# Capítulo 2

# Inteligência Artificial e Função como Serviço: Provisionando Aplicações com o AWS Lambda

Sayonara S. Araújo (UFC), Francisco R. P. da Ponte (UFC), Victória T. Oliveira (UFC), Wendley S. da Silva (UFC), Dario Vieira (EFREI), Miguel F. de Castro (UFC) e Emanuel B. Rodrigues (UFC)

***Abstract***

*Data can be a valuable resource for companies, as it helps in decision making, performance analysis, and problem-solving. Estimates indicate that an average of 2.5 quintillion bytes of data are generated daily. The use of technology like Artificial Intelligence (AI) is indispensable for extracting useful information from this large amount of data. Companies that provide Cloud Computing services, such as Amazon with its Amazon Web Service (AWS) platform, allow the easy deployment of AI solutions capable of taking advantage of this immense volume of data. An example of these services is AWS Lambda, one of the main tools of what is known as serverless computing. Within this context, this mini-course will introduce the concepts of Function as a Service (FaaS), which encompass the use of AWS Lambda to provision AI applications.*

***Resumo***

*Os dados podem ser um recurso muito valioso para as empresas, visto que eles auxiliam na tomada de decisão, análise de desempenho e na resolução de problemas. Estima-se que em média 2,5 quintilhões de bytes são gerados diariamente. Para conseguirmos extrair informações úteis dessa grande quantidade de dados, o uso de uma tecnologia como a Inteligência Artificial (AI) é indispensável. Empresas que prestam serviços de Computação em Nuvem, como a Amazon com sua plataforma Amazon Web Service (AWS), permitem a implantação facilitada de soluções de AI capazes de tirar proveito desse imenso volume de dados. Para isso, pode-se utilizar, por exemplo, o AWS Lambda, uma das principais ferramentas do chamado serviço de computação sem servidor. Dentro desse contexto, o presente minicurso introduz os conceitos de Função como Serviço (FaaS), englobando o uso de AWS Lambda para provisionar aplicações de AI.*

## 2.1. Introdução

Em uma época de destaque para o *Big Data*, negócios enfrentam desafios de lidar com uma grande quantidade de informações presentes no meio digital. Estima-se que 2,5 quintilhões de *bytes* são gerados por dia [1]. Para aproveitar esse volume de informações, é fundamental que haja meios eficientes de lidar com todo esse material.

Nesse contexto, a Inteligência Artificial – *Artificial Intelligence* (AI) – é uma tecnologia relevante para criar produtos que consigam analisar grande volume de dados. A AI se refere resumidamente a sistemas computacionais que tentam imitar a inteligência humana para realizar atividades com base nas informações disponíveis [2]. Devido ao seu potencial, nas últimas décadas, essa área vem impactando significativamente o mundo real [3].

Outra tecnologia que atualmente tem agregado nesse contexto de análise de dados é a Computação em Nuvem, um paradigma que surgiu para suportar o processamento e o armazenamento de grandes volumes de dados em conjuntos de computadores externos [4]. Assim, a junção de ambas as tecnologias permite potencializar o tratamento de informações.

O advento de plataformas de Computação em Nuvem, como a Amazon Web Service (AWS), permitiu a implantação facilitada de soluções capazes de tirar proveito desse imenso volume de dados [5]. Para isso, utiliza-se o chamado serviço de computação sem servidor, especialmente a Função como Serviço – *Function as a Service* (FaaS) –, que torna a implantação de arquiteturas de aplicativos corporativos para contêineres e microsserviços mais fácil e econômica [6].

O AWS Lambda foi a primeira ferramenta de mercado a possibilitar a execução de serviços como FaaS [7] e continua sendo uma das principais [8]. Dentro desse cenário, o presente minicurso introduz o conceito de FaaS, englobando o uso de *AWS Lambda* em soluções de AI. Ao longo deste trabalho, serão apresentados uma fundamentação teórica e alguns direcionamentos práticos para o provisionamento de aplicações de AI na FaaS da AWS. Para isso, o texto a seguir se divide nas seguintes seções: Inteligência Artificial (2.2), Contêiner (2.3), Função como Serviço (2.4), Prática (2.5) e Considerações Finais (2.6).

## 2.2. Inteligência Artificial

O termo Inteligência Artificial foi cunhado em 1956, em uma conferência no Dartmouth College[1], onde um grupo de matemáticos e cientistas se reuniu para discutir como aspectos de aprendizagem e inteligência podem, a princípio, serem descritos com tamanha precisão que uma máquina (i.e. um *software*) pode ser construída para simulá-los [9].

A disciplina de AI teve um desenvolvimento inicial lento, mas a partir da década de 90 e início do século XXI, com o desenvolvimento de computadores mais potentes e o aumento no volume de dados, as pesquisas nessa área aumentaram consideravelmente [10]. Esse crescimento pode ser visto na Figura 2.1, que mostra a quantidade de trabalhos publicados em AI ao longo dos últimos 30 anos, informação obtida através de uma

[1] Universidade estadunidense localizada no estado de New Hampshire: https://home.dartmouth.edu/.

consulta realizada no site Scopus[2] (acessado em 13 de agosto de 2022).

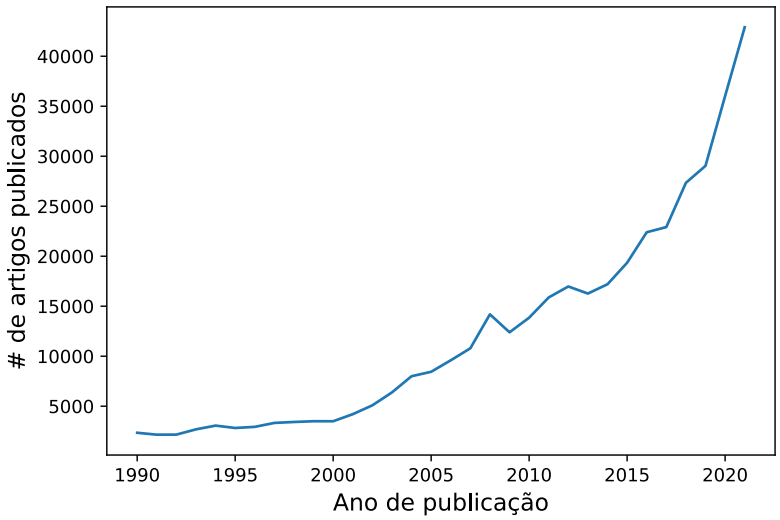


Figura 2.1: Número de artigos publicados em AI por ano desde 1990. Observa-se um crescimento acentuado de artigos publicados na área de AI a partir de 2000 [Autores].

### 2.2.1. Aprendizado de Máquina

A capacidade dos sistemas aprenderem e melhorarem seu funcionamento a partir de dados foi algo tão significativo para a área de AI, que um novo campo chamado de Aprendizado de Máquina – *Machine Learning* (ML) – surgiu. ML é um conjunto de métodos, ou algoritmos, que utilizam probabilidade e estatística para melhorar o desempenho de uma determinada tarefa [11]. Como podemos observar na Figura 2.2, que mostra um diagrama de Veen, ML é um subconjunto de AI.

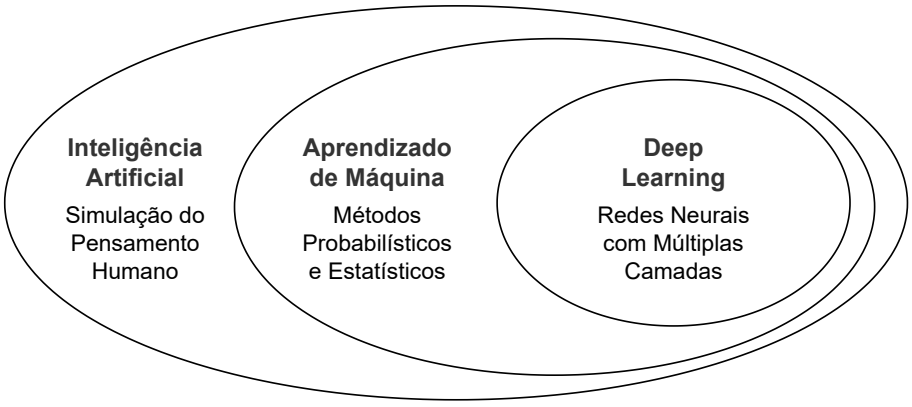


Figura 2.2: Diagrama de Veen que representa graficamente como os conjuntos de AI, ML e *Deep Learning* se correlacionam [Autores].

O processo de aprendizado é chamado de treinamento e, para que isso ocorra, um conjunto de dados deve ser fornecido como entrada para o algoritmo de ML. Padrões

[2]Agregador de diversas bases de dados de publicações acadêmicas: https://www.scopus.com/.

são extraídos desses dados e usados para construir um modelo preditivo capaz de fazer observações precisas em relação a dados nunca vistos [12]. As principais técnicas de aprendizado, as quais diferem entre si dada a presença ou ausência de rótulo nos dados, são: aprendizado supervisionado, semi-supervisionado e não supervisionado.

### 2.2.2. Redes Neurais Artificiais

Entre as diversas abordagens de ML, vale destacar as Redes Neurais Artificiais – *Artificial Neural Networks* (ANN), que são uma evolução dos métodos estatísticos tradicionais. As ANNs foram inspiradas em como sistemas biológicos processam informações, e são constituídas por nós chamados de neurônios e um conjunto de arestas que interliga esses nós formando uma rede [13]. Por sua vez, um grupo de neurônios forma uma camada e existem três tipos diferentes de camadas, que são: de entrada, oculta e de saída. A Figura 2.3 mostra um exemplo simples de uma rede neural com 7 neurônios: 2 na camada de entrada, 4 na camada oculta e 1 na camada de saída.

As ANNs são treinadas por meio de uma sequência de interações de *feedforward* e *backpropagation*. No processo de *feedforward*, os neurônios da camada oculta, utilizando uma função de ativação, ajustam seus pesos de modo que a rede neural tenha a melhor precisão na interpretação da saída, dada uma determinada entrada. Na camada de saída, calcula-se o erro, dado pela diferença entre a saída da rede neural e o valor desejado. O erro é propagado até a camada de entrada pelo processo de *backpropagation*. Essa sequência de passos é repetida diversas vezes até que o desempenho esperado para o modelo seja alcançado [14].

