

Projeto de Computação em Nuvem: Deploy em Nuvem de Modelo de ML Utilizando a IBM Cloud

Laianna L.V. Silva¹, Liviany R. Rodrigues¹, Lucas N. C. Couri¹,
Mariama C. S. Oliveira¹, Priscilla A. Lima¹,

¹ Centro de Informática – Universidade Federal de Pernambuco (UFPE)
50.740-560 – Recife – PE – Brasil

(llvs2, lrr, lncc2, mcs0, pal4)@cin.ufpe.br

Resumo. *Este documento descreve alguns problemas relacionados a dois tipos diferentes de métodos de deploy de modelo de machine learning, utilizando a plataforma da IBM Cloud, como parte da avaliação da disciplina de Computação em Nuvem do curso de Residência em Engenharia e Ciência de Dados da Universidade Federal de Pernambuco.*

Palavras-Chaves. *Deploy, Nuvem, IBM Cloud.*

1. Introdução

A computação em nuvem surgiu como um novo modelo de computação distribuída, oferecendo hardware e software como recursos de serviços habilitados para virtualização. Provedores de computação em nuvem como IBM, Amazon Web Services (AWS), Microsoft Azure oferecem aos proprietários a opção de implantar seus aplicativos através de uma rede virtual com vários recursos com praticamente nenhum investimento de capital inicial e custos operacionais [Wang et al. 2011].

Os serviços de computação em nuvem pública tornaram-se amplamente utilizados devido à sua premissa de pagamento conforme o uso [Goyal 2014]. Um serviço popular para modelo em computação em nuvem é o de Infraestrutura como Serviço (IaaS), em que o cliente em nuvem adquire uma máquina virtual (VM) de acordo com suas necessidades de serviço. Dentro do Modelo IaaS, componentes de software não são compartilhados entre os inquilinos, o que introduz uma carga de processamento.

Recentemente, uma nova estrutura multi-tenancy, conhecida como serviço de containerização, foi introduzida para substituir os hipervisores tradicionais [Medel et al. 2016]. Nesse caso, em vez de hospedar várias VMs executando o sistema operacional, os containers podem compartilhar as bibliotecas do sistema operacional host usando um mecanismo de container (por exemplo, Docker¹) para criar espaços para cada container, realizado como um processo tradicional em um sistema operacional host [Combe et al. 2016].

Docker e Kubernetes² revolucionaram a forma de gerenciar os DevOps [Armstrong 2016] e ambos são atualmente os líderes em ferramentas de orquestração para containers [Uphill et al. 2017]. Há sempre um desafio para controlar um aumento na demanda de dimensionamento e auto-recuperação da rede e instâncias virtuais. Gerenciar

¹Docker: <https://www.docker.com/>

²Kubernetes: <http://kubernetes.io/>

os containers é sempre uma tarefa desafiadora para qualquer empresa porque os micro-serviços que estão rodando nos containers não se comunicam uns com os outros. Eles trabalham de forma independente como uma entidade separada. Este é onde os kubernetes entram, pois é uma plataforma para gerenciar os containers. Esses recipientes podem ser containers docker ou quaisquer outros containers alternativos.

Este trabalho tem como objetivo analisar os problemas relacionados ao deployment levando em consideração o desempenho do modelo de machine learning (ML) em cloud. Diante disso, nosso trabalho teve como proposta testar duas abordagens:

1. Enviar o modelo utilizando a ferramenta de deploy da IBM Cloud;
2. Inserir o modelo no container utilizando o Kubernetes da IBM Cloud e o docker.

A computação em nuvem é um cenário de tecnologia vasto, complexo e em evolução, abrangendo uma pilha de recursos de várias camadas que deve ser orquestrada de uma maneira complexa para garantir que o aplicativo forneça um nível de qualidade aceitável até ao usuário final.

2. Estado da Arte

A Computação em Nuvem iniciou com a necessidade de promover serviços computacionais (software e hardware) em que o usuário fosse capaz de utilizar sobre demanda, ou seja, o usuário é cobrado, somente pelos serviços ou produtos que consumir [Carissimi 2015][Vieira et al. 2015]. Além disso, essa abordagem possibilita o uso de um conjunto de serviços de rede de alta escalabilidade, qualidade de serviço e infraestrutura barata de computação que pode ser acessada de uma forma simples [Sousa et al. 2009].

Na arquitetura em nuvem, o uso de Docker ou container se tornou bastante popular, pois oferece várias vantagens, como implantação facilitada, rapidez e aproveitamento da utilização dos recursos de computação [Menouer 2021]. Essa tecnologia surgiu como uma nova alternativa de virtualização mais leve, conhecida por oferecer um ambiente de desenvolvimento, que possibilita ao sistema operacional do host a criação e execução de aplicações com rapidez e facilidade [Combe et al. 2016][JUNIOR and LIMA][Boettiger 2015]. Um dos seus principais recursos, o Docker Registry, funciona como hospedagem de imagens de containers para colaboração, de forma similar ao GitHub [JUNIOR and LIMA].

Para gerenciar aplicativos automaticamente e escalar containers, foram desenvolvidos vários frameworks de orquestração como o Google Kubernetes, Apache Mesos e Docker SwarmKit [Menouer 2021]. Basicamente, esses recursos servem como uma arquitetura de comunicação entre os hosts, mestre e escravos [Menouer 2021].

Com a crescente adoção do uso da Computação em Nuvem várias empresas começaram a comercializar esses serviços, como por exemplo, a IBM Cloud. A empresa disponibiliza uma plataforma contendo um portfólio com mais de 170 serviços em que é possível desenvolver, implantar, executar e gerenciar aplicações [Azevedo 2020]. Dentro desses ambientes de cloud, é possível ainda, implantar modelos de machine learning (ML) para solucionar problemas que facilitam na tomada de decisão. ML é definida como uma área da Inteligência Artificial (IA) que utiliza operações computadorizada que aprende através da experiência [Rodrigues et al. 2021]. Resumidamente, os modelos de ML utilizam algoritmos de computadores capazes de aprender com uso de um conjunto de dados que funciona como uma base de dados de treinamento.

