UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENG. DE CONTROLE E AUTOMAÇÃO

**ALEX TREVISO - 00259921**

# AUTOMAÇÃO DO PROVISIONAMENTO DE INFRAESTRUTURA EM NUVEM PARA IMPLANTAÇÃO DE SISTEMAS

Porto Alegre
2022

**ALEX TREVISO - 00259921**

# AUTOMAÇÃO DO PROVISIONAMENTO DE INFRAESTRUTURA EM NUVEM PARA IMPLANTAÇÃO DE SISTEMAS

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de *Bacharel em Eng. de Controle e Automação* .

ORIENTADOR:

Prof. Dr. Marcelo Götz

Porto Alegre
2022

**ALEX TREVISO - 00259921**

# AUTOMAÇÃO DO PROVISIONAMENTO DE INFRAESTRUTURA EM NUVEM PARA IMPLANTAÇÃO DE SISTEMAS

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Disciplina de TCC do curso *Eng. de Controle e Automação* e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universität Paderborn - Paderborn, Alemanha

Banca Examinadora:

Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universität Paderborn - Paderborn, Alemanha

Prof. Dr. Ivan Müller, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. João Cesar Netto, UFRGS
Doutor pela Université Catholique de Louvain - Louvain-la-Neuve, Bélgica

_____
Prof. Dr. Mário Roland Sobczyk Sobrinho
Coordenador de Curso
Eng. de Controle e Automação

Porto Alegre, outubro de 2022.

# AGRADECIMENTOS

Agradeço aos meus pais, Ilso e Noeli, pelo incentivo e apoio incondicional ao longo de todo percurso e por saber que, independente da distância, sempre estão comigo. Também à meu irmão, Felipe, que, dentre outras coisas, foi minha inspiração nesta jornada.

Agradeço à minha namorada, Julia, por ser, acima de tudo, meu refúgio. Agradeço por todo amor, companheirismo e palavras de incentivo. Ainda, agradeço pela paciência e compreensão nos momentos de angústia.

Aos amigos e colegas de curso, em especial à Igor Diesel, Davi Bobsin e Emerson de Barros, pela parceria de longa data e que cruza fronteiras, e à Eduardo Brezolin, por ter encarado este último semestre comigo.

Aos colegas da disciplina de projetos, em especial à Bruno Cappellari, que, mesmo à distância, não mediu esforços para que o sistema utilizado estivesse funcional.

Ao meu orientador, Prof. Dr. Marcelo Götz, por todo o conhecimento compartilhado, pela orientação de maneira sábia, pela confiança e pelo apoio nos momentos em que precisei.

Agradeço também à Yan Haeffner pelo convite responsável por despertar em mim a curiosidade sobre este assunto, até então, desconhecido. Também, por toda a parceria diária e pela revisão, sempre pontual, deste trabalho.

À todos colegas e amigos de trabalho que fizeram parte de minha formação e contribuíram diariamente em meu desenvolvimento, sempre com discussões produtivas e enriquecedoras.

Também à todos os demais amigos que a UFRGS me proporcionou e que tornaram a jornada mais leve e alegre.

*“Anyone who stops learning is old, whether at twenty or eighty. Anyone who keeps learning stays young."*

Henry Ford

# RESUMO

Devido à crescente demanda por agilidade na disponibilização de infraestruturas consistentes e cada vez mais complexas, a configuração manual de recursos na nuvem muitas vezes resulta em falhas. Com o intuito de aprimorar os processos envolvidos no gerenciamento de infraestrutura, a utilização de ferramentas de automação apresenta-se como solução para redução de possíveis problemas. Desta forma, no presente trabalho, foi implementada a automação do provisionamento de infraestrutura na nuvem voltada para implantação de sistemas. A implementação foi concebida através da utilização da ferramenta Terraform e seu conceito de Infraestrutura como Código, que permite construir, modificar e gerenciar a infraestrutura de maneira segura e confiável. A plataforma de computação na nuvem utilizada foi a Amazon Web Services (AWS) onde, através dos recursos e serviços disponibilizados, foi desenvolvida uma arquitetura com alta disponibilidade e escalonável. Ainda, com o intuito de validar a arquitetura concebida, foi proposta a implantação de um sistema de gerenciamento de ponto eletrônico disponibilizado por meio de uma API HTTP executada em container. Desenvolveu-se também o protótipo de um dispositivo IoT utilizando o microcontrolador ESP8266 NodeMCU, responsável pela aquisição de dados de identificadores RFID através da utilização do módulo NFC PN532 e posterior inserção dos mesmos no sistema. A partir de testes realizados, analisou-se a viabilidade da utilização do Terraform na automação da infraestrutura proposta que, por sua vez, apresentou-se robusta, segura e altamente disponível mesmo em cenários com elevadas cargas de trabalhos devido à sua capacidade de escalabilidade.

**Palavras-chave: Computação em nuvem, Infraestrutura como Código, Terraform, Amazon Web Services, IoT.**

# ABSTRACT

Due to the growing demand for agility in provisioning consistent and complex infrastructures, the manual setting of cloud resources often results in failures. In order to improve the process involved in infrastructure management, the use of automation tools presents itself as a solution to reduce possible problems. Therefore, this work presents the implementation of the automation of a cloud infrastructure provisioning for systems deployment. The implementation was conceived using Terraform tool and its concept of Infrastructure as Code (IaC), which allows building, modifying and managing the infrastructure in a safe and reliable way. The cloud computing platform used was Amazon Web Services (AWS) and an architecture with high availability and scalability was develop using its resources and services. Also, in order to analyze the conceived architecture, the implementation of a working time management system available through an HTTP API running in a container was proposed. A prototype of an IoT device was also developed using the ESP8266 NodeMCU microcontroller, responsible for acquiring data and inserting it into the system. The potential of using Terraform in the automation of the proposed infrastructure was analyzed and from the tests performed, the infrastructure proved to be robust, secure and highly available even in scenarios with high workloads due to its scalability capacity.