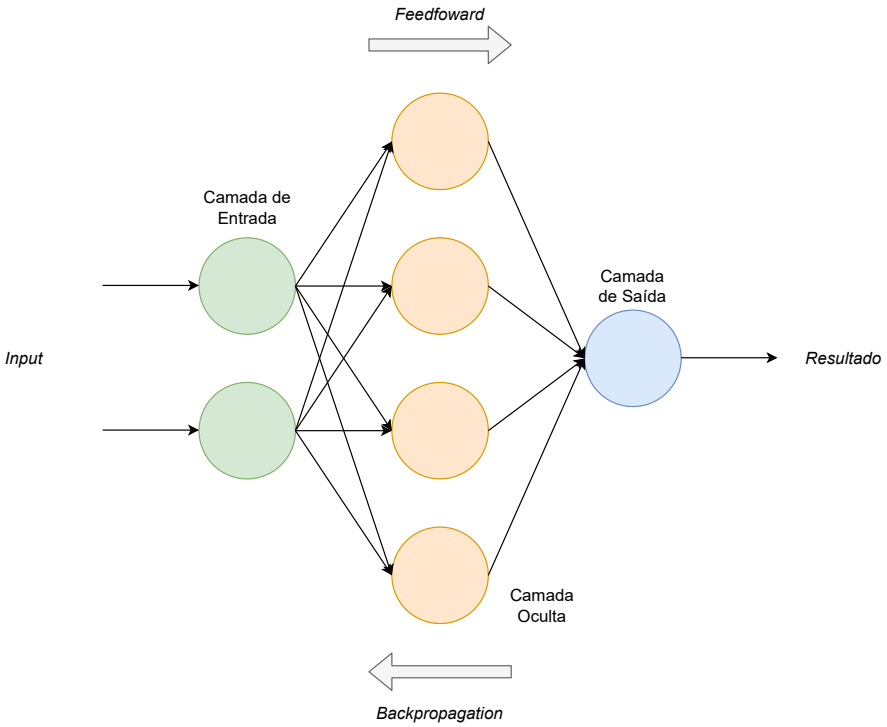


Figura 2.3: Exemplo de uma ANN simples, com neurônios dispostos em três camadas, sendo uma de entrada, uma oculta e uma de saída [Autores].

### 2.2.3. Aprendizado Profundo

Atualmente, grandes volumes de dados são gerados diariamente a uma velocidade sem precedentes e pelas mais variadas fontes (e.g. saúde, governo, financeiro, redes sociais, etc). Isso se deu graças a tecnologias emergentes, como, por exemplo, a Internet das Coisas – *Internet of Things* (IoT) – e Computação em Nuvem – *Clouding Computing* (CC) [15]. Diante desse cenário, desenvolveu-se a disciplina de *Big Data*, que lida com conjuntos de dados massivos, de difícil armazenamento e complexos de serem tratados e analisados por *softwares* tradicionais de processamento de dados [16].

Dado o crescimento no volume de dados gerados e o aumento no poder de processamento dos computadores modernos, os estudos dos algoritmos de Aprendizado Profundo – *Deep Learning* (DL) –, prosperaram. DL é um tipo de ANN e o termo "profundo" no nome desse método refere-se ao uso de várias camadas ocultas na rede neural, o que permite que o processamento de dados seja feito de uma forma não linear, que se traduz em um ganho de acurácia [17], como pode ser visto na Figura 2.4.

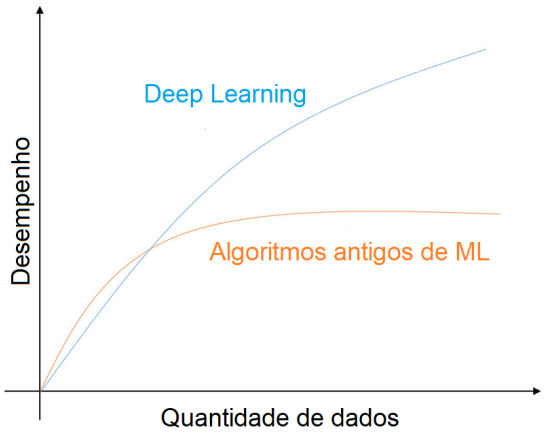


Figura 2.4: Diferença de desempenho dos algoritmos de DL em relação a algoritmos tradicionais de ML, com o aumento da quantidade de dados [Adaptado de Zahangir *et al.* [18]].

Nos nossos experimentos, utilizaremos uma Rede Neural Convolucional – *Convolutional Neural Network* (CNN) –, que é um tipo de algoritmo de DL utilizado no processamento de imagens. CNNs recebem como entrada um vetor de imagens, atribuem pesos às várias características observadas e classificam-na em diferentes classes. A arquitetura típica de uma CNN, como observada na Figura 2.5, é composta por três tipos básicos de camadas, que são: (*i*) convolucional, responsável por extrair recursos de baixo nível; (*ii*) *pooling*, usada para reduzir a complexidade do modelo; e (*iii*) totalmente conectado, responsável por produzir uma interpretação do que foi observado pelo modelo.

Em suma, DL pode alcançar maior precisão devido ao número de camadas ocultas adicionadas à rede e ao grande volume de dados utilizados no treinamento. No entanto, esse ganho de desempenho possui um *trade-off*: o alto poder computacional necessário para processar tamanho volume de informações [20]. Modelos de DL chegam a possuir milhões de parâmetros, como no caso do VGG16 [21], um modelo para classificação de

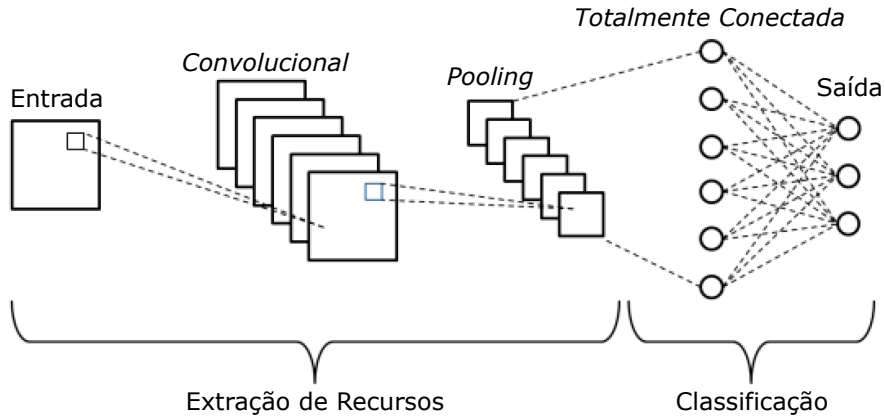


Figura 2.5: Exemplo simplificado da arquitetura de uma CNN com as camadas convolucionais, *pooling* e totalmente conectada [Adaptado de Phung *et al.* [19]].

imagem pré-treinado que analisa 138 milhões de parâmetros para chegar a um resultado.

### 2.2.4. *Frameworks* de Redes Neurais

Existem diversas ferramentas que podem ser utilizadas por cientistas de dados no desenvolvimento de modelos de aprendizado de máquina. Essas ferramentas são interessantes, pois permitem que os especialistas utilizem a Computação de Alto Desempenho – *High Performance Computing* (HPC) –, para desenvolver *softwares* de IA. Um exemplo disso são os Kits de Desenvolvimento de Software – *Software Development Kits* (SDKs) – da NVIDIA, o NVIDIA HPC SDK[3] e NVIDIA CUDA[4], que permitem que desenvolvedores utilizem todo o poder de processamento das placas gráficas da NVIDIA para melhorar o desempenho dos seus modelos de ML.

Esses SDKs podem ser utilizados através de *frameworks* que auxiliam os especialistas a desenvolverem soluções de forma facilitada. Alguns exemplos de *frameworks* existentes são: i) Tensorflow, biblioteca *open-source* que pode ser utilizada no desenvolvimento e treinamento de modelos de ML [22]; ii) PyTorch, uma biblioteca de aprendizado de máquina *open-source* desenvolvida pelo Facebook [23]; iii) Keras, API desenvolvida em *Python*, que permite o desenvolvimento de modelos de DL [24].

### 2.2.5. Operações de Aprendizado de Máquina (MLOps)

Como o conjunto de treinamento é finito e frequentemente novos dados (que modificam as propriedades estatísticas dos conjuntos) surgem, nem sempre é possível garantir um alto desempenho dos algoritmos. Essa perca de desempenho, que muitas vezes se traduz na perca de acurácia dos modelos, é um fenômeno chamado de desvio de conceito – *concept drift* [25].

Assim, é necessário que os modelos sejam re-treinados e implantados novamente

[3]Conjunto abrangente de compiladores, bibliotecas e ferramentas utilizadas no desenvolvimento de aplicações HPC: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/nvhpc.

[4]API que permite que processamento de informações seja feito pelas Unidades de Processamento Gráfico – *Graphical Processing Units* (GPU): https://catalog.ngc.nvidia.com/orgs/nvidia/containers/cuda.

em produção de tempos em tempos, garantindo assim a sua eficiência. Isso é possível graças à utilização de Operações de Aprendizado de Máquina – *Machine Learning Operations* (MLOps) –, que é a união entre ML e a cultura DevOps [26]. MLOps fornece um conjunto de práticas que incentiva a colaboração entre cientistas de dados e engenheiros de DevOps, garantindo que a transição entre treinamento e implantação do modelo em produção seja feito de forma automatizada e contínua [27].

## 2.3. Contêiner

A tecnologia de contêineres foi introduzida em 1979 pela IBM [28]. Nomes diferentes são usados para se referir a contêineres na literatura, incluindo virtualização em nível de sistema operacional e virtualização leve. Um contêiner é uma unidade padrão de *software* que empacota o código e todas as suas dependências para que o aplicativo seja executado de forma rápida e confiável em diferentes ambientes de computação.

Os contêineres fornecem um ambiente isolado em um único *host*, em vez de usar sistema operacional (SO) dedicado como máquinas virtuais [29]. Este isolamento é possível devido ao mecanismo do *kernel Namespace* [30] e *Cgroup* [31]. As tecnologias de virtualização baseadas em *hipervisor* e multitarefas tradicionais têm sido usadas nas últimas duas décadas. Mais tarde, os contêineres entraram no centro das atenções. A tecnologia de contêiner é mais eficiente que a máquina virtual (VM) e, como resultado, muitas organizações estão mudando o ambiente virtualizado para contêineres Linux [32]. As diferenças entre VMs e Contêineres estão principalmente em arquiteturas e funcionalidades. A virtualização baseada em *hypervisor* e em contêiner é mostrada na Figura 2.6.