3. Estudo de Caso

3.1. Watson Machine Learning

A plataforma de Cloud escolhida para o deploy foi a da IBM (utilizando o Watson Machine Learning), e por meio de uma REST API podemos fazer uma requisição para que o modelo hospedado na nuvem faça as previsões. Utilizamos um notebook em Python para realizar os testes das requisições, mas vale notar que o código também pode ser implementado em uma aplicação em Python. O primeiro passo é instalar e importar as bibliotecas e dependências necessárias para o processo, em especial faz-se necessário o uso da biblioteca *ibm-watson-machine-learning*³.

```
!pip install -U ibm-watson-machine-learning  
  
from ibm_watson_machine_learning import APIClient
```

A seguir criamos um espaço de API na plataforma IBM Cloud, para isso basta acessar a opção de Machine Learning dentro do catálogo de serviços. Para prosseguir selecione a região e o plano de hospedagem, escolhemos a região de Dallas e o plano Lite (grátis), que oferece algumas limitações mas é suficiente para fins de aprendizado. Acessando o Watson Studio na área de Deployments é possível criar um novo espaço de deployment, assim podemos ir na seção de acesso e criar uma nova chave de API. Também é necessário definir um dicionário no Python com as seguintes credenciais: a chave de API criada anteriormente e a URL da região de hospedagem.

Dessa forma temos tudo necessário para definir um cliente API de Machine Learning dentro do código em Python, esse cliente será responsável por acessar o espaço criado a partir da chave de API e URL fornecidas no dicionário. Podemos verificar os espaços existentes e obter o ID para a chave de API desejada, o qual armazenamos em uma variável para uso futuro.

O próximo passo é salvar e dar deploy no modelo de machine learning, para isso definimos a versão do Python e outras especificações do modelo, além de passar os dados de treinamento. De forma similar, é necessário definir as especificações do deploy, e criar o mesmo. Após a criação do deploy é possível fazer requisições para a API passando os dados, e recebendo como resultado as previsões feitas pelo modelo. A implementação de todo esse processo encontra-se no notebook disponibilizado no GitHub⁴ do projeto.

3.2. Deploy em Container

Para realizar o deploy em container nós utilizamos o FastAPI⁵ em conjunto com o modelo de análise de sentimentos *en_core_web_sm* da biblioteca *spacy*⁶ do python, para desenvolver uma pequena aplicação que é capaz de retornar a polaridade e a subjetividade de uma frase em inglês.

Com a aplicação pronta, nós programamos um Dockerfile e geramos uma imagem do Docker correspondente e testamos a funcionalidade. Após a verificação, nós iniciamos as etapas posteriores que nos levaram até o deploy em container.

³*ibm-watson-machine-learning*:
<https://pypi.org/project/ibm-watson-machine-learning/>

⁴GitHub do projeto: <https://github.com/lucascouri2/projeto-residencia-nuvem>

⁵FastAPI: <https://fastapi.tiangolo.com/>

⁶scapy: <https://spacy.io/models/en>

3.2.1. Criando um Cluster Kubernetes no IBM Cloud

Como o intuito era deixar o container do Docker rodando em um cluster gerenciado pela IBM Cloud, precisamos criar um cluster do Kubernetes no site da IBM Cloud para servir como servidor para o nosso projeto.

A criação do cluster foi feita através do próprio site da IBM Cloud, visto que, este já consta com uma área para gerar o cluster do Kubernetes, Figura 1.

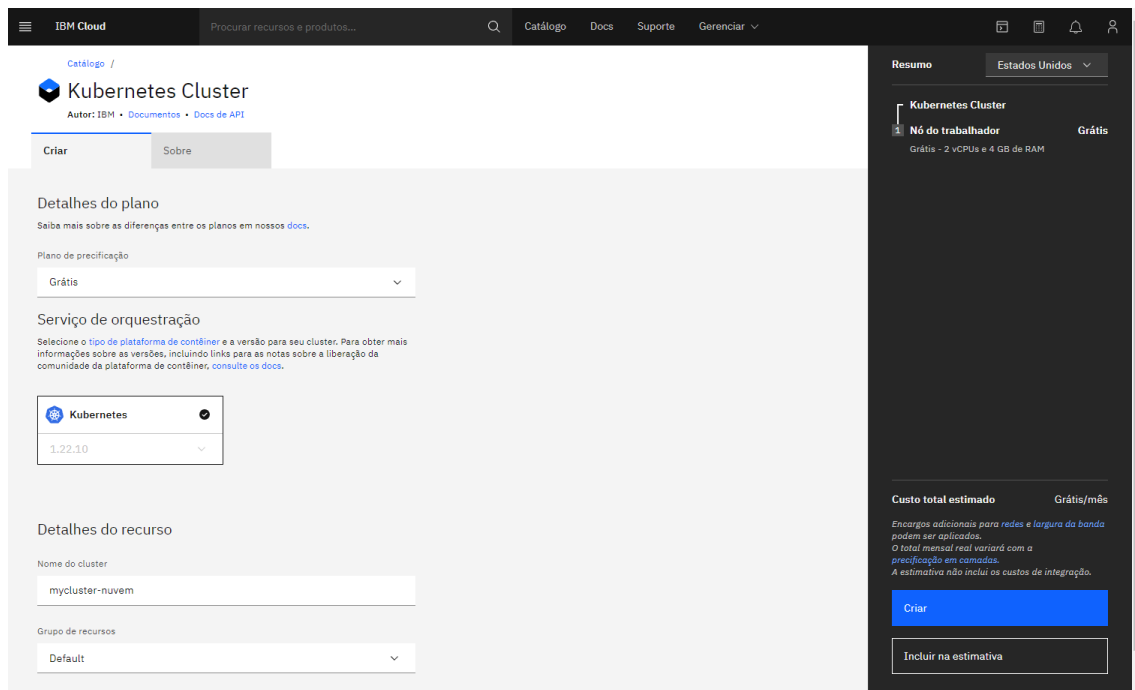


Figura 1. Criação de um cluster do Kubernetes gratuito através da página da IBM Cloud.

Nós optamos por criar um cluster gratuito, pois era suficiente para o nosso projeto, mas as configurações do cluster a ser gerado podem ser mudadas de acordo com as necessidades e a disponibilidade de pagamento do usuário. Uma coisa que deve ser lembrada, é que o cluster gratuito fica disponível apenas por trinta dias e depois disso é necessário reativá-lo.

Após a finalização da configuração, a lista de clusters do Kubernetes no site da IBM Cloud foi atualizada, ficando parecida com a mostrada na Figura 2.