**Keywords: Cloud Computing, Infrastructure as Code, Terraform, Amazon Web Services, IoT.**

# SUMÁRIO

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS

| | |
|---|---|
| API | Application Programming Interface |
| ARN | Amazon Resource Name |
| AuFS | Another Union File System |
| AWS | Amazon Web Services |
| AZ | Availability Zone |
| CCA | Curso de Eng. de Controle e Automação |
| CIPS | Cloud Infrastructure and Platform Services |
| CoT | Cloud of Things |
| CPU | Central Processing Unit |
| ECR | Elastic Container Registry |
| ECS | Elastic Container Service |
| GCP | Google Cloud Platform |
| HCL | HashiCorp Configuration Language |
| HSU | High Speed UART |
| HTTP | Hypertext Transfer Protocol |
| IA | Inteligência Artificial |
| IaaS | Infrastructure as a Service |
| IaC | Infrastructure as Code |
| IAM | Identity and Access Management |
| I2C | Inter-Integrated Circuit |
| I/O | Input/Output |
| IoT | Internet of Things |
| IP | Internet Protocol |
| JSON | JavaScript Object Notation |
| LCD | Liquid Crystal Display |
| LXC | Linux Containers |

| | |
|---|---|
| MCU | Micro Controller Unit |
| MISO | Master In Slave Out |
| MOSI | Master Out Slave In |
| NAT | Network Address Translation |
| NFC | Near Field Communication |
| OS | Operating System |
| PaaS | Platform as a Service |
| RAM | Random Access Memory |
| RDS | Relational Database Service |
| RFID | Radio-Frequency Identification |
| RISC | Reduced Instruction Set Computer |
| SaaS | Software as a Service |
| SCL | Serial Clock |
| SCLK | Serial Clock |
| SDA | Serial Data |
| SoC | System-on-a-chip |
| SPI | Serial Peripheral Interface |
| SS | Slave Select |
| SSH | Secure Shell |
| TCP | Transmission Control Protocol |
| TI | Tecnologia da Informação |
| URL | Uniform Resource Locator |
| USB | Universal Serial Bus |
| vCPU | Virtual Central Processing Unit |
| VM | Virtual Machine |
| VPC | Virtual Private Cloud |
| WSN | Wireless Sensor Network |

# 1 INTRODUÇÃO

Presenciamos nos últimos anos a ascensão de um dos maiores avanços da história da computação (DE DONNO; TANGE; DRAGONI, 2019). O surgimento de tecnologias de *cloud computing* representa uma reinvenção nos meios em que serviços de tecnologia da informação (TI) são desenvolvidos, implementados e utilizados (MARSTON et al., 2011). Dentre os benefícios proporcionados com sua utilização, pode-se destacar a redução de custos de infraestrutura, a escalabilidade sob demanda de recursos computacionais, a confiabilidade, segurança e a capacidade de armazenamento de dados na nuvem (GAJBHIYE; SHRIVASTVA, 2014).

De acordo com Manvi e Shyam (2014), os maiores desafios associados com infraestrutura em ambiente *cloud* são o gerenciamento de recursos, o gerenciamento de infraestrutura de redes, interoperabilidade, virtualização e o gerenciamento de dados. Deste modo, a utilização de ferramentas de Infraestrutura como Código (IaC) apresenta-se como uma alternativa capaz de facilitar o processo de provisionamento e gerenciamento de infraestrutura. Dentre as principais ferramentas de IaC, o Terraform destaca-se pela sua eficiência e segurança na criação, configuração, gerenciamento e versionamento de infraestrutura (PRASSANNA; PAWAR; NEELANARAYANAN, 2017).

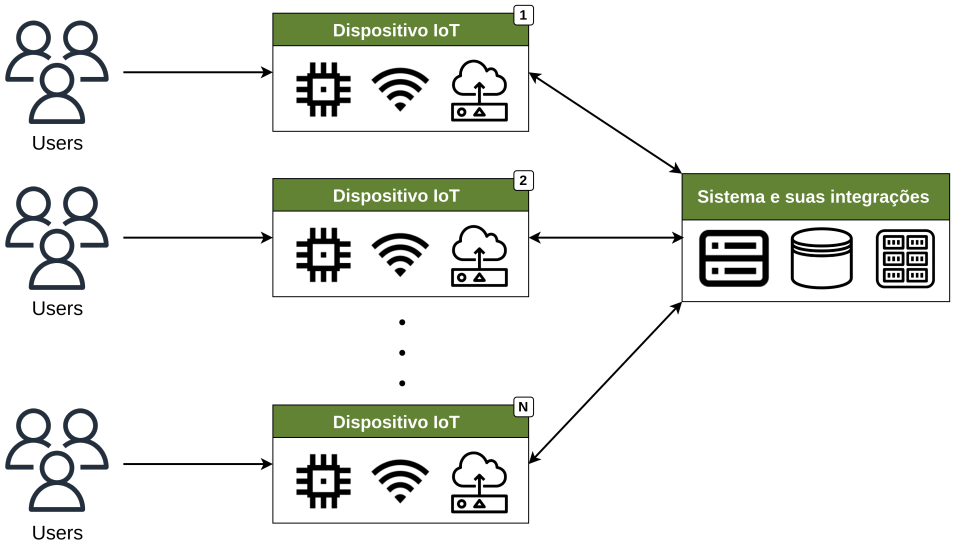
Em um mercado cada vez mais competitivo, a agilidade na disponibilização de novas funcionalidades de *software* e de lançamentos de novos produtos faz-se necessária. Sendo assim, a automação do processo de provisionamento de infraestrutura através da abordagem de IaC, surge como uma estratégia capaz de agilizar os processos de operação na disponibilização da infraestrutura necessária.

Juntamente com a ascenção de tecnologias de *cloud computing*, deparamo-nos com a expansão e crescente desenvolvimento de tecnologias de *Internet of Things* (IoT). Novas aplicações baseadas em IoT encontram-se constantemente sendo desenvolvidas e utilizadas para os mais diversos fins e em diferentes indústrias, com o objetivo de auxiliar no que diz respeito ao monitoramento e gerenciamento de ambientes de trabalho. A partir da integração de sistemas IoT com a tecnologia de *cloud computing*, pode-se realizar a centralização do gerenciamento dos dados provenientes de diferentes dispositivos e localidades em um único lugar com alta disponibilidade (VILELA et al., 2019). Esta integração também permite que serviços sejam entregues por meio de APIs (*Application Programming Interfaces*), tornando os mesmos disponíveis para aplicações IoT e permitindo o desenvolvimento de sistemas inteligentes (MAHMOUD et al., 2018).