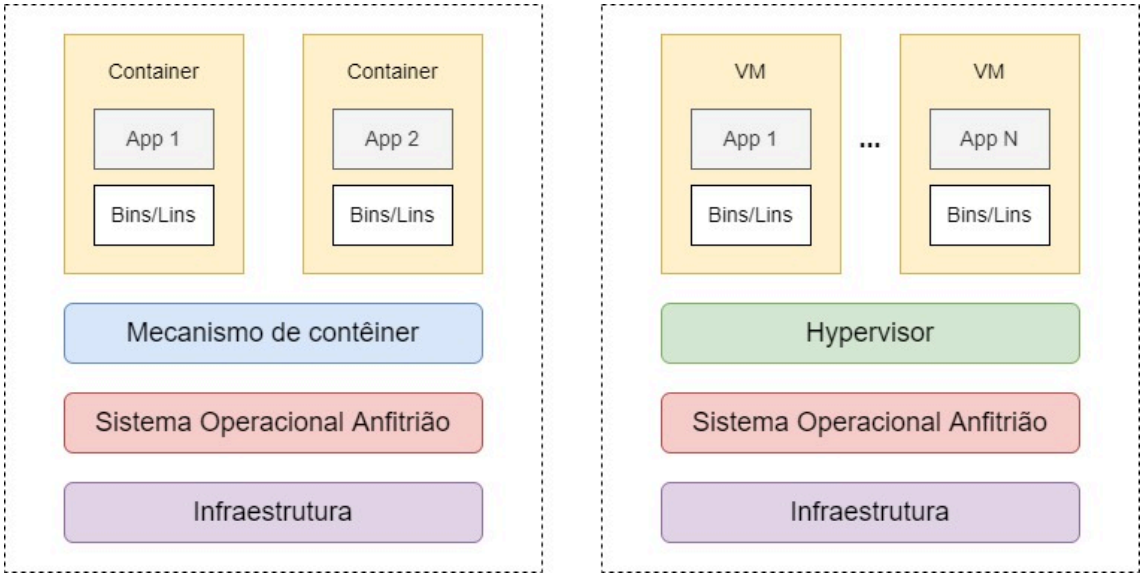


Figura 2.6: Virtualização baseada em *hipervisor* e em contêiner [Autores].

Cada VM contém o Sistema Operacional (SO) convidado, a cópia virtual de *hardware*, o aplicativo e todas as bibliotecas e dependências associadas. Por outro lado, os contêineres virtualizam o sistema operacional e contêm apenas o aplicativo, suas bibliotecas e os arquivos executáveis necessários. Os contêineres virtualizam a camada de

software acima do sistema operacional, onde as VMs virtualizam o *hardware* (memória, CPU, rede, armazenamento). Além disso, os contêineres são mais rápidos, portáteis, leves e eficientes. Um possível motivo para essas vantagens é que os contêineres não contêm SO convidado para todas as instâncias. Assim, os contêineres podem reduzir o consumo de recursos do sistema operacional do *host*. A tecnologia de contêiner e as VMs têm algumas características em comum, como portabilidade, escalabilidade e ciclo de vida de desenvolvimento de software aprimorado [33]. No entanto, o contêiner oferece benefícios significativos sobre as VMs, incluindo peso reduzido, maior eficiência de recursos, maior produtividade do desenvolvedor, melhor balanceamento de carga e tempo de início reduzido [34].

Os contêineres melhoram duas principais desvantagens das VMs [35]:

1. Os contêineres compartilham o mesmo *kernel* do sistema operacional e podem compartilhar recursos enquanto cada VM precisa de sua própria cópia;
2. Os contêineres podem ser iniciados e parados quase instantaneamente, enquanto as VMs precisam de um tempo considerável para iniciar [34].

Os contêineres também provaram ser mais eficientes do que as VMs para alguns aplicativos, tais como microsserviços, porque são leves e não exigem uma cópia completa do sistema operacional para cada imagem. No entanto, os contêineres ainda precisam de um *kernel* totalmente funcional que seja compartilhado entre diferentes contêineres. Além disso, o *design* de microsserviço ressalta a importância dos contêineres de estado efêmero, em que qualquer persistência de dados vai para outro armazenamento de dados ou serviço. Os contêineres efêmeros diferem de outros contêineres porque não possuem garantias de recursos ou execução e nunca serão reiniciados automaticamente. Os contêineres são considerados a maneira padrão de implantar microsserviços na Nuvem [36].

Um estudo comparativo de contêineres e virtualização é fornecido em [37], onde são investigados os conceitos de contêiner, vantagens, desvantagens e soluções. Os desafios da adoção, orquestração e segurança da tecnologia de contêiner são abordados em [38] [39] para tecnologias de contêiner na Nuvem. Além disso, a avaliação de desempenho entre tecnologias de virtualização e contêiner também são investigadas em [40] [41]. Outro trabalho de pesquisa analisa o desempenho de uma máquina virtual convencional e o contrasta com contêineres, e tira conclusões sobre qual é o ideal para usar para as necessidades desejadas [42].

Docker[5], LXC[6] e RKT[7] são exemplos de gerenciadores de contêiner. Muitos estudos se concentram no Docker porque é o ambiente de tempo de execução de contêiner predominante.

### 2.3.1. Docker

O Docker é a tecnologia de contêiner mais usada e popular devido aos contêineres de alto desempenho gerenciados diretamente pelo *kernel* do *host*. O Docker é uma plataforma de

[5]https://www.docker.com
[6]https://linuxcontainers.org/
[7]https://www.redhat.com/pt-br/topics/containers/what-is-rkt

conteinerização gratuita e de código aberto amplamente usada em empresas do setor e na Nuvem. É a base do Kubernetes, Google *Cloud* e é compatível com a Nuvem Amazon AWS. A plataforma Docker permite empacotar aplicativos em contêineres. Os contêineres contêm todo o código-fonte do aplicativo, bibliotecas e dependências necessárias. A plataforma Docker torna a criação, implantação, operação e atualização de contêineres mais fácil e mais segura contra quaisquer vulnerabilidades [43].

Um conceito interessante no qual o Docker se baseia é o que chamamos de *copy-on-write*. Muitos sistemas de arquivos, tais como Btrfs, *Device Mapper* e AuFS, suportam esse modelo *copy-on-write*, que é chamado de sistema de arquivos de união por desenvolvedores do *kernel*. Essencialmente, o que acontece é que você constrói camadas de sistemas de arquivos. Portanto, todo contêiner do Docker é construído sobre o que chamamos de imagem. A imagem do Docker é como um sistema de arquivos pré-configurado que contém uma camada muito fina de bibliotecas e binários necessários para fazer seu aplicativo funcionar, e talvez o código do aplicativo e alguns pacotes de suporte [44].

Os contêineres do Docker têm o Dockerfile, que contém instruções da interface de linha de comando (CLI), especifica as tarefas iniciais e cria as imagens do Docker [45]. A imagem é somente leitura e contém todo o código-fonte executável, bibliotecas e dependências para fornecer um ambiente conveniente para criar contêineres Docker. As imagens do Docker são feitas de camadas e novas camadas são criadas na parte superior se alguma alteração for feita usando um comando específico do Docker. As imagens podem ser desenvolvidas a partir do zero, bem como podem ser extraídas do Docker Hub[8]. O Docker Hub é um repositório público que contém imagens do Docker predefinidas e os contêineres do Docker são as instâncias ativas, em execução e executáveis de imagens do Docker. Usando o Docker-Compose, é possível executar vários contêineres Docker usando um arquivo YAML[9]. Este arquivo especifica quais serviços estão associados a quais contêineres e, portanto, permite gerenciar vários contêineres [45].

## 2.4. Função como Serviço

A Computação em Nuvem é uma tendência crescente no mercado tecnológico e proporciona um acesso onipresente, via rede e sob demanda a um conjunto de recursos computacionais configuráveis [46]. Os serviços de Computação em Nuvem são ofertados em diversos níveis de abstração, e alguns dessas principais abordagens de abstração são comparadas na Figura 2.7, onde as camadas em azul são abstraídas pelo usuário e de responsabilidade do provedor, já as camadas em amarelo são configuradas pelo usuário [47]. Nesta figura, temos:

- **Infraestrutura como Serviço**: É conhecida também como *Infrastructure as a Service* (IaaS). Esse modelo oferta recursos básicos como o armazenamento, a rede e os servidores;
- **Plataforma como Serviço**: O termo em inglês é *Platform as a Service* (PaaS). Ela oferece um conjunto de ferramentas para o desenvolvimento, operação e gerenciamento de aplicações;

[8]https://hub.docker.com/
[9]https://yaml.org/

- **Função como Serviço**: Essa abordagem permite executar códigos sem a necessidade de uma configuração complexa das ferramentas.

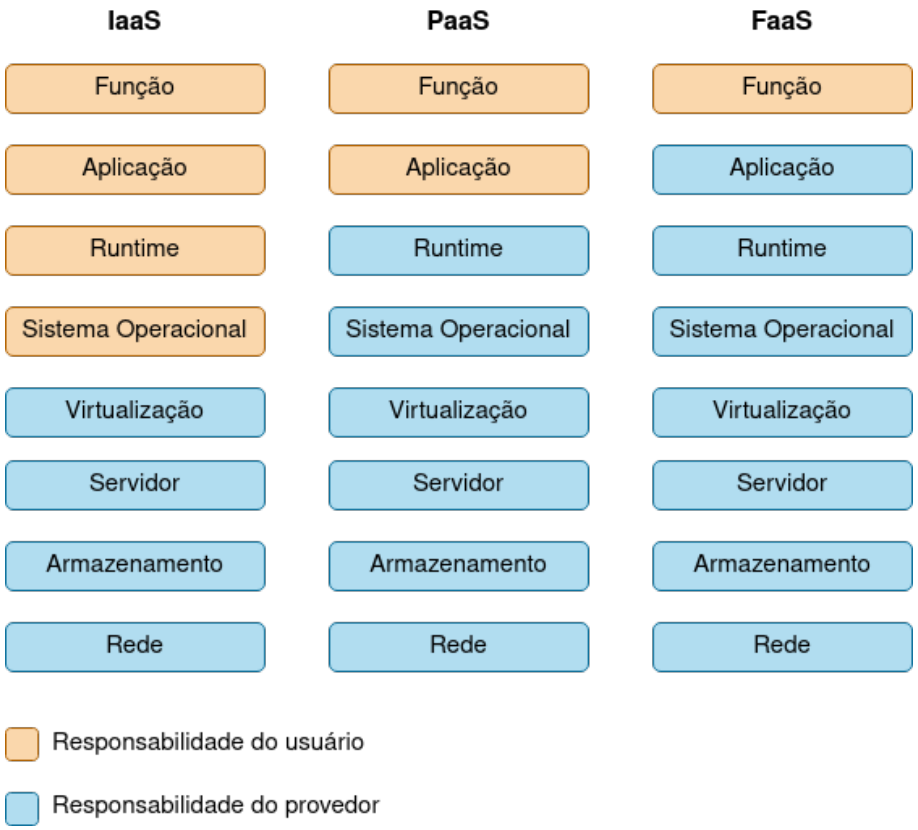


Figura 2.7: Comparação entre os serviços de Nuvem IaaS, PaaS e FaaS [Adaptada de S. K. R *et al.* [47]].