O cluster que criamos tem o nome *mycluster-nuvem* e está localizado em Milão na Itália. Esta informação é importante, pois precisamos dela para escolher a região do usuário nos próximos passos. Outra informação relevante, é o ID do cluster, *calnfggf0pp4d05hm7g*, que também encontra-se disponível no site, Figura 3.

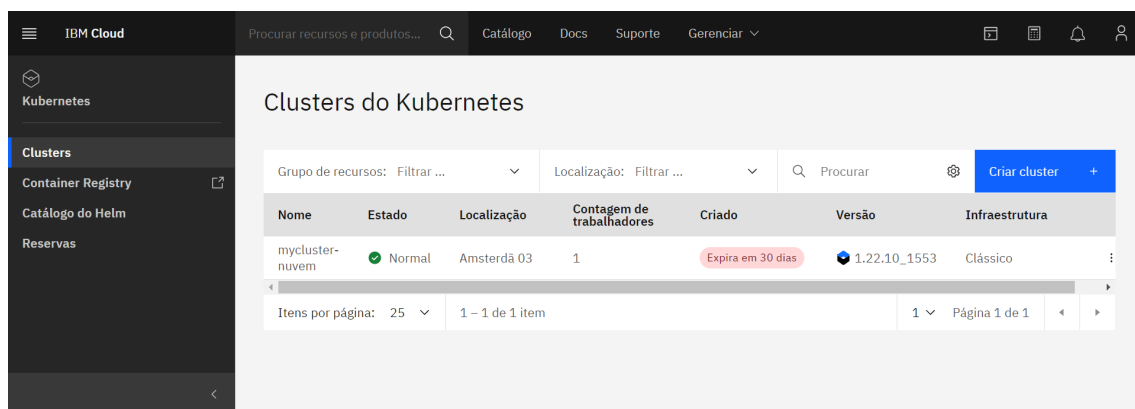


Figura 2. Painel do Kubernetes no site da IBM Cloud após criação de um cluster.

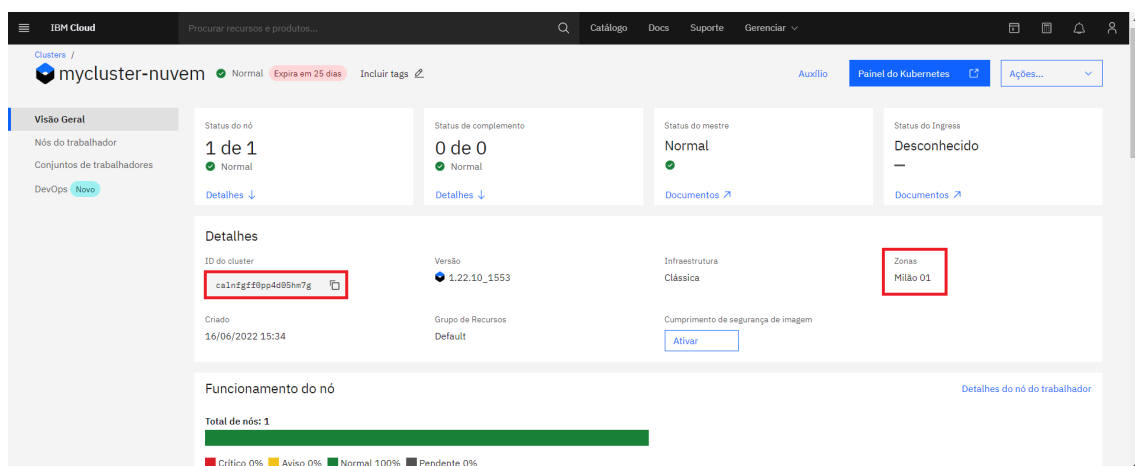


Figura 3. Painel de informações do *mycluster-nuvem* no site da IBM Cloud, destacando as informações de ID e região.

3.2.2. Configurando o ambiente do IBM CLI

Para possibilitar interações com o IBM Cloud, foi necessário fazer a instalação do IBM CLI⁷ (Cloud Command Line), que provém um conjunto de ferramentas que atuam nesse processo⁸.

Utilizando o IBM CLI, seguimos o tutorial inicial de configuração dos plug-ins e das ferramentas⁹. O primeiro passo para a configuração do ambiente foi o de logar na conta pessoal da IBM Cloud usando o comando

```
ibmcloud cr login
```

, permitindo a entrada das informações de usuário e senha. Durante o processo de login, foi solicitada a região de acesso. Nesse ponto, deve-se colocar a região mais próxima a onde o cluster do Kubernetes se encontra.

⁷IBM Cloud Developer Tools CLI: <https://www.ibm.com/br-pt/cloud/cli>

⁸Getting started with the IBM Cloud CLI: <https://cloud.ibm.com/docs/cli?topic=cli-getting-started>

⁹Installing the tools and plug-ins manually: <https://cloud.ibm.com/docs/cli?topic=cli-install-devtools-manually>

Após completar o login com as credenciais do IBM Cloud, foi necessário fazer a instalação dos plug-ins para que os comandos mais a frente funcionassem. Para isso, optou-se por utilizar o comando

```
ibmcloud plugin install -all
```

, onde foram instalados todos os plug-ins do IBM CLI.

3.2.3. Configurando o Namespace

O próximo passo foi a criação de um namespace. O namespace é criado no grupo de recursos especificado pelo usuário. Caso o grupo de recursos não seja especificado, será automaticamente utilizado o padrão¹⁰.

Para criar um namespace em um grupo de recursos padrão, utilizamos o comando

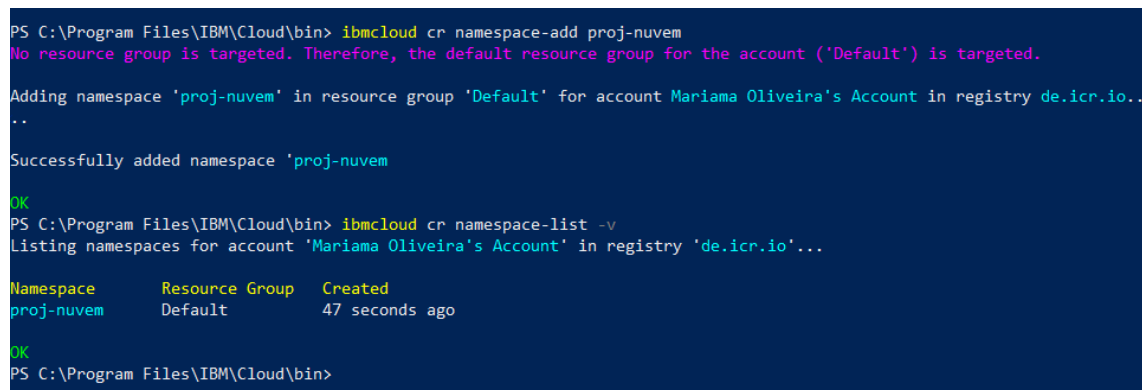
```
ibmcloud cr namespace-add proj-nuvem
```

, onde *proj-nuvem* é o nome escolhido para o nosso namespace.