Com base na relevância do tema proposto, a motivação deste trabalho busca apresentar e validar uma proposta de solução de automação para a construção de um modelo de arquitetura em ambiente *cloud* para implantação de sistemas, proporcionando a centralização do gerenciamento de dados provenientes de diferentes dispositivos IoT. Para fins de automação do provisionamento da infraestrutura, o desenvolvimento será realizado via

Terraform. Ainda, através da estruturação de uma arquitetura robusta, espera-se assegurar características como resiliência, segurança, alta disponibilidade e escalabilidade. A Figura1 apresenta uma visão da arquitetura geral e suas integrações.

**Figura 1:** *Visão da arquitetura geral proposta e suas integrações.*




Fonte: Elaborada pelo autor.

## 1.1 Objetivos

O presente trabalho tem como objetivo a implementação da automação do processo de provisionamento de infraestrutura em ambiente *cloud* para implantação de sistemas inteligentes, concebendo o desenvolvimento de uma arquitetura escalonável e com alta disponibilidade através da Internet.

O projeto incluíra a elaboração do processo de automação através da utilização de Terraform e seu conceito de Infraestrutura como Código, que permite construir, modificar e gerenciar a infraestrutura de maneira segura e confiável. A plataforma de computação na nuvem utilizada será a Amazon Web Services (AWS), que conta com recursos e serviços gerenciados que possibilitam com que a concepção proposta seja alcançada.

Por fim, propõe-se a análise da arquitetura concebida através da implantação de um sistema de gerenciamento de ponto eletrônico disponibilizado por meio de uma API executada em container Docker. Ainda, propõe-se o desenvolvimento de um protótipo de um dispositivo IoT utilizando o módulo ESP8266 NodeMCU, que será utilizado para correta validação do sistema e suas integrações.

# 2 REVISÃO BIBLIOGRÁFICA

Neste capítulo são apresentados os principais conceitos e ferramentas utilizados para desenvolvimento deste trabalho. Ao longo do mesmo, as vantagens que motivaram suas escolhas são expostas como forma de elucidar, de maneira clara, as decisões de projeto que foram realizadas.

## 2.1 *Cloud Computing*

Segundo Gartner, Inc. ([s.d]), *cloud computing* é a disponibilidade, através da Internet, de recursos computacionais, como armazenamento e a capacidade de processamento, em que o usuário paga apenas pelo serviço utilizado. Alguns relatos históricos dão conta de que a ideia teve seu surgimento por volta da década de 1960, quando Jhon McCarthy, cientista norte-americano conhecido por seu pioneirismo nos estudos sobre inteligência artificial (IA), propôs a ideia de computação por *time-sharing*, que consistia no compartilhamento do tempo ocioso das máquinas entre os usuários. Porém, o termo *cloud computing* teve sua primeira aparição somente em 1997, quando Ramnath Chellappa definiu-o como um novo paradigma em que os limites computacionais seriam determinados pelo custo e não somente pelos limites técnicos (CHELLAPPA, 1997).

O conceito de *cloud computing* baseia-se na contratação de um serviço de armazenamento, gerenciamento e processamento de dados junto a um provedor. Desta forma, faz com que não seja mais necessário lidar com custos, manutenções e outras preocupações de gerir e manter uma infraestrutura física de *hardware* e *software*. Por estes e outros motivos, a parcela do orçamento de empresas dedicada a tecnologias *cloud* cresce anualmente (COLUMBUS, 2020).

No que diz respeito à classificação de *cloud computing*, possui distinção quanto ao seu tipo, sendo estes classificadas em termos de quem gerencia o ambiente. As principais distinções giram em torno dos conceitos de *cloud* pública e *cloud* privada. O mais popular dos tipos de *cloud* é a pública (ou *cloud* externa), sendo esta fornecida por um provedor de serviços *cloud* e compartilhada entre usuários (que possuem níveis de acesso devidamente separados) por meio da Internet. O provedor fica encarregado pelo fornecimento da infraestrutura e de serviços, como hospedagem, manutenção, gestão e segurança dos dados de seus clientes. Visto que se trata de um tipo de servidor compartilhado, apresenta um custo significativamente inferior aos demais. Este modelo destaca-se no cenário e vêm ganhando grande relevância, com ampla aceitação e adoção, devido aos benefícios proporcionados pela vasta quantidade de *data centers* que as grandes empresas fornecedoras do serviço desenvolveram nos últimos anos, proporcionando uma maior liberdade no que diz respeito a dimensionamento de alocação de recursos com baixo custo, visto que o pagamento será realizado apenas pelo seu uso (JIN et al., 2010).

Diferentemente de uma *cloud* pública, a *cloud* privada é ambientada dentro da própria empresa. Resumidamente, é utilizada quando a infraestrutura de servidores e *data centers* tem como intuito atender unicamente uma empresa ou organização. Por apresentar um ambiente mais controlado e, consequentemente, mais seguro, é comumente utilizada por departamentos governamentais e instituições financeiras (JIN et al., 2010).

Além de possuir distintas classificações quanto à seu tipo, conta também com três principais ofertas de serviços: *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS) e *Software as a Service* (SaaS) (GARRISON; KIM; WAKEFIELD, 2012). O serviço de IaaS é focado na disponibilização de recursos de *hardware*, como processamento, armazenamento, rede e recursos básicos de computação através de máquinas e dispositivos virtuais, possibilitando que o usuário armazene dados, instale e execute suas aplicações. No tipo de serviço de PaaS, o cliente pode ter a disposição uma plataforma para desenvolvimento e testes de aplicações em ambiente *cloud*, utlizando APIs e ferramentas fornecidas pelo provedor. No serviço de SaaS, mais comumente utilizado pela população, o acesso à um *software* é oferecido e disponibilizado ao usuários por meio da Internet, não sendo necessário o gerenciamento, instalação ou atualizações do mesmo.