Ao longo do curso, iremos focar na FaaS, que é uma forma de computação sem servidor -– *serverless computing* —, no qual os desenvolvedores escrevem os códigos que serão executados em contêineres gerenciados por uma plataforma [48]. *Serverless* refere a diversos serviços (e.g. computação, banco de dados, mensageria) que têm a configuração, o gerenciamento e o faturamento invisíveis ao usuário final. Dentre esses serviços, a Função como Serviço se concentra em aplicações que não guardam estado e são orientadas a eventos, e os contêineres dessas aplicações só são executados quando solicitados por um evento [49].

### 2.4.1. Funcionamento da FaaS

Esse modelo de computação introduz uma forma diferente de cobrança por recurso em Nuvem, trazendo uma maior granularidade. Diferente das máquinas virtuais, na quais o usuário paga pelos recursos reservados mesmo quando o aplicativo não está em execução, com a FaaS o usuário só precisa pagar pelos recursos que foram utilizados durante a execução da função [50]. Então os provedores implementam o modelo iniciando uma instância da aplicação quando um certo evento ocorre e, ao final dessa execução, a instância é removida. O modelo de preço dessa abordagem depende da memória alocada às funções e do tempo de CPU para executá-las [51].

Essa natureza leve das funções, além de reduzir drasticamente o custo de implantação [52] e de manutenção, muda significativamente a atuação das equipes de desenvolvimento e operações. No desenvolvimento tradicional, essas equipes precisam gerenciar explicitamente suas máquinas virtuais. Com esse modelo *serveless*, os desenvolvedores podem se concentrar no código da aplicação, enquanto o provedor de Nuvem lida com o provisionamento, manutenção e dimensionamento da infraestrutura [53].

A FaaS pode ser útil para diversos tipos de aplicações (e.g. web, Internet das Coisas), até mesmo para produtos de Inteligência Artificial. No pipeline – fluxo de desenvolvimento e entrega – de aplicações de Aprendizado de Máquina, por exemplo, o treinamento do modelo e a inferência dos dados podem ser provisionados em funções [54]. Apesar dessa aplicabilidade, a abordagem apresenta vantagens e desvantagens que devem ser avaliadas antes de seu uso. A seguir listamos alguns desses aspectos.

### 2.4.2. Benefícios

Alguns dos principais pontos favoráveis ao uso de Função como Serviço são:

- **Gerenciamento simplificado**: Não há necessidade de gerenciar os servidores [48], mas apenas realizar configurações básicas como permissões, gatilhos e quantidade de CPU;
- **Maior foco no desenvolvimento**: Com níveis menor de infraestrutura para gerir, as equipes de desenvolvimento conseguem um maior tempo para se concentrarem na codificação das aplicações [50];
- **Economia nos custos de aplicações com tempo ocioso**: Em aplicações que possuem períodos de inatividade, é possível reduzir o custo de provisionamento em até 77,08% substituindo as máquinas virtuais por FaaS como mostram os autores M. Villamizar *et al.* [55];
- **Dimensionamento elástico**: As funções possuem a capacidade de dimensionar automaticamente, de forma independente e sem a necessidade de um ajuste complexo de regras, comuns em outros modelos de provisionamento [49].

### 2.4.3. Limitações

A Função como Serviço também possui suas desvantagens dependendo da necessidade dos usuários. Alguns desses pontos negativos são:

- **Aumento dos gastos de aplicações com carga de trabalho alta**: Em aplicações de uso frequente, sem tempo ocioso, o custo de uma FaaS pode ser três vezes maior que de uma máquina virtual [56];
- **Pouco controle**: Para regras de negócio que determinam um maior controle sobre a infraestrutura, a FaaS não é uma boa solução, pois oferece menos opções de gerenciamento de infraestrutura [53];
- **Dependência do provedor**: Cada provedor possui sua forma de configurar as funções, o que pode dificultar a migração das aplicações entre os provedores [53].

Como podemos observar, a FaaS possui muitos atrativos, mas não se aplica para todas as arquiteturas de *softwares*. Por exemplo, se uma aplicação deve estar 24 horas em execução todos os dias, esse modelo de Nuvem pode não ser aconselhável. Então antes de adotar a FaaS ou qualquer outro modelo, é interessante levantar os requisitos da aplicação, analisar a carga de trabalho, a necessidade de gerenciamento, a capacidade e o conhecimento da equipe de desenvolvimento, entre outros fatores.

### 2.4.4. AWS Lambda

Os principais provedores de Nuvem oferecem FaaS, tais como o AWS Lambda da Amazon[10], Cloud Functions da Google[11], Azure Functions da Microsoft [12] e IBM Cloud Functions da IBM[13]. O AWS Lamda foi o primeiro serviço de FaaS oferecido no mercado em 2014, teve uma grande adesão em 2016 [51] e continua sendo uma das principais ferramentas de FaaS.

Para realizar a parte prática deste curso, iremos utilizar a solução da Amazon. Essa ferramenta possui uma infraestrutura de alta disponibilidade e seu provedor administra os recursos computacionais, realiza a manutenção do servidor, gerencia o provisionamento e a escalabilidade automática, e monitora os *logs* (registro) das execuções das funções. O Lambda tem suporte para códigos em Java, Go, PowerShell, Node.js, C#, Python e Ruby, além possibilitar o acionamento das funções a partir de 200 serviços da própria AWS e de aplicações de Software como Serviço – *Software as a Service* (SaaS) [57].

#### 2.4.4.1. Casos de uso

Comumente o fluxo de funcionamento do Lambda possui a estrutura apresentada na Figura 2.8. No fluxo, uma ação de um serviço (A) aciona a função, o código é executado e, ao final da execução, o resultado é enviado para outro serviço (B).


Figura 2.8: Fluxo comum de uso do Lambda [Autores].

O Lambda pode ser empregado em vários casos de uso, e o fluxo citado se encaixa em alguns casos como:

- **Processamento de dados em escala**: Pode-se criar integração com serviços da AWS, como o Amazon Simple Storage Service (Amazon S3)[14], o Amazon Dy-

[10] AWS Lambda: https://aws.amazon.com/pt/lambda/
[11] Cloud Functions: https://cloud.google.com/functions
[12] Azure Functions: https://docs.microsoft.com/en-us/azure/azure-functions/
[13] IBM Cloud Functions: https://cloud.ibm.com/functions/
[14] Serviço de armazenamento de objetos da AWS: https://aws.amazon.com/s3/

namoDB[15] e o Amazon Kinesis[16], para processamento de arquivo e de *streaming* (transmissão de dados);

- **Execução de *backend* (processo interno) de aplicações**: É possível provisionar *backend* de aplicações *web*, móveis, de IoT, entre outras.

#### 2.4.4.2. Recursos

Para gerenciar seus serviços, a AWS proporciona o console *web*[17] e a Interface da Linha de Comando – *Command Line Interface* (CLI) – da AWS[18]. Além disso, as funções Lambda também podem ser configuradas e versionadas através de *softwares* de automatização como o Terraform[19] e o Serverless Framework[20], que possuem suporte ao provedor de Nuvem da Amazon.

Com essas ferramentas, é possível usufruir de uma série de recursos proporcionados pelo Lambda. Alguns dos principais recursos são:

- **Tolerância a falhas**: O Lambda estabelece sua computação em várias zonas de disponibilidades, que são *datacenters* (centro físico de processamento) com energia e rede em diversos locais do mundo. Isso possibilita alta disponibilidade das funções e elimina janelas de manutenção com tempo de inatividade do serviço;

- **Implantação como imagens de contêiner**: O Lambda é compatível com a implantação de funções utilizando imagens de contêiner, o que facilita a criação de aplicações com dependências não nativas do Lambda;

- **Escalabilidade automática**: Sendo um dos benefícios da FaaS, o Lambda dimensiona as funções de acordo com as solicitações recebidas. Quando a função é invocada enquanto uma instância sua está em execução, outra instância é criada para atender à solicitação. No fim dessas execuções, as instâncias inativas são congeladas ou interrompidas;

- **Controle de desempenho**: É possível habilitar a simultaneidade provisionada para manter funções já inicializadas, acarretando respostas ao evento em questão de milissegundos;

- **Orquestração de funções**: Com o AWS Step Functions[21], pode-se criar fluxo de trabalho para coordenar várias funções Lambda;

[15] Serviço de banco de dados NoSQL da AWS: https://aws.amazon.com/pt/dynamodb/

[16] Serviço de captura, processamento e armazenamento de transmissões de dados da AWS: https://aws.amazon.com/pt/kinesis/

[17] Console de gerenciamento da AWS: https://aws.amazon.com/console/

[18] AWS CLI: https://aws.amazon.com/cli/

[19] Terraform: https://www.terraform.io/

[20] Serverless Framework: https://www.serverless.com/framework

[21] Serviço de visualização de fluxos de trabalho da AWS: https://aws.amazon.com/step-functions

- **Controle de acesso**: Com o Identity and Access Management (IAM)[22], é possível controlar os usuários e as aplicações que têm acesso aos códigos e recursos das funções do Lambda;
- **URLs de função**: Pode-se atribuir um endereço *web* HTTP à função do Lambda. Com isso, é possível invocar a função por meio de navegadores ou outros clientes HTTP;
- **Modelo de funções**: Estão disponíveis no console da AWS vários esquemas de funções que apresentam um código de exemplo e configurações que demonstram a integração com outros serviços da AWS ou de terceiros;
- **Monitoramento automático**: o Lambda monitora automaticamente as funções enviando relatórios das métricas e as saídas das execuções para o Amazon CloudWatch[23].