Para verificar se o namespace foi criado, executamos o comando

```
ibmcloud cr namespace-list -v
```

, que lista os namespaces pertencentes ao usuário e o grupo de recursos correspondente. Os dois comandos executados podem ser vistos na Figura 4.



```
PS C:\Program Files\IBM\Cloud\bin> ibmcloud cr namespace-add proj-nuvem
No resource group is targeted. Therefore, the default resource group for the account ('Default') is targeted.
Adding namespace 'proj-nuvem' in resource group 'Default' for account Mariama Oliveira's Account in registry de.icr.io...
..
Successfully added namespace 'proj-nuvem'
OK
PS C:\Program Files\IBM\Cloud\bin> ibmcloud cr namespace-list -v
Listing namespaces for account 'Mariama Oliveira's Account' in registry 'de.icr.io'...

Namespace      Resource Group  Created
proj-nuvem     Default        47 seconds ago
OK
PS C:\Program Files\IBM\Cloud\bin>
```

Figura 4. IBM CLI após a execução dos comandos de criação e exibição do namespace.

Após a criação bem sucedida, as informações do namespace no cointainer registry foram atualizados no site da IBM Cloud, Figura 5.

3.2.4. Selecionando o Cluster

Antes de subir a imagem do Docker e criar o container, foi preciso selecionar no terminal o cluster que seria utilizado para o trabalho,

¹⁰Getting started with IBM Cloud Container Registry: <https://cloud.ibm.com/docs/Registry?topic=Registry-getting-started#getting-started>

`ibmcloud ks cluster config -c calnfgff0pp4d05hm7g`
, utilizando o ID que indicamos na Seção 3.2.1. A saída do IBM CLI pode ser vista na Figura 6.

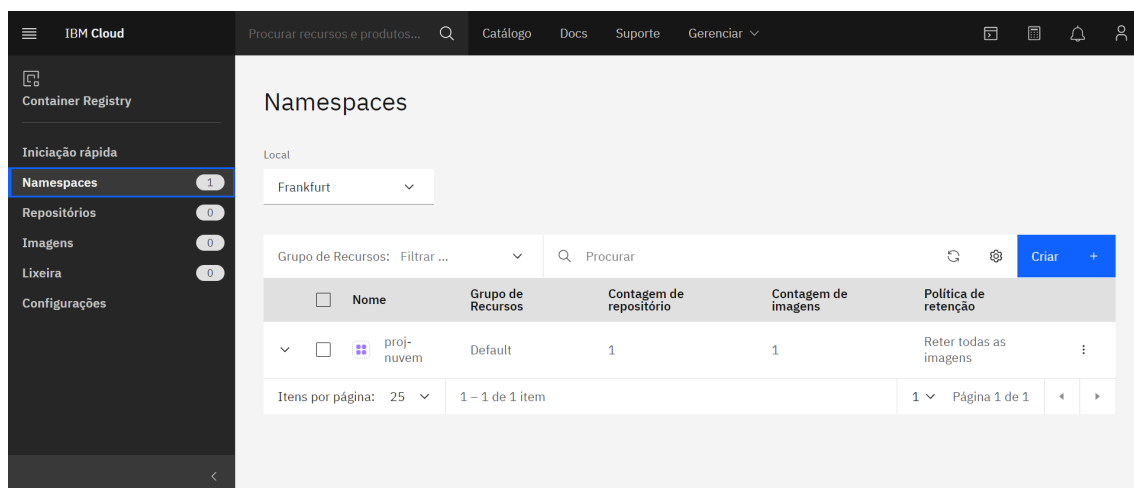


Figura 5. Painel do Container Registry no site da IBM Cloud após criação do namespace *proj-nuvem*.

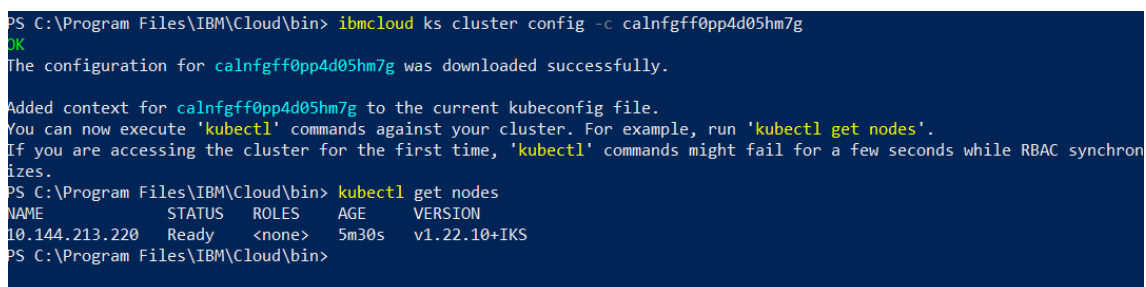


Figura 6. IBM CLI após a execução do comando de configuração do cluster Kubernetes *mycluster-nuvem*.

3.2.5. Fazendo o Deploy

Depois de configurar o cluster pelo IBM CLI, criamos uma cópia da imagem do nosso projeto que estava no Docker, utilizando a nomenclatura aceita pela IBM Cloud. Para isso utilizamos o comando,

`docker tag nuvem:latest de.icr.io/proj-nuvem/nuvem:latest`
, onde *nuvem:latest* é o nome da imagem do Docker que queremos copiar; *de* é a região que escolhemos quando fizemos o login do usuário; e *proj-nuvem* é o nome do namespace.

