevOps*, foi aberto o *Merge Request* contendo as alterações necessárias. Após análise a equipe de *DevOps* explicou que haviam certas questões acerca do projeto que traria instabilidade para o ambiente compartilhado da empresa o que caracterizou um conflito de interesses entre ambos os times. Após uma conversa entre os times puderam chegar a um consenso e então o projeto pôde ser finalmente colocado em produção para uso dos times envolvidos na manipulação do Watson.

5 Impactos da sua formação no seu trabalho

Conhecimentos teóricos de computação, gerência e disciplinas técnicas de desenvolvimento de software ajudaram no aprendizado de habilidades que reforçam a atenção para aspectos como performance do sistema, manutenibilidade, escalabilidade, disponibilidade e qualidade da aplicação; características amplamente discutidas nas reuniões de concepção do projeto. Cada disciplina deu uma idéia ainda que remota de como funcionariam os processos num ambiente real do mercado de trabalho o que ajudou muito a se ter a confiança necessária para cada fase da carreira até o presente momento. A experiência obtida na graduação permitiu pouco a pouco melhorar as habilidades teóricas e práticas, principalmente a respeito de conceitos que são muito utilizados como: Complexidade de código, desenvolvimento de sistemas, gerência, estratégia e adimnistração; todas que quando juntas constroem o valor da especialização em sistemas de informação.

6 Conclusão

Este trabalho apresentou o processo de construção do sistema trabalhado durante o início da atuação na empresa até o momento de elaboração e conclusão deste documento. Até o momento o projeto continua com sua importância nítida e bem definida de forma que se esperam inclusive melhorias da automação watson desde o que foi apresentado neste documento. São elas:

- Expandir a utilização do projeto para mais times que atuam diretamente com o Watson;
- Melhorias relacionadas a usabilidade da solução;
- Construção de um algoritmo capaz de verificar a diferença entre os *JSONs* das habilidades de homologação e produção (Em fase de testes);
- Geração de relatórios de auditoria (A iniciar pesquisa).

São as melhorias mais prioritárias nesse momento e que vão compor os próximos meses de trabalho dos participantes. A apresentação do problema exigiu uma análise produnda do contexto e momento da empresa, cada qual possui seu próprio quadro de idéias, dificilmente um desenvolvedor tem a oportunidade de começar com o quadro limpo, o dia a dia é muito mais desafiador. Tal quadro já está preenchido e cheio de soluções que podem ou não fazer parte do contexto atual da empresa, podem não resolver em sua plenitude o problema em questão ou até mesmo não serem eficientes

para os parâmetros estabelecidos pela empresa. Teve-se a oportunidade de entender o problema, elaborar uma solução e construí-la de forma a entregar o maior valor possível aos usuários finais.

A partir do entendimento dos dados e dos requisitos identificados foi possível definir a arquitetura e o plano de desenvolvimento da solução, assim como a metodologia aplicada ao processo de desenvolvimento. A arquitetura proposta buscou otimizar e tornar os processos em questão mais eficientes e automatizados que pudessem atender aos requisitos da solução sem demandar uma implementação complexa. Os resultados do projeto foram satisfatórios pois atingiram os requisitos estabelecidos pela liderança da empresa que tem um caráter mais crítico quanto a custos sendo baseados tanto nos recursos utilizados quanto no tempo de trabalho dos colaboradores. A escolha das tecnologias foi pensada de forma a atender aos requisitos de eficiência diminuindo o esforço para a implantação e manutenção do *software* dentro da infraestrutura do watson.

Como apresentado este trabalho apresenta um futuro promissor já tendo tarefas mapeadas que serão desenvolvidas ao longo do ano em questão de forma a tornar o projeto mais robusto e resolver subproblemas que apareceram devido a sua implementação como a construção de um algoritmo capaz de verificar a diferença entre os *JSONs* das habilidades de homologação e produção que auxiliará os desenvolvedores a entenderem com mais assertividade quando um ou mais fluxos foram alterados.

Também é importante que tenhamos um processo de auditoria mais eficiente tendo em vista que a forma que temos de entender o que foi feito, quando foi feito e quem fez é no próprio gitlab. É neste contexto que a geração de relatórios a cada utilização se faz necessária pois não é algo feito de forma específica hoje. Além das novidades é importante estar sempre atento a ferramenta em si já que foi construída com o máximo de agilidade possível então temos que estar sempre buscando identificar pontos de bug e corrigí-los para termos sempre uma ferramenta eficaz.

Referências Bibliográficas

[AMAZON 2022]AMAZON. *O que é o PLN?* 2022. Disponível em: <https://aws.amazon.com/pt/what-is/nlp/>.

[ATLASSIAN 2022]ATLASSIAN. *Entrega contínua*. 2022. Disponível em: <https://www.atlassian.com/br/continuous-delivery>.

[ATLASSIAN 2022]ATLASSIAN. *Integração contínua*. 2022. Disponível em: <https://www.atlassian.com/br/continuous-delivery/continuous-integration>.

[Bartié 2002]BARTIé, A. Garantia da qualidade de software: adquirindo maturidade. Rio de Janeiro: Campus, 2002.

[Erich, Amrit e Daneva 2014]ERICH, F.; AMRIT, C.; DANEVA, M. Report: Devops literature review. 2014. DOI: 10.13140/2.1.5125.1201.

[Fernando 2018]FERNANDO, J. *Descomplicando Docker*. [s.n.], 2018. Disponível em: <https://livro.descomplicandodocker.com.br/pt/>.

[Fernando 2018]FERNANDO, J. *Descomplicando Kubernetes*. [s.n.], 2018. Disponível em: <https://livro.descomplicandokubernetes.com.br/pt/>.

[Fowler 2006]FOWLER, m. Continuous integration. *https://www.academia.edu/download/33643236/ContinuousIntegration.pdf*, 2006.

[HENRY 2017]HENRY, W. From pots and vats to programs and apps: How software learned to package itself. Createspace Independent Publishing Platform, 2017.

[IBM 2021]IBM. *IBM Watson é IA para negócios mais inteligentes*. 2021. Disponível em: <https://www.ibm.com/br-pt/watson>.

[KUBERNETES 2021]KUBERNETES. *O que é Kubernetes?* 2021. Disponível em: <https://kubernetes.io/pt-br/docs/concepts/overview/what-is-kubernetes/>.

[Lisboa 2018]LISBOA, M. J. A importância do gerenciamento de processos de negócios (bpm) na otimização e melhoria contínua de processos de ti. *https://repositorio.animaeducacao.com.br/handle/ANIMA/12184*, 2018.

[Pressman e Maxim 2021]PRESSMAN, R. S.; MAXIM, B. R. Engenharia de software - 9.ed. McGraw Hill Brasil, 2021.

[REDES 2021]REDES, E. S. *Containers e Docker: o que são e como utilizar*. 2021. Disponível em: <https://esr.rnp.br/administracao-de-sistemas/containers-docker-como-utilizar/>.

[REDHAT 2022]REDHAT. *What is an API?* 2022. Disponível em: <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces>.

[ZENDESK 2022]ZENDESK. *O que é chatbot: guia definitivo e completo sobre essa novidade tecnológica*. 2022. Disponível em: <https://www.zendesk.com.br/blog/o-que-sao-chatbots/>.