#### 2.4.4.3. Conceitos básicos

Para uma melhor compreensão da parte prática, apresentamos alguns termos e conceitos presentes no AWS Lambda:

- **Função**: A função é o recurso central, pois possui as configurações necessárias, dependências e o código que será invocado e executado;
- ***Trigger* (Gatilho)**: O *trigger* é o mecanismo que aciona a execução da função. Esse gatilho pode vir de diversos serviços da AWS como Amazon S3, o Amazon Simple Queue Service (Amazon SQS)[24] ou o Amazon EventBridge[25];
- **Evento**: O evento é um documento JSON que vem através do acionamento da função e é consumido pelo código. O conteúdo do evento pode apresentar informações personalizadas dependendo do serviço de origem do gatilho;
- **Arquiteturas de conjunto de instruções**: As arquiteturas determinam o tipo de processador que será utilizado para executar o código da função. A AWS oferece a arquitetura ARM de 64 bits para o processador AWS Graviton2 (`arm64`) e arquitetura x86 de 64 bits para processadores baseados em x86 (`x86_64`);
- ***Runtime* (Ambiente de execução)**: O *runtime* é um ambiente de execução isolado e específico para a linguagem do código do usuário;
- **Pacote de implantação**: É possível implantar uma aplicação na função Lambda utilizando um arquivo Zip ou uma imagem de contêiner que possua o código e suas dependências. No primeiro caso, o Lambda oferece o sistema operacional e o ambiente de execução. Já para o contêiner, esses itens devem estar presentes na imagem;

[22] Serviço de controle de acesso aos recursos da AWS: https://aws.amazon.com/iam/
[23] Serviço de monitoramento da AWS: https://aws.amazon.com/pt/cloudwatch/
[24] Serviço de mensageria da AWS: https://aws.amazon.com/pt/sqs/
[25] Barramento de eventos da AWS: https://aws.amazon.com/pt/eventbridge/

- ***Destination* (Destino)**: Ao terminar a execução da função, o Lambda pode acionar outros serviços que são configurados através do recurso de destino.

## 2.5. Prática

O objetivo principal desta prática é provisionar uma aplicação de AI na FaaS. Para isso, desenvolvemos usando Python um Classificador que indica se uma floresta está sofrendo incêndio ou não através de reconhecimento de imagens. O Classificador é provisionado em uma função no AWS Lambda. Para acioná-la e armazenar as imagens das florestas, utilizamos o Amazon S3. Também criamos imagens Docker que são guardadas no Amazon Elastic Container Registry (Amazon ECR)[26]. Além disso, o gerenciamento de acesso aos serviços citados acima é realizado através do IAM.

Na Figura 2.9, podemos ver o fluxo de funcionamento da aplicação. No fluxo, temos o Coletor de Imagens que envia a imagem da floresta para o *bucket* do S3. Esse envio do arquivo aciona a execução da função Lambda com o Classificador. Ao final da execução, o resultado da análise da imagem é enviado para o Serviço de Mensagens.

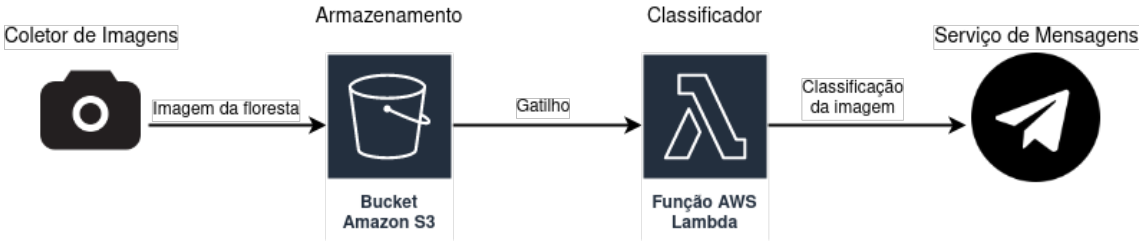


Figura 2.9: Arquitetura do serviço de classificação desta prática [Autores].

Com esse objetivo em mente, apresentamos a seguir algumas configurações básicas e trechos de códigos necessários para utilizar o AWS Lambda. Alguns trechos foram baseados na documentação da AWS[27], e todo o código desenvolvido para a prática está disponível na Internet[28].

### 2.5.1. Usando o AWS Lambda

Nesta seção, descrevemos o passo-a-passo para criar funções padrões em Python e com contêiner Docker no Lambda. Para isso, utilizamos o console *web* e o AWS CLI para a configurar os serviços de forma manual.

#### 2.5.1.1. Criação de função padrão

Para criar uma função padrão no console *web*, basta ir à página funções – *Functions - Lambda*[29] – e clicar em "Create function" (Criar função). Isso direciona o usuário para uma tela de configuração, onde deve-se digitar o nome, a linguagem e a arquitetura do

[26]Registro de contêiner da AWS: https://aws.amazon.com/ecr/
[27]AWS Lambda - Guia do desenvolvedor: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
[28]Repositório da prática online: https://github.com/sayonarasantos/wscad2022-ai-aws-lambda
[29]Functions - Lambda: https://console.aws.amazon.com/lambda/home/functions

processador da função. Ao clicar em “Create function“ dessa segunda tela, é configurada uma função com código simples da linguagem escolhida e um *role* de execução que possui permissão de escrita dos resultados das execuções no Amazon CloudWatch. Na Figura 2.10, vemos a criação de uma função chamada “myFirstFunction” em Python 3.8 com arquitetura `x86_64`.

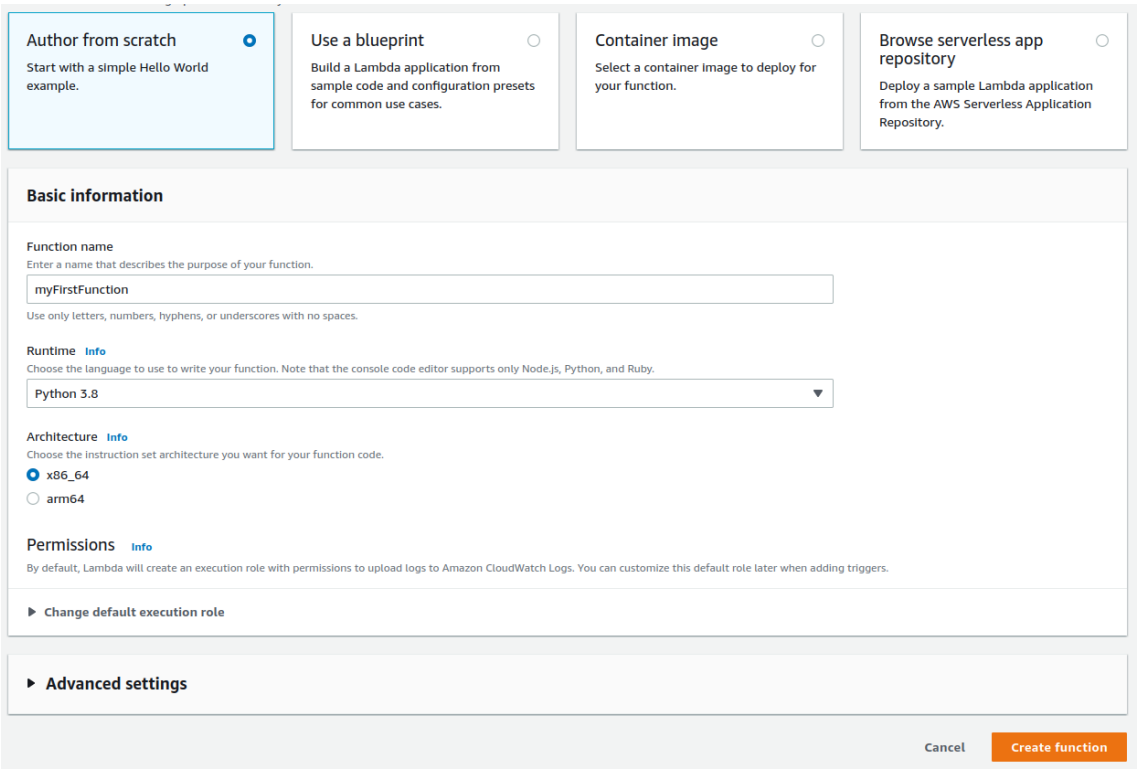


Figura 2.10: Tela de criação de uma função AWS Lambda padrão [Autores].

### 2.5.1.2. Teste de função

Depois de criar a função, pode-se testá-la utilizando os exemplos de eventos fornecidos pela AWS. Para isso, deve-se clicar no botão “Test” (Testar) na aba “Code” (Código) na página da função, que abrirá uma tela para configurar um evento de teste. Nessa tela, é necessário selecionar “New event” (Novo evento), digitar o nome, escolher o “Template” (Modelo) e salvar essa configuração clicando em “Save” (Salvar). A Figura 2.11 mostra a configuração de um evento de teste privado chamado “Hello” e baseado no modelo “hello-world”.

Depois desse processo, basta clicar em “Test” novamente, que a função será executada recebendo o evento de teste. Após a execução, a saída será apresentada na aba “Execution result” (Resultado da execução) e no CloudWatch.

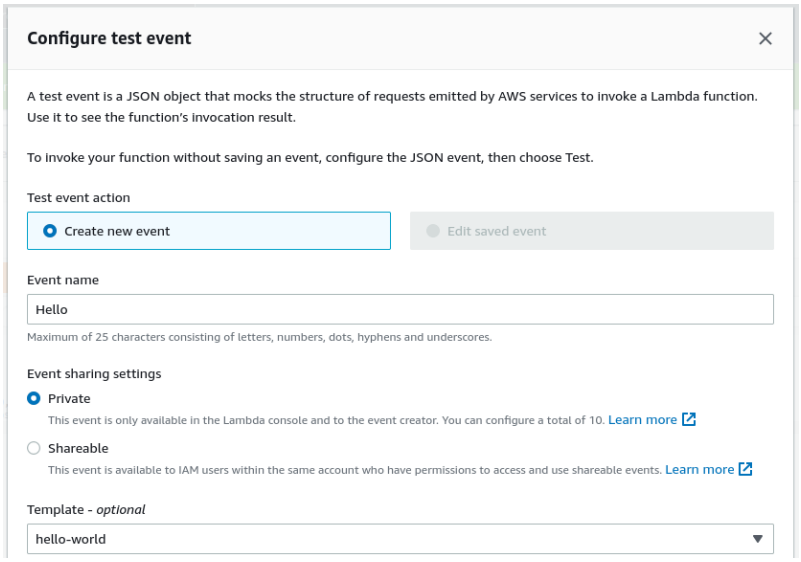


Figura 2.11: Tela de configuração do evento de teste [Autores].