Atualmente existem várias empresas de serviços de nuvem pública disponíveis no mercado. Dentre as empresas que apresentam-se como provedoras disponíveis, as principais plataformas encontradas são: Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, IBM Cloud, Oracle Cloud e Alibaba Cloud.

### 2.1.1 Amazon Web Service

A Amazon Web Service (AWS) é a principal plataforma de serviços de *cloud computing* atualmente no mercado. A plataforma foi fundada pela empresa norte-americana Amazon no ano de 2006, após mais de uma década e milhões de dólares investidos na construção de uma infraestrutura de TI que faz dela hoje uma das maiores plataformas, com centenas de milhares de clientes espalhados pelo mundo (VARIA; MATHEW et al., 2014).

A infraestrutura global da AWS subdivide-se por regiões. Cada região consiste em duas ou mais zonas de disponibilidade (AZs, do inglês *Availability Zones*) onde encontram-se localizados os *data centers* da empresa. Atualmente a infraestrutura conta com 24 regiões geográficas em todo o mundo, com 84 AZs atendendo à 245 países (AWS, 2022).

A vasta infraestrutura global e as vantagens por ela proporcionadas potencializaram a rápida aceitação da AWS como provedora de serviços de computação em nuvem (HASHEMIPOUR; ALI, 2020). Além de seu pioneirismo de ofertas *on-demand*, oferece segurança, escalabilidade e alta disponibilidade. A partir da contratação, a empresa assegura que o tempo de inatividade não exceda seis minutos em um ano.

Ainda, segundo pesquisa recente da Gartner, a AWS é posicionada como líder no chamado quadrante mágico de serviços de infraestrutura e plataforma de nuvem (CIPS). O quadrante mágico é definido pela capacidade de execução dos serviços e pela abrangência da visão da empresa, posicionando a AWS como a plataforma mais inovadora e mais capacitada no que diz respeito à CIPS. A Figura 2 apresenta o quadrante mágico para CIPS.

**Figura 2:** *Quadrante mágico para infraestrutura em nuvem e serviços de plataforma.*


Fonte: (BALA et al., 2021)

A AWS é a plataforma com a maior quantidade de serviços e recursos de nuvem. Oferece serviços voltados desde tecnologias de infraestrutura (computação, armazenamento, redes e banco de dados) até tecnologias emergentes, como *machine learning*, inteligência artificial, *data lakes* e IoT. Os principais serviços utilizados no desenvolvimento deste projeto são apresentados abaixo:

- **Virtual Private Cloud**: A Amazon Virtual Private Cloud (VPC) é o principal serviço de rede privada da AWS. A VPC estabelece uma seção isolada logicamente dentro do ambiente para execução de recursos, definindo como os mesmos estarão distribuídos dentro dos servidores da AWS e, principalmente, como será o acesso de rede das aplicações para redes externas.

- **Elastic Container Service**: O Elastic Container Service (ECS) é o serviço de orquestração de containers totalmente gerenciado da AWS que auxilia na implantação, gerenciamento e escalabilidade de aplicações. Dentre as modalidades de oferta, apresenta como alternativa a opção de computação sem servidor (*serverless*) do AWS Fargate, eliminando a necessidade de configurar e gerenciar planos de controle, nós e instâncias.

- **AWS Fargate**: O AWS Fargate é uma tecnologia disponível que pode ser utilizada com o Amazon ECS para execução de containers. Através da sua utilização, elimina a necessidade de gerenciamento de servidores ou *clusters*. Possibilita que a aplicação seja executada através de container, sendo necessário apenas a especificação dos requisitos de sistema operacional, CPU e memória.

- **Elastic Container Registry**: O Amazon Elastic Container Registry (ECR) é um registro de container totalmente gerenciado da AWS que fornece hospedagem de

alta performance para imagens e artefatos de aplicações. Ainda, possibilita a opção de criação de repositórios públicos ou privados, apresentando uma elavada confiabilidade que auxilia na simplificação dos *workloads* de implantações através de integrações entre ambientes autogerenciados.

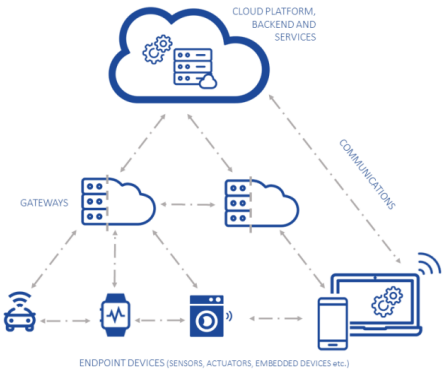
- **Relational Database Service**: O serviço do Amazon RDS (Relational Database Service) apresenta a possibilidade de configurar, operar e escalar diferentes bancos de dados relacionais na nuvem. Através da sua utilização, tarefas administrativas relacionadas a operação de banco de dados, muitas vezes ineficientes e onerosas, são eliminadas. O serviço proporciona a implantação de um banco de dados sem a necessidade de provisionamento de infraestrutura ou qualquer manutenção de *software*, garantindo alta disponibilidade através de implantações multi-AZ (*Multiple Availability Zones*).

## 2.2 *Internet of Things* - IoT

O conceito de IoT, cada vez mais presente no mundo, descreve a rede de dispositivos físicos que incorpora sensores, *software* e outras tecnologias com o objetivo de conectar e trocar dados com outros dispositivos e sistemas através da Internet. As tecnologias utilizadas em IoT incluem *Radio-Frequency Identification* (RFID) e *Wireless Sensor Networks* (WSNs), que possiblitam que informações sejam detectadas a partir de sensores e posteriormente possam ser processadas e utilizadas como instrumento de auxílio na tomada de decisões (FAROOQ et al., 2015).

A rede de dispositivos físicos baseia-se em instrumentos capazes de integrar-se através da Internet. Entretanto, por tratarem-se de dispositivos com recursos limitados, a elevada quantidade de dados gerados não pode ser processada e armazenada localmente. Por isso, a utilização de recursos e serviços *cloud* apresenta-se como alternativa para o armazenamento e processamento dos dados gerados, tornando necessária a realização de integração entre o ambiente *cloud* com o dispositivo IoT (KUMAR; DUBEY; PANDEY, 2021). Essa integração, atualmente, é conhecida como *Cloud of Things* (CoT). A Figura 3 apresenta uma ilustração a respeito do conceito de CoT.