Depois disso, aplicamos o comando

`docker push de.icr.io/proj-nuvem/nuvem:latest`
, para enviar a nova imagem para o namespace *proj-nuvem*, Figura 7. Seguido pelo comando

```
ibmcloud cr image-list
```

, que verifica se a imagem foi subida corretamente para o namespace. Outra forma de fazer essa verificação é através do site da IBM Cloud, Figura 8.

```
PS C:\Program Files\IBM\Cloud\bin> docker push de.icr.io/proj-nuvem/nuvem:latest
The push refers to repository [de.icr.io/proj-nuvem/nuvem]
14bc3b5db2e2: Pushed
408346f32923: Pushed
2ef9611d247d: Pushed
b4a62f0deac1: Pushing [=====> ] 189.6MB/201.7MB
3dde1e8b1316: Pushed
52f29e7bfaa2: Pushed
c6679bcc482d: Pushed
aedb4b362167: Pushed
285c8ba236e9: Pushed
13b045a1dfd2: Pushed
2fbabeba902e: Pushing [=====> ] 144.3MB/528.7MB
ee509ed6e976: Pushing [=====> ] 34.04MB/151.9MB
9177197c67d0: Pushing [====> ] 1.222MB/18.95MB
7dbadf2b9bd8: Pushing [> ] 107kB/10.69MB
e7597c345c2e: Waiting
```

Figura 7. IBM CLI mostrando as informações do serviço.

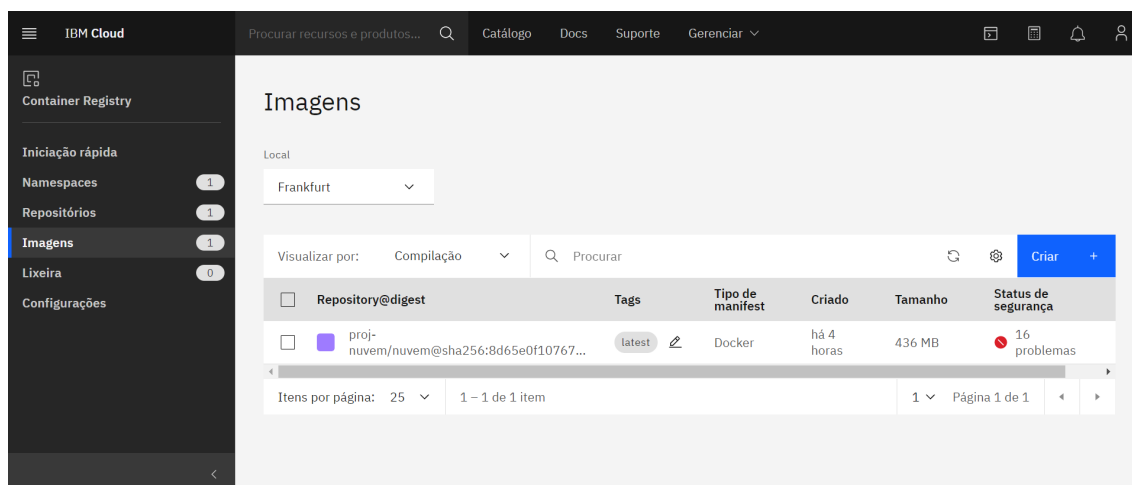


Figura 8. Painel do Container Registry no site da IBM Cloud após o upload da imagem do Docker.

Por fim, usamos os dois comandos

```
kubectl create deployment nuvem-deployment
--image=de.icr.io/proj-nuvem/nuvem
```

```
kubectl expose deployment/nuvem-deployment --type=NodePort
--port=8080 --name=nuvem-service --target-port=8080
```

que, respectivamente, criou e publicou o deploy utilizando a imagem que encontrava-se em nosso namespace, Figura 9.

3.2.6. Obtendo as Informações Para Comunicação Com a Aplicação

Com o intuito de nos comunicarmos com a aplicação, precisamos descobrir em qual IP e em qual porta o servidor encontra-se disponível.

Para descobrir o IP, utilizamos o comando


```
ibmcloud ks worker ls -cluster mycluster-nuvem
```

, que nos retornou o IP 169.51.203.17 como podemos ver na Figura 10 no campo de *Public IP*.

```
PS C:\Program Files\IBM\Cloud\bin> ibmcloud cr image-list
Listing images...

Repository          Tag      Digest      Namespace    Created      Size      Security status
de.icr.io/proj-nuvem/nuvem  latest  8d65e0f10767  proj-nuvem   1 hour ago  436 MB    Scanning...

OK
PS C:\Program Files\IBM\Cloud\bin> kubectl create deployment nuvem-deployment --image=de.icr.io/proj-nuvem/nuvem
deployment.apps/nuvem-deployment created
PS C:\Program Files\IBM\Cloud\bin> kubectl expose deployment/nuvem-deployment --type=NodePort --port=8080 --name=nuvem-s
service --target-port=8080
service/nuvem-service exposed
```

Figura 9. IBM CLI após o deployment.

```
PS C:\Program Files\IBM\Cloud\bin> ibmcloud ks worker ls --cluster mycluster-nuvem
OK
ID                                     Public IP      Private IP      Flavor  State  Status  Zone
e   Version
kube-calnfgff0pp4d05hm7g-myclusternu-default-000000d2  169.51.203.17  10.144.213.220  free   normal Ready  mil
01  1.22.10.1554
PS C:\Program Files\IBM\Cloud\bin>
```

Figura 10. IBM CLI mostrando o IP público que utilizamos para acessar a aplicação.

Após a finalização do deploy, foi possível obter os detalhes da aplicação através do comando

```
kubectl describe service nuvem-service
```

, que nos permitiu ter informações do serviço, incluindo o NodePort que foi designados para nossa aplicação. Segundo os dados informados na Figura 11, temos que 31284 é a porta que deve ser utilizada para comunicação.

```
PS C:\Program Files\IBM\Cloud\bin> kubectl describe service nuvem-service
Name:          nuvem-service
Namespace:     default
Labels:        app=nuvem-deployment
Annotations:    <none>
Selector:      app=nuvem-deployment
Type:          NodePort
IP Families:   <none>
IP:            172.21.198.52
IPs:           172.21.198.52
Port:          <unset> 8080/TCP
TargetPort:    8080/TCP
NodePort:      <unset> 31284/TCP
Endpoints:     172.30.108.77:8080
Session Affinity: None
External Traffic Policy: Cluster
Events:        <none>
PS C:\Program Files\IBM\Cloud\bin>
```

Figura 11. IBM CLI mostrando as informações do serviço.

Algumas informações também estão disponíveis através do site da IBM Cloud, como mostrado na Figura 12 e Figura 13.