#### 2.5.1.3. Monitoramento de função

Como citamos na seção 2.4, o Lambda monitora as funções enviando métricas e *log* para o CloudWatch. Na aba "Monitor" (Monitorar) da página da função, é possível ver esses *logs* das execuções e gráficos dessas métricas (e.g. número de invocações, durações da execução, contagem de erro). Também é possível visualizar essa informações clicando no botão "View logs in CloudWatch", destacado em vermelho na Figura 2.12.

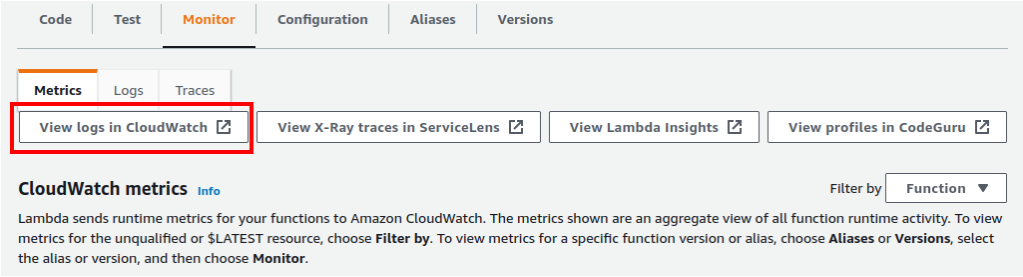


Figura 2.12: Tela de monitoramento com o botão de visualização no CloudWatch em destaque [Autores].

#### 2.5.1.4. Gatilho S3

Um dos tipos de gatilhos disponíveis no Lambda é o acionamento da função através da movimentação dos objetos em um *bucket* do S3. Antes de adicionar esse gatilho, precisa-se criar um *bucket*. Para fazê-lo pelo console *web*, deve acessar a página inicial do S3[30] e clicar em "Create bucket" (Criar *bucket*), que direciona o usuário à tela de criação. Nessa tela, é necessário definir um nome exclusivo, escolher a região, o "Object Ownership" – a

[30] S3 *home*: https://console.aws.amazon.com/s3/

propriedade dos objetos que serão enviados – e os “Bucket settings for Block Public Access” (Configurações de *bucket* para o Bloqueio de Acesso Público), e clicar em “Create bucket” (Criar bucket).

Com o *bucket* criado, pode-se adicionar o gatilho na página da função, clicando em “Add trigger” (Adicionar gatilho) e selecionando o serviço S3 como a origem do acionamento. Na tela de configuração do gatilho S3, deve-se determinar o *bucket* e o tipo de evento que acionará a função. Também é possível definir um prefixo ou um sufixo dos objetos que participarão do acionamento. Na Figura 2.13, temos um exemplo de criação de um gatilho que invocará a função sempre que um objeto com o nome terminado em “.jpg” (*Suffix .jpg*) for adicionado (*Event type POST*) ao *bucket* “2022-lambda-course”.

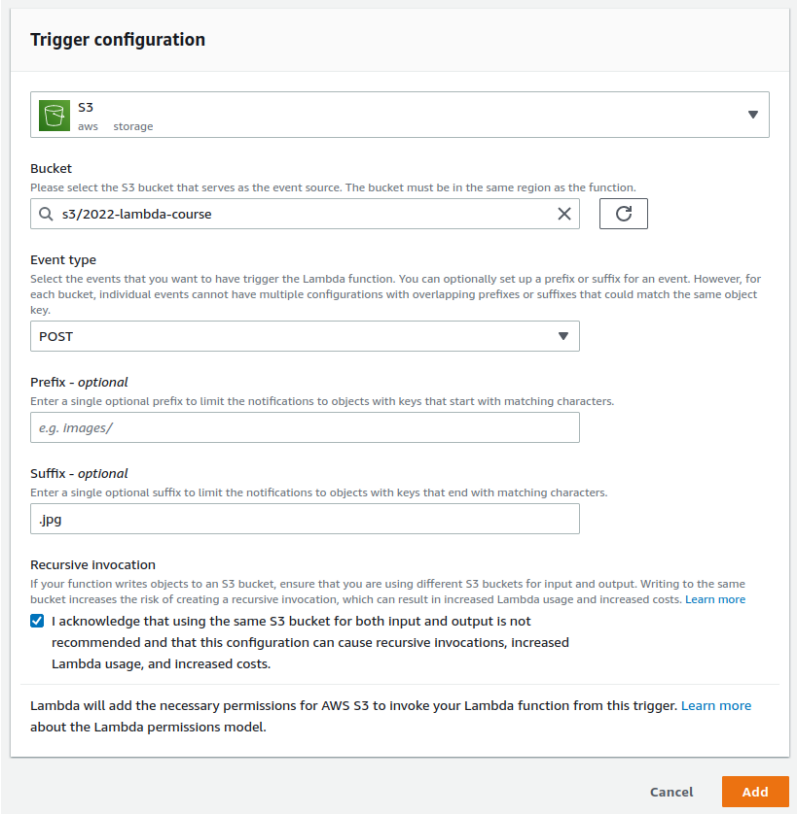


Figura 2.13: Tela de configuração de gatilho S3 [Autores].

Caso a função precise acessar os objetos do *bucket*, deve-se anexar uma política que conceda as permissões desejadas ao *role* da função. Na aba “Configuration” (Configuração) da página da função, o *role* está definido em “Permission” (Permissões) na seção “Execution role” (Função de execução). Para configurar suas permissões, basta clicar na identificação do *role*, que o usuário será redirecionado à página do *role* no IAM. Nessa tela, é possível anexar políticas feitas pela AWS ou customizadas pelo usuário.

#### 2.5.1.5. Criação de função com imagem de contêiner Docker

Outra forma de criar uma função Lambda é utilizando imagem de contêiner. Esse tipo de abordagem é interessante para instalar dependências da aplicação, principalmente pacotes

que não são compatíveis com os formatos suportados pelo Lambda como pacotes Linux. Esse tipo de função carrega uma imagem de contêiner presente no Amazon ECR. A seguir, descrevemos um conjunto de etapas necessárias para configurar um Lambda com contêiner de forma manual.

- **Criação de repositório**: Para criar o repositório dessa imagem, é necessário navegar à página de repositório[31], clicar em "Create repository" (Criar repositório), selecionar a visibilidade em "Visibility settings" (Configurações de visibilidade), digitar o "Repository name" (Nome do repositório), determinar se as *tags* são imutáveis em "Tag immutability" (Imutabilidade de tag), escolher se haverá criptografia em "KMS encryption" (Criptografia KMS) e clicar "Create repository" (Criar repositório).

- **Criação do usuário do IAM**: Para criar a imagem Docker e enviá-la de forma manual, deve-se ter o Docker Engine e o AWS CLI instalados em seu computador e um usuário do IAM com permissões de escrita no ECR. Esse usuário pode ser criado na página inicial do IAM[32], clicando em "Users" e depois em "Add users". Na primeira tela de configuração, precisa-se digitar o nome de identificação do usuário "User name" (Nome do usuário), selecionar o tipo de credencial útil para usar com o AWS CLI – o "Access key - Programmatic access" (Chave de acesso - Acesso programático) – e clicar em "Next" (Próximo). A segunda tela é para anexar uma política que permita ao usuário escrever no ECR, como a política fornecida pela AWS "AmazonEC2ContainerRegistryFullAccess", que fornece acesso administrativo do ECR. Para finalizar, basta clicar em "Next" e "Create user" (Criar usuário) e baixar o arquivo CSV ou copiar o "Access key ID" (Código da chave de acesso) e o "Secret access key" (Chave de acesso secreta).

- **Instalação do Docker e do AWS CLI**: As instalações do Docker e AWS CLI irão depender do sistema operacional utilizado. Portanto, é recomendável seguir a documentação oficial de instalação do Docker Engine[33], instalação do AWS CLI[34] e guia de uso do ECR com AWS CLI[35]. No repositório *online* deste curso, descrevemos o passo-a-passo e os comandos de configuração que utilizamos no sistema operacional Ubuntu.

- **Configuração da imagem**: Com as configurações acima feitas, a próxima etapa é criar um Dockerfile. Nos Códigos 2.1 e 2.2, apresentamos um código Python simples e um Dockerfile que cumprem os requisitos necessários para executar no Lambda, como: (I) o código implementa a interface de execução do Lambda e (II) a imagem é baseada em um sistema Linux. Nesse exemplo, utilizamos uma imagem base Linux (a distribuição Debian bullseye) e implementamos a interface de execução do Lambda em Python (o pacote awslambdaric).

[31] Repositories - ECR: https://console.aws.amazon.com/ecr/repositories

[32] IAM dashboard: https://console.aws.amazon.com/iam/

[33] Instalação do Docker Engine: https://docs.docker.com/engine/install/

[34] Instalar ou atualizar o AWS CLI: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

[35] Uso do Amazon ECR com o AWS CLI: https://docs.aws.amazon.com/AmazonECR/latest/userguide/getting-started-cli.html

Código 2.1: Código Python simples

```
import json


def lambda_handler(event, context):
    print("Received event: " + json.dumps(event, indent=2))
```

Código 2.2: Dockerfile simples para código Python

```
FROM python:3.8-slim-bullseye

WORKDIR /project

RUN apt-get update && \
    apt-get install -y \
    g++ \
    make \
    cmake \
    unzip \
    libcurl4-openssl-dev

RUN pip install --no-cache-dir awslambdaric

COPY app.py .

CMD [ "python", "-m", "awslambdaric", "app.lambda_handler" ]
```

- **Envio da imagem**: Com os códigos escritos, pode-se construir a imagem e enviá-la para o ECR utilizando os comandos:

```
# Construir imagem
$ docker build -t simple-example:v1 .

# Login no Docker
$ aws ecr get-login-password --region REGION | docker login
    --username AWS --password-stdin ACCOUNT_ID.dkr.ecr.us-
    east-1.amazonaws.com

# Marcar imagem para corresponder ao nome no ECR
$ docker tag simple-example:v1 ACCOUNT_ID.dkr.ecr.REGION.
    amazonaws.com/simple-example:v1

# Enviar a imagem
$ docker push ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/
    simple-example:v1
```

- **Configuração da função**: A criação da função pelo console *web* deve ser feita na página das funções do Lambda, clicando em "Create function". Na tela de configuração que será aberta, deve-se escolher o tipo de função "Container image"

(Imagem de contêiner), digitar o nome, escolher a arquitetura da imagem e clicar em "Create function". A Figura 2.14 apresenta a criação da função "containerSimpleExample" baseada em imagem de contêiner "simples-example" com arquitetura `x86_64`.