**Figura 3:** *Arquitetura de integração de sistemas IoT e computação em nuvem - CoT.*




Fonte: (SKOULOUDI; FERNÁNDEZ, 2018).

A expansão e crescente desenvolvimento de tecnologias de IoT está diretamente ligada com o serviço de computação em nuvem. Através da sua utilização, as empresas puderam migrar suas aplicações e serviços de armazenamento de dados para recursos virtualizados na *cloud*. O modelo de computação na nuvem também fornece todos os recursos necessários como forma de serviços para aplicações IoT, possibilitando que os usuários possam acessar dados de qualquer dispositivo e a qualquer instante através da Internet (VILELA et al., 2019). A Tabela 1 apresenta a comparação entre as características dos conceitos apresentados.

**Tabela 1:** *Características de IoT, Cloud Computing e CoT.*

| **IoT** | ***Cloud Computing*** | **CoT** |
|---|---|---|
| Possibilidade de implantação de dispositivos em diferentes localidades | Possibilidade de acesso a recursos de qualquer lugar | Capacidade de implantação e acesso a recursos de qualquer lugar |
| Interação com dispositivos físicos | Interação com recursos virtuais | Interação com dispositivos físicos e recursos virtuais |
| Capacidades de armazenamento e processamento restritas | Capacidades de armazenamento e processamento virtualmente ilimitadas | Capacidades de armazenamento e processamento virtualmente ilimitadas |

Fonte: Adaptado de (MAHMOUD et al., 2018).

## 2.3 Automação de Infraestrutura de TI

A automação consiste na técnica onde um sistema elimina a necessidade da interferência humana a partir do emprego de processos automáticos que comandam e controlam mecanismos para o seu próprio funcionamento. Tem sua utilização nas mais diferentes áreas, fazendo-se também presente no que diz respeito à automação de processos de infraestrutura de TI, eliminando a necessidade da alocação e gerenciamento de recursos de maneira manual.

Problemas de disponibilidade de servidores em grandes *data centers* são ocasionados, na grande maioria das vezes, por erros humanos (OPPENHEIMER; GANAPATHI; PATTERSON, 2003). Desta forma, a utilização de procedimentos de automação apresenta-se como solução para a redução de problemas de disponibilidade. Ainda, a partir da implementação de rotinas automatizadas, pode-se ter um ganho de agilidade e uma redução nos custos de infraestrutura nos processos de implantação de *software* (BROWN; HELLERSTEIN, 2005).

### 2.3.1 Infraestrutura como Código

Infraestrutura como código (IaC) é um conceito de automação baseado em práticas do desenvolvimento de *software*, tendo seu foco voltado ao provisionamento e gerenciamento de infraestrutura (MORRIS, 2016). Segundo Nelson-Smith (2013), o conceito de IaC teve seu surgimento em 2006, quando a AWS iniciou o fornecimento de serviços de computação em nuvem. A partir disto, qualquer ideia para aplicações *web* pôde ser implementada em um intervalo de tempo consideravelmente menor quando comparado ao cenário atual da época. Essa mudança de paradigma impulsionou o crescimento de pequenas empresas de TI e elevou a demanda por rotinas comuns do processo de operações. Devido a isto, começaram a surgir as primeiras ferramentas de gestão de infraestrutura, que visavam facilitar a administração de ambientes complexos.

A partir da sua utilização, inúmeros benefícios foram incorporados ao processo de

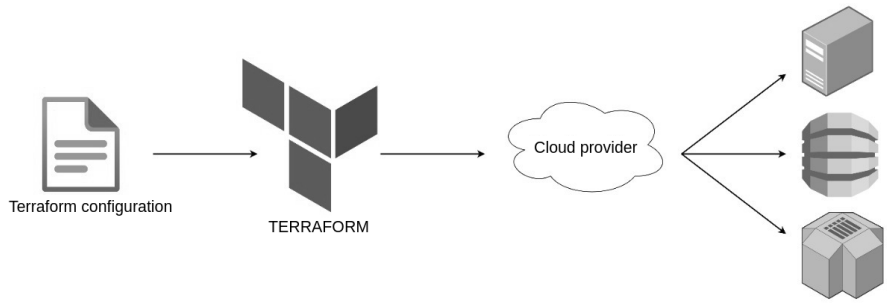
implantação de sistemas: redução de custos, aumento na velocidade das implantações, redução de erros humanos, melhoria na consistência da infraestrutura e a eliminação de desvios de configuração. As principais ferramentas de IaC encontradas atualmente no mercado são o Ansible, Chef, Puppet e Terraform, tendo sido utilizado o Terraform na realização do projeto.

### 2.3.2 Terraform

O Terraform é uma ferramenta *open source* criada pela HashiCorp que possibilita o provisionamento e o gerenciamento de infraestrutura através de diferentes *cloud providers* utilizando a abordagem de infraestrutura como código (BRIKMAN, 2019). Através de sua utilização, é possível construir, modificar e versionar a infraestrutura de maneira eficiente e segura. A ferramenta é desenvolvida em Go e teve sua primeira versão disponibilizada em 2012. Devido à sua simplicidade de operação, o Terraform ganhou elevada popularidade, contando, atualmente, com mais de 1600 colaboradores na sua comunidade do *GitHub*.

A ferramenta utiliza sua própria linguagem declarativa de configuração, denominada HCL (*HashiCorp Configuration Language*) (SABHARWAL; PANDEY; PANDEY, 2021). Alternativamente, também pode-se utilizar arquivos de configuração em formato JSON (*JavaScript Object Notation*). A linguagem declarativa HCL apresenta como grande vantagem a possibilidade de trabalhar com diferentes provedores de serviços de *cloud computing*. Por se tratar de uma linguagem declarativa, apresenta seu foco no estado final resultante, contrapondo a abordagem de linguagens procedurais, onde comandos são executados na ordem em que são escritos. Desta forma, a partir da utilização de arquivos de configurações, pode-se executar planos de ação e aplicar modificações na infraestrutura de interesse. A Figura 4 apresenta o fluxo de funcionamento do Terraform.