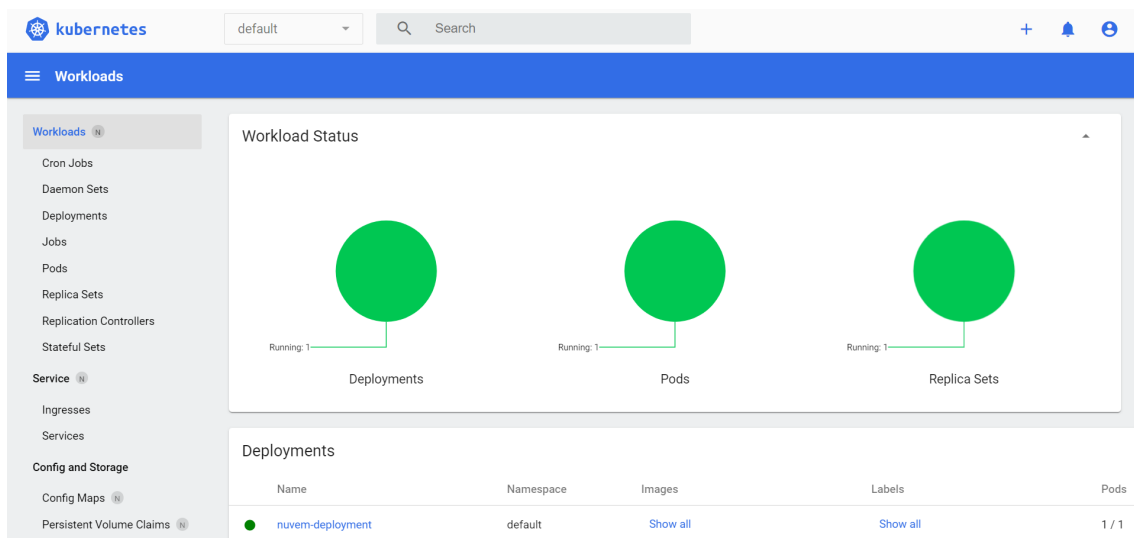


Figura 12. Workload status do *mycluster-nuvem* no Painel do Kubernetes.

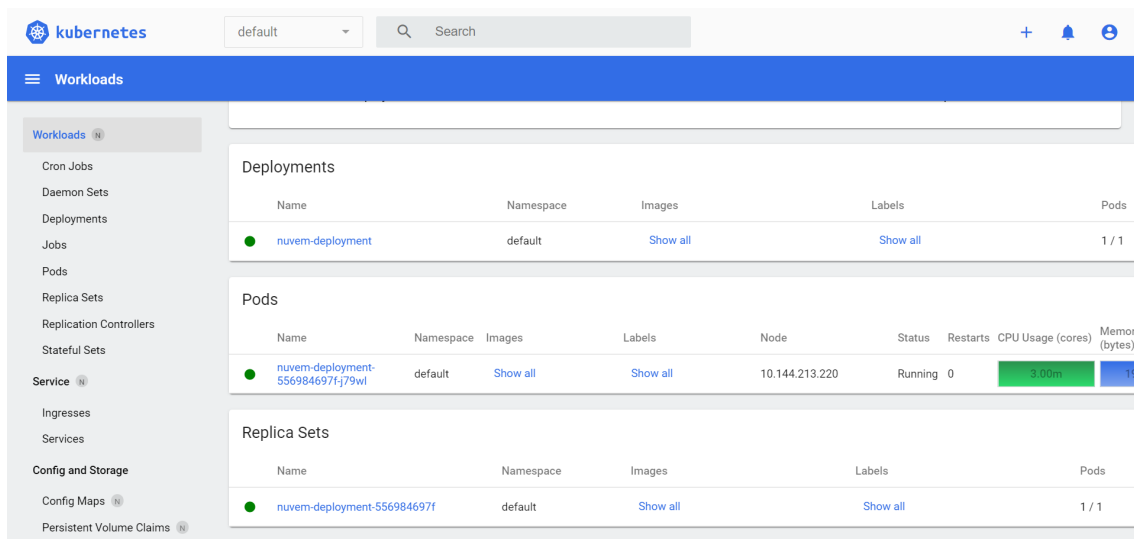


Figura 13. Deployments, pods e replica sets do *mycluster-nuvem* no Painel do Kubernetes.

3.2.7. Testando a Comunicação Com a Aplicação

Em posse do IP e da porta, podemos nos comunicar com a API Rest. Nós fizemos a comunicação de duas formas distintas: a primeira, foi através de um notebook em python e a segunda através de um navegador.

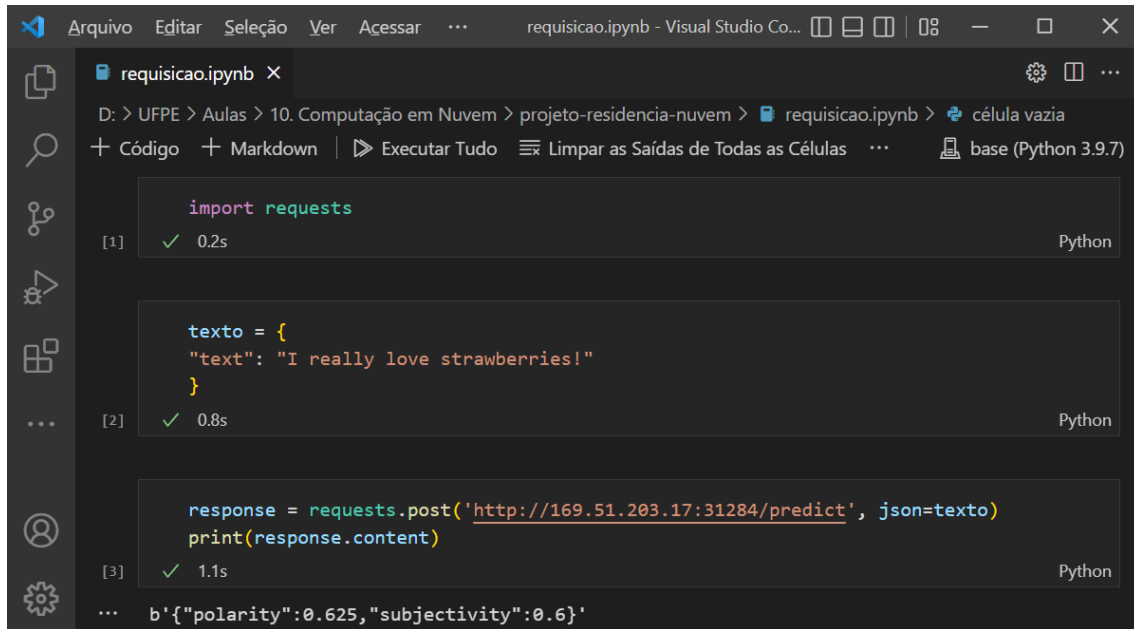
Para fazer a comunicação através de um notebook, nós utilizamos a biblioteca *requests*¹¹ do python e enviamos a mensagem em inglês: *"I really love strawberries!"*. Como resultado, obtivemos a resposta da aplicação, Figura 14, que nos retornou a polaridade e subjetividade da mensagem assim como nós esperávamos.