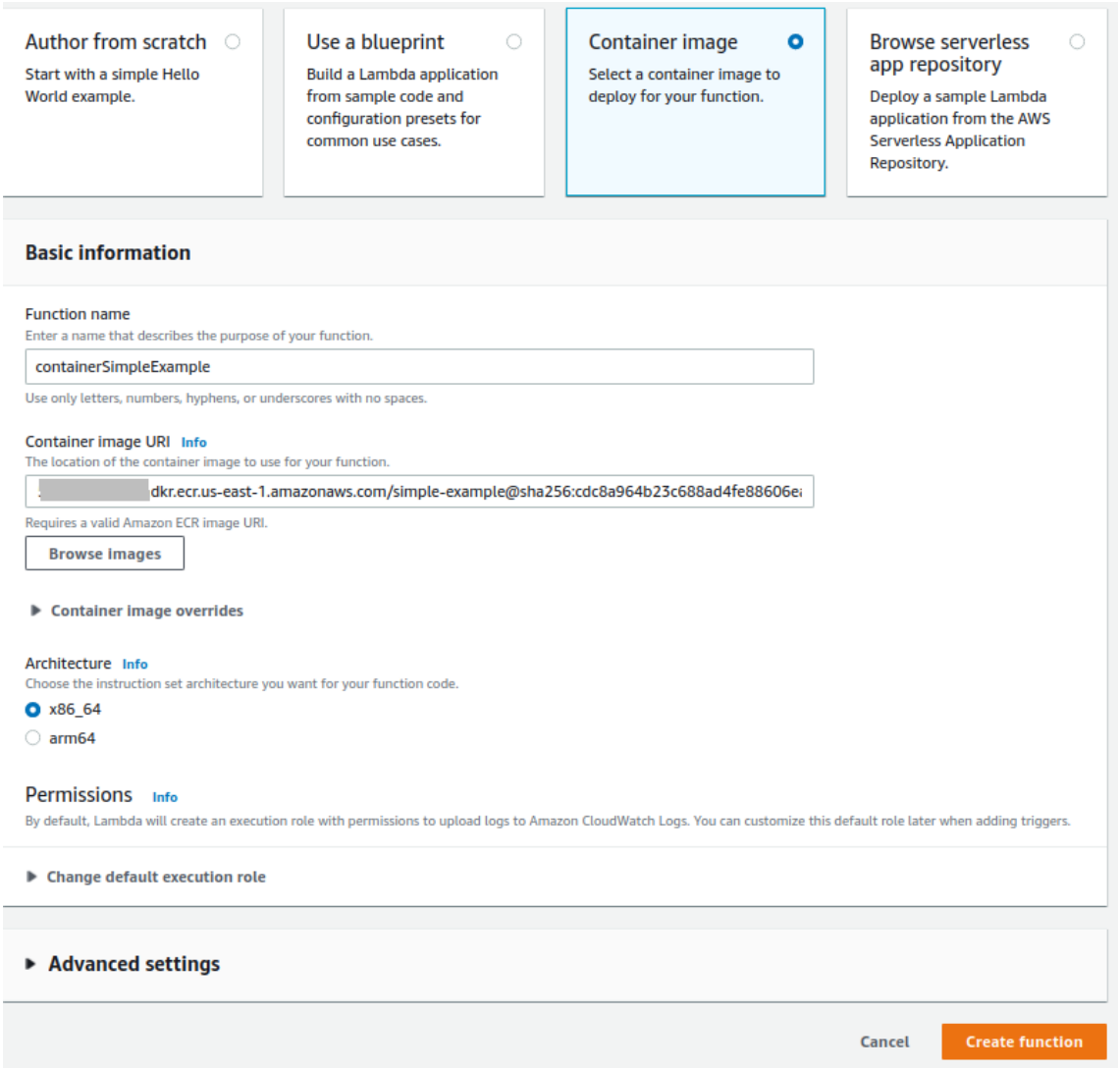


Figura 2.14: Tela de criação de uma função AWS Lambda baseado em imagem de contêiner [Autores].

### 2.5.2. Criação do código de implantação do modelo de ML

Como dito anteriormente, desenvolvemos uma solução de AI capaz de analisar uma imagem e identificar se ela mostra um incêndio florestal ou não. Para isso, utilizamos o *framework* Tensorflow (abordado na Subseção 2.2.4) e o modelo pré-treinado para classificação de imagens MobileNet [58], para criar uma solução de DL.

Para treinar o modelo utilizamos um conjunto de imagens para detecção de incêndios florestais [59]. Esse *dataset* contém 1900 imagens, sendo 950 imagens de florestas sem incêndio (que identificamos com o rótulo 0) e 950 imagens de florestas com incêndio

(cujo rótulo aplicado foi 1). A Figura 2.15 mostra um exemplo de imagens utilizadas no treinamento.

(a) Floresta normal. (b) Floresta com incêndio.

Figura 2.15: Exemplo de imagens encontradas no *dataset* [Autores].

Salvamos o modelo, após ele ter sido treinado e validado, em um arquivo do tipo Formato de Dados Hierárquicos – *Hierarchical Data Format* (HDF)[36] – (formato de arquivo utilizado para armazenar grandes volumes de dados). Esse arquivo será utilizado para carregar o modelo treinado e realizar predições.

O Código 2.3 é uma função Lambda que, quando acionada, carrega o modelo de DL treinado, realiza a classificação de uma imagem e dá como saída sua classificação. Da linha 1 até a linha 6 importamos as bibliotecas utilizadas no programa que nos permitem manipular a imagem analisada, bem como interagir com o Tensorflow e a AWS. Na linha 8 inicializamos uma variável chamada `LABELS` utilizada para traduzir a saída do modelo, e na linha 13 instanciamos a variável `s3`, um objeto do tipo `bot3.client()`, utilizada para interagir com os recursos da AWS.

Código 2.3: Código Lambda utilizado na predição das imagens.

```
1  import boto3
2  import tensorflow as tf
3
4  from io import BytesIO
5  import numpy as np
6  from PIL import Image
7
8  LABELS = {
9      0: "With Fire",
10     1: 'Without Fire'
11 }
12
13 s3 = boto3.client("s3")
```

[36]Gravação de modelos no formato HDF5: https://www.tensorflow.org/guide/keras/save_and_serialize.

```
14
15
16 def lambda_handler(event, context):
17
18     model = tf.keras.models.load_model("./model.h5")
19
20     bucket_name = event["Records"][0]["s3"]["bucket"]["name"]
21     key = event["Records"][0]["s3"]["object"]["key"]
22
23     file_byte_string =\
24         s3.get_object(
25             Bucket=bucket_name, Key=key)["Body"].read()
26
27     image = Image.open(BytesIO(file_byte_string))
28     image = image.resize((224, 224))
29
30     image = image - np.mean(image)
31
32     input_tensor =\
33         np.array(
34             np.expand_dims(image, axis=0), dtype=np.float32)
35
36     output_data = model.predict(x=input_tensor, verbose=0)
37
38     output_data = np.squeeze(output_data)
39     result = np.argmax(output_data, axis=-1)
40
41     print(f"Prediction: {LABELS[result]}")
42     print(f"Probability: {output_data[result]:.2f}")
```

A função Lambda, definida na linha 16, recebe como parâmetro as variáveis `event` (que contém informações úteis para a função processar) e `context` (que traz informações sobre o contexto onde a função está sendo executada). Na linha 18 carregamos o modelo treinado utilizando a função do Tensorflow `load_module()` que recebe como parâmetro o endereço para o arquivo do modelo salvo. Nas linhas 20 e 21 obtemos o nome do *bucket* onde a imagem analisada foi salva e a chave de segurança utilizada para recuperar essa imagem, respectivamente.

Na linha 23 utilizamos a função `get_object()` do objeto instanciado `s3` para recuperar a imagem que será analisada. Para isso, precisamos passar o nome do *bucket* onde a imagem foi salva e a chave de acesso. Ambas informações foram recuperadas nas linhas 20 e 21.

Da linha 27 a 30 realizamos um pré-processamento na imagem. Isso é necessário, pois o modelo espera receber uma imagem que possua um tamanho de 224×224 *pixels* e cujo valor médio dos *pixels* tenha sido subtraído de cada canal da imagem. A Figura 2.16 mostra um exemplo de como as imagens ficam após o pré-processamento.

Na linha 32 inicializamos a variável `input_tensor` com a imagem pré-processada.

(a) Floresta normal.

(b) Floresta com incêndio.

Figura 2.16: Exemplo de imagens pré-processadas [Autores].

Essa variável será passada ao modelo, na linha 36, para que a predição seja. Nas linhas 38 e 39 utilizamos as funções `squeeze()` (que reduz a dimensionalidade de um *array*) e `argmax()` (que retorna o índice do maior valor no *array*) da biblioteca *numpy* para preparar o resultado dado pelo modelo. E por fim, nas linhas 41 e 42 imprimimos na tela o rótulo predito pelo modelo para a imagem e a probabilidade de certeza em relação ao resultado, respectivamente.

### 2.5.3. Provisionamento da aplicação de ML no AWS Lambda

Para provisionar a aplicação ML de classificação descrita na seção anterior, escrevemos o Dockerfile presente no Código 2.4. A imagem base "python:3.8-slim-bullseye" vem do repositório Docker Hub, e seu sistema é um Debian bullseye. Da linha 5 a 11 é feita a instalação das dependências Linux necessárias para compilar a interface de execução do AWS Lambda. Nas linhas 13 e 15 utiliza-se o arquivo "requiments.txt" para instalar os pacotes Python necessários. Nas linhas 17 e 19 copia-se o código Python e o modelo de ML. Por fim, na última linha declara-se o comando que manterá a execução do contêiner da função Lambda.