**Figura 4:** *Fluxo de funcionamento do Terraform.*



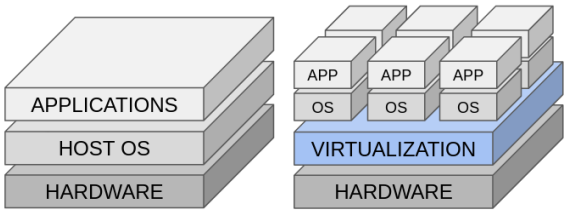

Fonte: Elaborada pelo autor.

Um dos componentes de um projeto Terraform é o arquivo de estado. O arquivo de estado é o responsável pelo armazenamento do estado atual da infraestrutura, mapeando possíveis modificações na mesma a partir da comparação do estado atual com a configuração executada. Apresenta também a possibilidade do desenvolvimento de módulos, que são agrupamentos de recursos que visam proporcionar uma melhor organização ao projeto. Ainda, uma vez desenvolvido, um módulo pode ser compartilhado por diferentes projetos, fornecendo uma maior consistência e ajudando a diminuir a probabilidade de erros.

## 2.4 Virtualização

O processo de virtualização consiste na execução de instâncias virtuais de um sistema em uma camada de abstração, possibilitando que elementos do *hardware* como processamento, memória e armazenamento sejam divididos em múltiplas instâncias virtuais (SIDDIQUI; SIDDIQUI; KHAN, 2019). As instâncias virtualizadas ficam situadas na camada acima da virtualização, conforme apresentado na Figura 5.

**Figura 5:** *Diferença entre as camadas encontradas em sistemas operacionais tradicionais e sistemas operacionais virtualizados.*




Fonte: Elaborada pelo autor.

Uma instância virtual, comumente denominada VM (*virtual machine*), executa seu próprio sistema operacional (OS) e funciona como um computador independente (IBM, 2019). Atualmente, a virtualização é uma prática padrão aplicada na arquitetura de TI, sendo um elemento fundamental para a mesma. Alguns benefícios da utilização do processo de virtualização são a possibilidade do compartilhamento de recursos, a redução de custos operacionais, a redução de *downtimes* e a facilidade de gerenciamento e controle de recursos.

Do ponto de vista de virtualização de servidores, existem três principais tipos: total, para-virtualização e em nível de sistema operacional (JAIN; CHOUDHARY, 2016). As técnicas de virtualização total e para-virtualização fazem uso de *hypervisor*, que consiste em uma camada de abstração de software acima do hardware e é responsável pela criação e execução de VMs. Em contrapartida, a virtualização em nível de sistema operacional não requer um *hypervisor*. Neste caso, a virtualização acontece à nível do próprio *host* OS a partir da possibilidade da existência de várias instâncias isoladas compartilhando do mesmo kernel. O processo de virtualização por container, também denominado containerização, é um exemplo de virtualização em nível de sistema operacional.

### 2.4.1 Docker container

Docker é uma ferramenta de código aberto que possibilita, de maneira simples, o empacotamento de aplicações e todas as suas dependências dentro de um container (SCHEEPERS, 2014). A ferramenta utiliza a tecnologia de AuFS (*Advanced Multi-Layered Unification Filesystem*) como sistema de arquivos do container, assegurando uma economia no consumo de espaço de armazenamento e memória.

A portabilidade do Docker é uma de suas principais características quando comparado a outros mecanismos de containers, como LXC. Diferentemente do LXC, nativo para execução em sistemas Linux, a utilização de Docker possibilita que aplicações containerizadas sejam portadas para qualquer *host* OS que possua a plataforma Docker instalada. Dentre os principais elementos que constituem a ferramentas, podemos destacar:

- *Docker image*: uma imagem nada mais é do que um artefato gerado a partir de um arquivo contendo todas as informações necessárias para sua criação, denominado *Dockerfile*. O arquivo *Dockerfile* apresenta em seu corpo uma imagem de sistema operacional base utilizada, como Ubuntu, Python, Alpine, Dotnet, etc. Na sequência, sobre a imagem base, a aplicação é construída e todas suas possíveis dependências instaladas a partir de comandos que são executados dentro do container. Por fim, tem-se uma imagem Docker que consiste em um pacote executável que contará com tudo que for preciso para execução da aplicação.

- *Docker container*: um container nada mais é do que uma instância que executa uma imagem Docker. Consiste em um ambiente isolado que executa seu próprio OS (exceto o kernel), fornecendo um ambiente virtual que permite definir limitações de recursos como CPU, memória, I/O, entre outros.

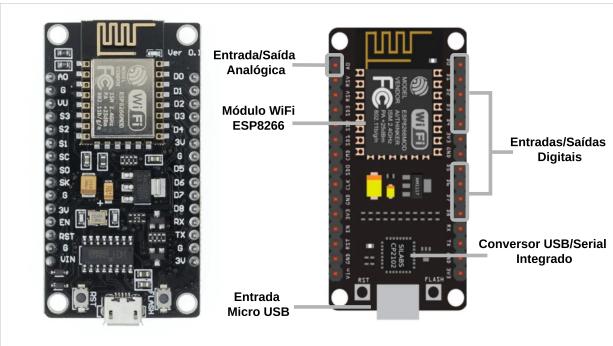
## 2.5 Plataforma NodeMCU

Atualmente no mercado encontramos diversas plataformas de desenvolvimento, baseadas em distintos microcontroladores, voltadas ao desenvolvimento de dispositivos IoT. Dentre elas, o NodeMCU apresenta-se como uma plataforma de código aberto da família ESP8266, microcontrolador do fabricante chinês Espressif. É composto por um chip controlador ESP8266 (ESP-12E), que é um SoC (*system-on-a-chip*) com a pilha do protocolo TCP/IP integrada, permitindo o acesso a rede WiFi.

O módulo ESP8266 NodeMCU vem programado com um *firmware* que permite a prototipagem de seu produto voltado à IoT de maneira simplificada, não sendo necessário a integração com outro dispositivo para colocá-lo em funcionamento. Desta forma, apresenta-se como um sistema microcontrolado com possibilidade de comunicação sem fio, além de possuir interface Micro-USB. Devido à estas e outras características, a plataforma NodeMCU é atualmente uma das plataformas IoT de código aberto mais utilizadas pela comunidade científica (VOGEL et al., 2020).