Já a comunicação através do navegador, foi feita digitando na barra de endereço

¹¹requests: <https://pypi.org/project/requests/>

16951.203.17:31284/docs

, que corresponde ao IP, a porta e a seção que queríamos acessar. Nós utilizamos a mesma frase anterior e obtivemos o resultado exposto na Figura 15.



```
import requests

[1] ✓ 0.2s Python

texto = {
    "text": "I really love strawberries!"
}

[2] ✓ 0.8s Python

response = requests.post('http://169.51.203.17:31284/predict', json=texto)
print(response.content)

[3] ✓ 1.1s Python

... b'{"polarity":0.625,"subjectivity":0.6}'
```

Figura 14. Comunicação com a API Rest utilizando um notebook em python no Visual Studio Code.

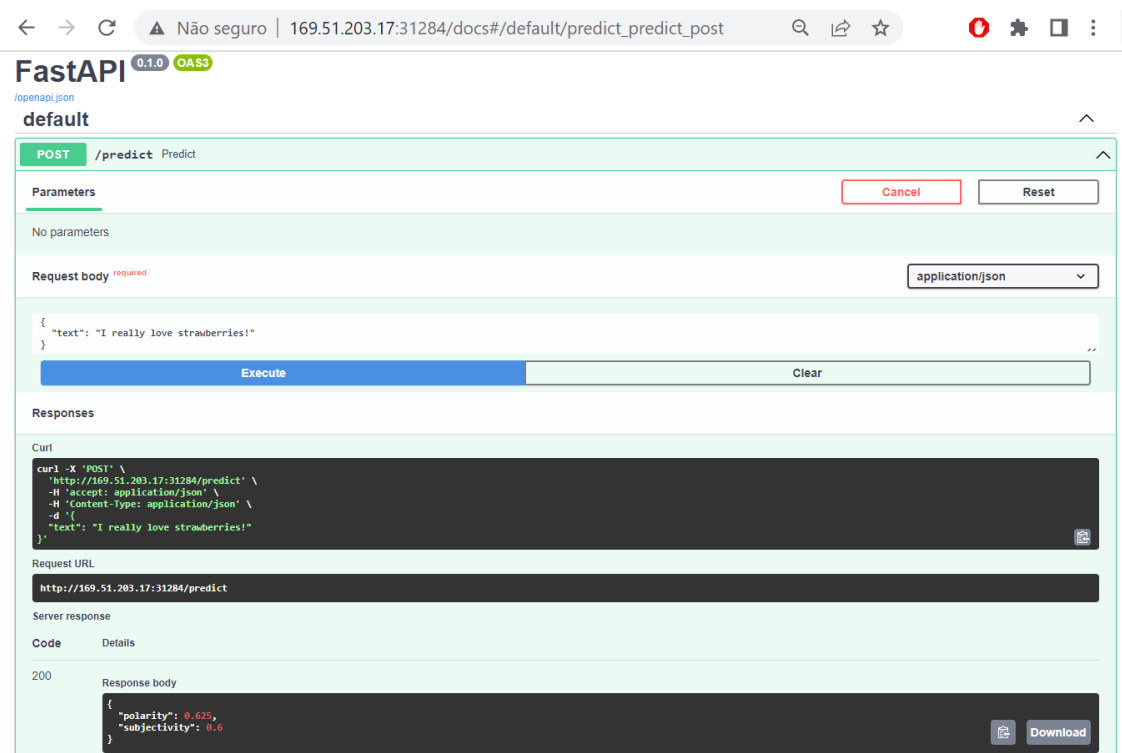


Figura 15. Comunicação com a API Rest utilizando o navegador.

4. Considerações Finais

No presente estudo, duas formas de fazer deploy em modelos de ML foram exploradas, utilizando container ou fazendo o upload direto para o serviço de cloud. A partir disso, foram analisadas algumas características dessas duas abordagens: tempo e facilidade de implementação, manutenção, monitoramento e custo. Essas características não exaurem todos os aspectos dos métodos escolhidos, no entanto devido ao tempo de investigação, optou-se por priorizar essas quatro propriedades.

4.1. Tempo e facilidade de implementação

O tempo e facilidade de implementação, no âmbito de desenvolvimento de software, são características que podem ser relativas ao indivíduo e equipe envolvida no projeto. Em sua grande maioria, equipes experientes em determinada tecnologia ou ferramenta tendem a ter maior facilidade e agilidade de implementação, devido a diversos motivos desde nível de expertise acumuladas ou até mesmo reuso de código. Nesse projeto, no entanto, nenhum componente da equipe tinha experiência prévia relevante em nenhuma das abordagens utilizadas. Portanto, as considerações a seguir são relatos de pessoas iniciantes no contexto de deployment em cloud. Para tal, serão avaliados a documentação existente, tutoriais da comunidade disponíveis e facilidade de uso das ferramentas.

Para ambas as abordagens, a documentação disponibilizada pela IBM apresenta problemas. Ela não é simples de ser acompanhada por pessoas iniciantes, além de apresentar informações desatualizadas. Durante o deploy do container, por exemplo, a equipe se perdeu diversas vezes no tutorial, pois não sabia que inicialmente deveria ser criado um node do Kubernetes para fazer deploy no container. Ao final, para conseguir compreender algum tutorial, foi necessário recorrer a vídeos produzidos pela comunidade, pois estes eram mais explicativos. A ajuda de material produzido pela comunidade foi imprescindível para a realização do projeto. O número de tutoriais ensinando em como fazer deploy de uma API REST utilizando containers foi sem dúvida o encontrado com mais facilidade. Já o material da comunidade sobre deploy de containers e modelos de ML no serviço da IBM já eram mais reduzidos, mas ainda é possível encontrar blogs e vídeos sobre o assunto.