Código 2.4: Dockerfile para o Classificador

```
1  FROM python:3.8-slim-bullseye
2
3  WORKDIR /project
4
5  RUN apt-get update && \
6    apt-get install -y \
7    g++ \
8    make \
9    cmake \
10   unzip \
11   libcurl4-openssl-dev
12
```

```
13 COPY requirements.txt   .
14
15 RUN pip install --no-cache-dir -r requirements.txt
16
17 COPY model.tflite  .
18
19 COPY app.py  .
20
21 CMD [ "python", "-m", "awslambdaric", "app.lambda_handler" ]
```

## 2.6. Considerações finais

A proposta deste minicurso foi apresentar uma introdução sobre o provisionamento de aplicações de Inteligência Artificial na Nuvem através de Funções como Serviços, que é uma abordagem atual que introduz a capacidade de se executar tarefas sem um servidor. Para esse tipo de tarefa, utilizamos como caso de uso a implantação de uma aplicação de Aprendizado de Máquina no AWS Lambda.

Após uma breve introdução sobre o tema, abordamos AI junto com Redes Profundas e Redes Convolucionais. Em seguida, falamos sobre contêineres, dando um foco na ferramenta Docker, utilizada para empacotar a aplicação executada no Lambda. Depois descrevemos os conceitos da FaaS, além de apresentar o AWS Lambda. Por fim, demonstramos a base técnica para realização da prática do minicurso.

A prática envolve a implantação de um modelo de Aprendizado Profundo capaz de identificar se está ocorrendo ou não um incêndio florestal, através da classificação de uma imagem na nuvem usando o AWS Lambda. Nessa prática guiamos os alunos por todo o processo de configuração e implantação do modelo de IA. Importante destacar que nossa metodologia é genérica e pode ser aplicada em outras aplicações.

Diante do conteúdo exposto, podemos concluir que a FaaS pode ser empregada no *pipeline* de soluções de AI. Contudo, devido às limitações, esse modelo de serviço de Nuvem não será sempre a melhor alternativa para o provisionamento de aplicações. Para determinar se o modelo é útil para aplicação, é necessário levar em consideração alguns pontos como a carga de trabalho e o nível de controle exigido pela empresa contratante.

## Referências

[1] K. K. Mohbey and S. Kumar, "The impact of big data in predictive analytics towards technological development in cloud computing," *International Journal of Engineering Systems Modelling and Simulation*, vol. 13, no. 1, pp. 61–75, 2022.

[2] Oracle, "O que é inteligência artificial (ia)? saiba mais sobre inteligência artificial." https://www.oracle.com/br/artificial-intelligence/what-is-ai/, 2022. Online; acessado em Ago 2022.

[3] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable ai: A review of machine learning interpretability methods," *Entropy*, vol. 23, no. 1, 2021.

[4] B. P. Rimal, E. Choi, and I. Lumb, "A taxonomy and survey of cloud computing systems," in *2009 Fifth International Joint Conference on INC, IMS and IDC*, pp. 44–51, Ieee, 2009.

[5] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 257–262, 2018.

[6] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, *et al.*, "Serverless computing: Current trends and open problems," in *Research advances in cloud computing*, pp. 1–20, Springer, 2017.

[7] T. Elgamal, "Costless: Optimizing cost of serverless computing through function fusion and placement," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 300–312, 2018.

[8] R. Bala, B. Gill, D. Smith, K. Ji, and D. Wright, "Quadrante mágico para infraestrutura em nuvem e serviços de plataforma." https://www.gartner.com/technology/media-products/reprints/AWS/1-271W1OT3-PTB.html, 2021. Online; acessado em Jun 2022.

[9] V. Rajaraman, "Johnmccarthy—father of artificial intelligence," *Resonance*, vol. 19, no. 3, pp. 198–207, 2014.

[10] D. Crevier, *AI: the tumultuous history of the search for artificial intelligence*. Basic Books, Inc., 1993.

[11] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[12] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *International conference on machine learning*, pp. 3145–3153, PMLR, 2017.

[13] A. Krenker, J. Bešter, and A. Kos, "Introduction to the artificial neural networks," *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*, pp. 1–18, 2011.

[14] P. Joshi, *Artificial intelligence with python*. Packt Publishing Ltd, 2017.

[15] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future generation computer systems*, vol. 56, pp. 684–700, 2016.

[16] S. Sagiroglu and D. Sinanc, "Big data: A review," in *2013 international conference on collaboration technologies and systems (CTS)*, pp. 42–47, IEEE, 2013.

[17] R. Alshamrani and X. Ma, *Deep Learning*, pp. 1–5. Cham: Springer International Publishing, 2019.

[18] M. Z. Alom, T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, B. Essen, A. Awwal, and V. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, p. 292, 03 2019.

[19] V. H. Phung and E. J. Rhee, "A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets," *Applied Sciences*, vol. 9, no. 21, 2019.

[20] R. Alshamrani and X. Ma, "Deep learning," *por Laurie A. Schintler y Connie L. McNeely. Cham: Springer International Publishing*, pp. 1–5, 2019.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[22] TensorFlow. URL https://www.tensorflow.org/?hl=pt-br. Acessado: 14/08/2022.

[23] PyTorch. URL https://pytorch.org/. Acessado: 14/08/2022.

[24] Keras. URL https://keras.io/. Acessado: 14/08/2022.

[25] J. Zenisek, F. Holzinger, and M. Affenzeller, "Machine learning based concept drift detection for predictive maintenance," *Computers & Industrial Engineering*, vol. 137, p. 106031, 2019.

[26] S. Alla and S. K. Adari, "What is mlops?," in *Beginning MLOps with MLFlow*, pp. 79–124, Springer, 2021.

[27] S. Mäkinen, H. Skogström, E. Laaksonen, and T. Mikkonen, "Who needs mlops: What data scientists seek to accomplish and how can mlops help?," in *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*, pp. 109–112, 2021.

[28] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, "A comparative study of containers and virtual machines in big data environment," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pp. 178–185, IEEE, 2018.

[29] Docker, "What is a container." https://www.docker.com/resources/what-container/, 2021. Online; acessado em Ago 2022.

[30] Namespaces-Overview, "Namespaces-overview of linux namespaces oct.." https://man7:org/linux/man-pages/man7/namespaces:7:html, 2021. Online; acessado em Ago 2022.

[31] C. Namespaces-Overview, "Cgroup namespaces-overview of linux cgroup namespaces oct.." https://www:man7:org/linux/manpages/man7/cgroup_namespaces:7:html, 2021. Online; acessado em Ago 2022.

[32] A. Lingayat, R. R. Badre, and A. K. Gupta, "Integration of linux containers in openstack: An introspection," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 12, no. 3, pp. 1094–1105, 2018.

[33] L. Benedicic, F. A. Cruz, A. Madonna, and K. Mariotti, "Sarus: Highly scalable docker containers for hpc systems," in *International Conference on High Performance Computing*, pp. 46–60, Springer, 2019.

[34] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers," *IEEE wireless communications*, vol. 24, no. 3, pp. 48–56, 2017.

[35] B. M. Abbott, *A security evaluation methodology for container images*. Brigham Young University, 2017.

[36] S. Vaucher, R. Pires, P. Felber, M. Pasin, V. Schiavoni, and C. Fetzer, "Sgx-aware container orchestration for heterogeneous clusters," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 730–741, IEEE, 2018.

[37] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support paas," in *2014 IEEE International Conference on Cloud Engineering*, pp. 610–614, IEEE, 2014.

[38] B. Bermejo, C. Juiz, and C. Guerrero, "Virtualization and consolidation: a systematic review of the past 10 years of research on energy and performance," *The Journal of Supercomputing*, vol. 75, no. 2, pp. 808–836, 2019.

[39] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance overhead comparison between hypervisor and container based virtualization," in *2017 IEEE 31st International Conference on advanced information networking and applications (AINA)*, pp. 955–962, IEEE, 2017.

[40] C. Pahl, "Containerization and the paas cloud," *IEEE Cloud Computing*, vol. 2, no. 3, pp. 24–31, 2015.

[41] S. A. Babu, M. Hareesh, J. P. Martin, S. Cherian, and Y. Sastri, "System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver," in *2014 fourth international conference on advances in computing and communications*, pp. 247–250, IEEE, 2014.

[42] P. R. Desai, "A survey of performance comparison between virtual machines and containers," *Int. J. Comput. Sci. Eng*, vol. 4, no. 7, pp. 55–59, 2016.

[43] P. Liu, S. Ji, L. Fu, K. Lu, X. Zhang, W.-H. Lee, T. Lu, W. Chen, and R. Beyah, "Understanding the security risks of docker hub," in *European Symposium on Research in Computer Security*, pp. 257–276, Springer, 2020.

[44] C. Anderson, "Docker [software engineering]," *IEEE Software*, vol. 32, no. 3, pp. 102–c3, 2015.

[45] Docker, "Dockerfile reference." https://docs.docker.com/engine/reference/builder/, 2021. Online; acessado em Ago 2022.

[46] P. Mell, T. Grance, *et al.*, "The nist definition of cloud computing," 2011.

[47] S. K. R and J. Lakshmi, "Qos aware faas platform," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 812–819, 2021.

[48] I. Red Hat, "Cloud computing." https://www.redhat.com/pt-br/topics/cloud, 2018. Online; acessado em Ago 2022.

[49] I. C. Education, "Faas (function-as-a-service)." https://www.ibm.com/cloud/learn/faas, 2022. Online; acessado em Ago 2022.

[50] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," 2018.

[51] T. Elgamal, A. Sandur, K. Nahrstedt, and G. Agha, "Costless: Optimizing cost of serverless computing through function fusion and placement," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 300–312, 2018.

[52] T. Lynn, P. Rosati, A. Lejeune, and V. Emeakaroha, "A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 162–169, 2017.

[53] B. King, "What is faas? function as a service explained." https://www.digitalocean.com/blog/what-is-faas-function-as-a-service-explained, 2022. Online; acessado em Ago 2022.

[54] M. Chadha, A. Jindal, and M. Gerndt, "Towards federated learning using faas fabric," in *Proceedings of the 2020 Sixth International Workshop on Serverless Computing*, WoSC'20, (New York, NY, USA), p. 49–54, Association for Computing Machinery, 2020.

[55] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano Merino, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, "Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures," *Service Oriented Computing and Applications*, vol. 11, 06 2017.

[56] A. Eivy and J. Weinman, "Be wary of the economics of "serverless"cloud computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 6–12, 2017.

[57] Amazon, "Aws lambda: Execute código sem se preocupar com servidores ou clusters." https://aws.amazon.com/lambda/, 2022. Online; acessado em Ago 2022.

[58] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[59] A. Khan and B. Hassan, "Dataset for forest fire detection," Mendeley Data, V1, 2020. doi: 10.17632/gjmr63rz2r.1.