Apresenta processador de arquitetura RISC de 32 bits operando a 80MHz e memória para *cache*, melhorando a performance do sistema. Possui ainda 4Mb de memória *flash*, 20Kb de memória RAM, 16 pinos de GPIO e 1 entrada analógica com resolução de 10 *bits*. A Figura 6 apresenta o ESP8266 NodeMCU e seus principais componentes.

**Figura 6:** *ESP8266 NodeMCU e seus principais componentes.*




Fonte: Elaborada pelo autor.

# 3 MATERIAIS E MÉTODOS

A fim de desenvolver uma solução que possibilite que sistemas sejam implantados através de sua utilização, faz-se necessário a concepção de uma infraestrutura completa. Deste modo, a infraestrutura deve contemplar aspectos como configuração de arquitetura de redes, arquitetura de sistemas e de banco de dados. Ainda, deve-se atentar para segurança e robustez da mesma, a fim de evitar possíveis falhas. Neste capítulo é apresentado o projeto de arquitetura e também o desenvolvimento do processo de automação do provisionamento da mesma.

## 3.1 Proposta de Arquitetura de Rede

A configuração de rede em ambiente *cloud* apresenta-se como parte fundamental do projeto de arquitetura de implantação de sistemas. É através dela que serviços tem a capacidade de serem executados em ambientes privados e isolados. Ainda, assegura alta disponibilidade através do conceito de implantação multi-AZ, apresentando a possibilidade de criação de sub-redes que são executadas em distintas zonas de disponibilidade. O serviço voltado à provisionamento da mesma dentro da AWS, responsável pela criação não somente da rede mas também dos demais recursos que a compõe, é o Amazon Virtual Private Cloud.

Como componente fundamental da arquitetura, define-se uma VPC, que consiste em uma rede virtual privada logicamente isolada. Conforme descrito anteriormente e visando assegurar a alta disponibilidade dos recursos, propõe-se também a criação de sub-redes em quatro distintas zonas de disponibilidade. Deparamo-nos com dois diferentes tipos de sub-redes, sendo estes pública e privada. A principal diferença entre os tipos diz respeito ao tráfego de rede encaminhado a Internet pública, sendo este realizado por meio de um *Internet Gateway* no caso de uma *subnet* pública. Em contrapartida, uma *subnet* privada restringe toda e qualquer conexão com a Internet pública, sendo necessário a utilização de dispositivos NAT. Dessa forma, faz-se necessário a configuração de um recurso de *Internet Gateway* e de quatro NAT *Gateways*, associando um NAT para cada *subnet*.

Uma vez definidas as *subnets* e os demais recursos, é imprescindível a utilização de *route tables*, responsáveis pelo direcionamento do tráfego de rede baseado em um conjunto de regras. De maneira geral, é necessário a criação de *route tables* associadas as *subnets* públicas e as *subnets* privadas. Como regras associadas às mesmas, define-se que o tráfego cujo destino encontra-se dentro da mesma rede deve ser encaminhado localmente e o tráfego cujo destino encontra-se em qualquer outro endereçamento deve ser encaminhado via *Internet Gateway*, no caso das *subnets* públicas, ou via NAT *Gateway*, no caso das *subnets* privadas.

A Tabela 2 apresenta a listagem dos recursos utilizados na arquitetura proposta e a

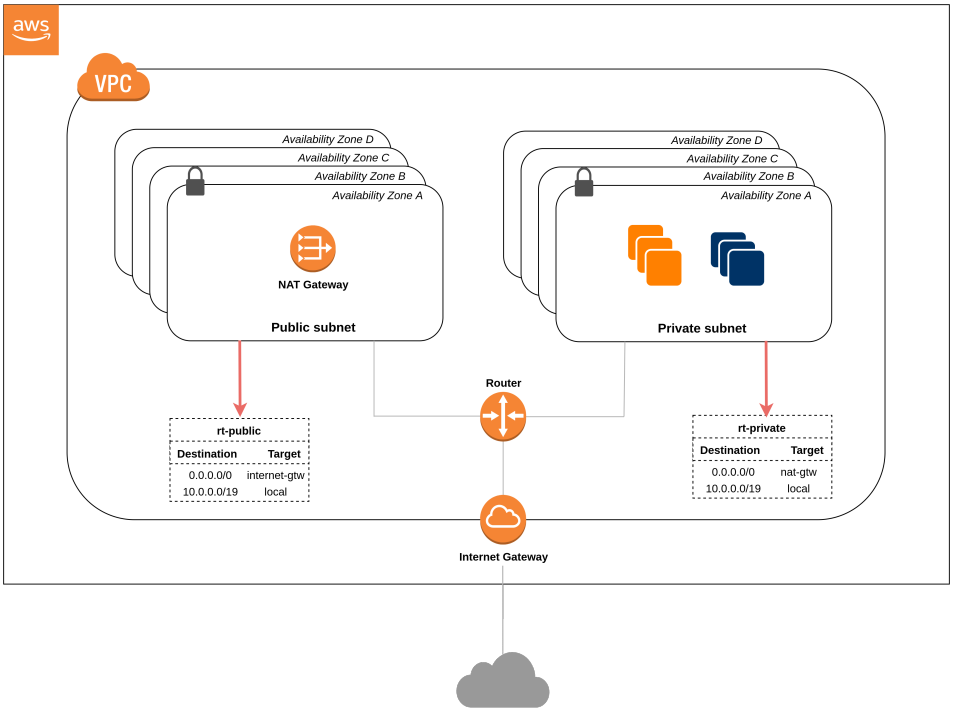
Figura 7 o diagrama do modelo proposto.

**Tabela 2:** *Recursos utilizados na arquitetura de rede proposta.*

| Recurso | Quantidade | Descrição |
|---|---|---|
| *Virtual Private Cloud* | 1 | Rede virtual privada logicamente isolada |
| *Subnet* | 8 | Subdivisões lógicas da VPC |
| *Internet Gateway* | 1 | *Gateway* responsável pela comunicação entre a VPC e a Internet |
| *NAT Gateway* | 4 | Comunicação de dispositivos de uma *subnet* privada a serviços fora da VPC |
| *Route Tables* | 2 | Roteamento do tráfego de rede |

Fonte: Elaborada pelo autor.