Quanto a facilidade de uso, o deploy de forma direta no serviço, à primeira vista, apresenta maior facilidade de uso, pois o número de passos é menor do que o método utilizando containers. Quando utilizando containers, além de codificar as funções da API REST, o desenvolvedor fica responsável por configurar o container — tarefa essa que demandou bastante tempo da equipe, devido ao conhecimento limitado da tecnologia. Para só no final, fazer deploy do container utilizando Kubernetes. Todos esses passos tornam o método de container mais difícil de uso, uma vez que cada etapa dessa está sujeita a erros e demandam conhecimento específico. Todavia, caso o pipeline do modelo seja orientado a containers, essas dificuldades talvez não façam sentido, pois o deploy em container talvez seja a saída mais natural. Outra facilidade dos containers é na compatibilidade do modelo gerado. Qualquer modelo será “compatível” utilizando containers, uma vez que a containerização torna o modelo construído independente das restrições da plataforma de cloud. Fato que não é verdade para o deploy direto em cloud. A plataforma de ML da IBM, por exemplo, restringe o upload de modelos a somente a alguns formatos.

4.2. Manutenção

O serviço de deploy de modelos de ML em serviços de cloud pode ser facilmente comparado com o conceito de Serverless, no qual o serviço de cloud fica responsável por executar “pedaços” de código, mas abstraindo boa parte da infraestrutura por trás para o desenvolvedor. No caso de deploys diretos de modelos de ML, o mesmo ocorre, o que torna a manutenção do modelo muito mais fácil. Toda parte de configuração, gestão da infraestrutura e segurança é realizada pelo serviço de cloud. Já no método de container a camada de abstração é um pouco menor. Ainda que haja diversas facilidades oferecidas pelas plataformas de gestão de containers, o usuário ainda é responsável por gerir os containers dentro do serviço de orquestração.

Outro aspecto a se considerar na manutenção é a possibilidade de migração para outro serviço. Com containers essa tarefa é bastante simples, pois a imagem gerada pela plataforma de containerização é facilmente utilizada nos diferentes serviços de cloud. Já a migração de um modelo carregado diretamente ainda é uma tarefa obscura, uma vez que esses serviços ainda estão em consolidação e não existe um único padrão aceito em todas as plataformas.

4.3. Monitoramento

De acordo com a IBM Cloud, o monitoramento dos modelos de ML a partir do deploy direto em sua cloud é uma das grandes vantagens do serviço. Por ele é possível ver informações específicas sobre o modelo, como tipos de dados e algoritmo. Além disso, há um dashboard com as métricas do modelo e um serviço que ajuda a detectar drifts de dados. Isso tudo torna o trabalho do cientista de dados muito mais fácil, uma vez que ele não precisa em se preocupar em implementar tudo isso. Embora essas features existam, a equipe não conseguiu fazer o uso na plataforma, pois nem conseguiu encontrá-las. Tudo isso também é possível com containers, mas é algo que demanda implementação extra.

4.4. Custo

Em relação às despesas na IBM Cloud, no deployment direto, o cliente contrata um plano que lhe provê horas de processamento na plataforma. Dessa forma, ele só paga pelo que utiliza. Já utilizando o deployment de containers, o cliente deve pagar pelas horas que deixa o cluster Kubernetes ativo, ou seja, ele pagará pelo serviço mesmo que não haja requisições para a API. Em virtude dessas características, utilizar o serviço de container pode ser um alto investimento quando comparado com o deployment direto. Para alguns projetos, no entanto, o gasto de deixar um cluster Kubernetes ativo pode valer a pena, pois em alguns casos os benefícios trazidos pela containerização justifica o custo mais elevado. Pagar um cluster para fazer deploy de somente um microserviço talvez não faça sentido, mas imagine o cenário na qual a aplicação dependa de diversos microserviços, o investimento com containerização começa a recompensar.

4.5. Conclusão

Baseado na breve análise realizada de cada opção de deployment, observa-se que cada serviço possui vantagens e desvantagens. Cabe ao desenvolvedor analisar quais características são mais convenientes para o projeto em desenvolvimento, para desse modo fazer a escolha mais apropriada.

Referências

- Armstrong, S. (2016). *DevOps for Networking*. Packt Publishing Ltd.
- Azevedo, L. G. (2020). Desenvolvimento de soluções com serviços: Soa, cloud e micro-serviços. *Sociedade Brasileira de Computação*.
- Boettiger, C. (2015). An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79.
- Carissimi, A. (2015). Desmistificando a computação em nuvem. *Instituto de Informática–Universidade Federal do Rio Grande do Sul (UFRGS)–Porto Alegre–RS*.
- Combe, T., Martin, A., and Di Pietro, R. (2016). To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62.
- Goyal, S. (2014). Public vs private vs hybrid vs community-cloud computing: a critical review. *International Journal of Computer Network and Information Security*, 6(3):20–29.
- JUNIOR, F. C. S. and LIMA, B. S. N. M. Tecnologia docker.
- Medel, V., Rana, O., Banares, J. A., and Arronategui, U. (2016). Modelling performance & resource management in kubernetes. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, pages 257–262.
- Menouer, T. (2021). Kcss: Kubernetes container scheduling strategy. *The Journal of Supercomputing*, 77(5):4267–4293.
- Rodrigues, F., Rodrigues, F. A., and Rocha, T. V. (2021). Modelos de machine learning para predição do sucesso de startups. *Gestão e Projetos: GeP*, 12(2):28–55.
- Sousa, F. R., Moreira, L. O., and Machado, J. C. (2009). Computação em nuvem: Conceitos, tecnologias, aplicações e desafios. *II Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)*, pages 150–175.
- Uphill, T., Arundel, J., Khare, N., Saito, H., Lee, H.-C. C., and Hsu, K.-J. C. (2017). *DevOps: Puppet, Docker, and Kubernetes*. Packt Publishing Ltd.
- Vieira, C. S., Mereilles, F. d. S., and Cunha, M. A. (2015). Fatores que influenciam o indivíduo na utilização da computação em nuvem.
- Wang, L., Ranjan, R., Chen, J., and Benatallah, B. (2011). *Cloud computing: methodology, systems, and applications*. CRC Press.