**Figura 7:** *Modelo de arquitetura de rede proposto.*




Fonte: Elaborada pelo autor.

## 3.2 Infraestrutura do Sistema e suas Integrações

O principal componente da arquitetura de execução de sistemas é o serviço de computação sob demanda. Para escolha do serviço de computação, levou-se em conta a possibilidade de utilização do mecanismo AWS Fargate, que é um mecanismo que tem como principal vantagem a possibilidade de execução de aplicações sob demanda e sem a necessidade de manutenção e gerenciamento de servidores de infraestrutura. O mecanismo possui suporte ao Amazon ECS, serviço gerenciado pela AWS para gerenciamento de

containers. Desta forma, definiu-se como serviço de execução a utilização do AWS ECS, na modalidade Fargate.

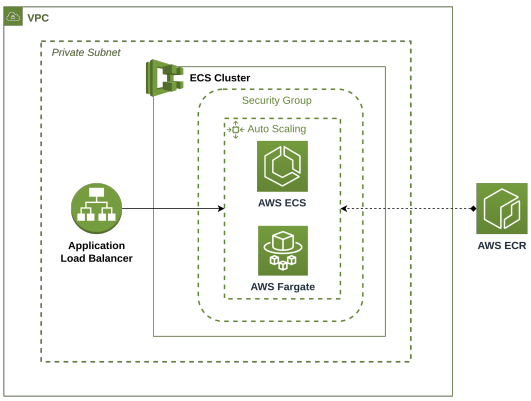
Para execução do mesmo, faz-se necessário a implantação de um *cluster* que consiste num agrupamento lógico de serviços. Também será utilizado uma Task Definition, recurso responsável pela definição das especificações referentes a execução da aplicação, como imagem Docker e dimensionamento de recursos como CPU e memória. Por se tratar de uma arquitetura construída dentro da AWS, optou-se pela utilização de um repositório privado dentro do ECR para armazenamento de imagens da aplicação.

A fim de garantir alta disponibilidade do sistema, definiram-se também duas políticas de escalabilidade. Existem três principais tipos de políticas de escalabilidade: sob demanda, agendada e preditiva. A política de escalabilidade sob demanda visa assegurar que novas instâncias sejam provisionadas uma vez que o valor limite referente à uma métrica de monitoramento especificada tenha sido atingido. A escalabilidade agendada consiste no provisionamento de novas instâncias em horários especificados e pré-configurados pelo usuário, baseado em conhecimentos prévios de um possível acréscimo na demanda no período. Já a política de escalabilidade preditiva executa algoritmos de inteligência artificial que prevem o tráfego futuro baseado em tendências diárias ou semanais, provisionando de maneira antecipada novas instâncias. Devido à compatibilidade da política sob demanda, a mesma foi utilizada na arquitetura proposta.

Como componente responsável pelo recebimento do fluxo do tráfego de entrada, utilizou-se um Application Load Balancer, que será responsável pela distribuição do tráfego entre as instâncias em execução e disponíveis. Para as políticas de escalabilidade sob demandas, definiu-se a utilização de métricas referentes à utilização de recursos de CPU e memória. As políticas tem como intuito assegurar que, caso o sistema apresente uma carga elevada de consumo de recursos, automaticamente novas instâncias sejam provisionadas. O cenário descrito representa o caso de escalabilidade horizontal, onde novas e idênticas instâncias são provisionadas momentaneamente a fim de suportar a alta demanda gerada, distribuída pelo balanceador de carga.

A Figura 8 apresenta uma visão geral da arquitetura proposta para execução de sistemas e os componentes necessários para sua construção são descritos na Tabela 3.

**Figura 8:** *Visão geral da arquitetura proposta para execução de sistemas utilizando ECS.*




Fonte: Elaborada pelo autor.

**Tabela 3:** *Recursos utilizados na arquitetura para implantação do sistema.*

| Recurso | Quantidade | Descrição |
|---|---|---|
| ECS Cluster | 1 | Agrupamento lógico de serviços dentro do ECS |
| ECS Service | 1 | Serviço responsável pelo gerenciamento e execução do container |
| ECR Registry | 1 | Repositório privado responsável pelo armazenamento de imagens da aplicação |
| Task Definition | 1 | Definição de tarefa que contém especificações referentes a execução da aplicação |
| Application Load Balancer | 1 | Balanceador de carga responsável por distribuir o tráfego entre as instâncias em execução |
| Security Group | 1 | Grupo de segurança que atua como um *firewall* de entrada na aplicação |
| Auto Scaling Policies | 2 | Políticas de escalabilidade utilizadas pelo serviço de execução de containers |
| IAM Role | 1 | Função de execução do serviço dentro do ECS |
| IAM Policy | 1 | Política que garante a integração com os demais componentes |

Fonte: Elaborada pelo autor.

A utilização de bancos de dados como componente responsável pelo armazenamento de informações e demais dados de interesse, muitas vezes provenientes de dispositivos IoT, é tipicamente empregada na arquitetura de sistemas. Devido a isso, faz-se necessário a disponibilidade de uma infraestrutura responsável pela configuração e execução de bancos de dados. Desta forma, optou-se pela utilização do serviço AWS RDS, que consiste em um serviço de banco de dados relacional gerenciado pela própria AWS. O serviço oferece suporte aos principais bancos relacionais utilizados no mercado, como PostgreSQL, SQL Server, MySQL, dentre outros.

A estrutura de execução do serviço RDS conta com três principais recursos, apresentados na Tabela 4. A instância RDS consiste em um servidor virtual onde será executado o banco de dados e apresenta diferenciação baseada em classes, responsáveis por determinar a capacidade computacional e de memória alocadas ao mesmo.

Os recursos *Subnet Group* e *Security Group* são utilizados como forma de assegurar a segurança e proteção dos dados. O grupo de sub-redes faz referência e define em quais sub-redes as instâncias serão executadas. Dessa forma, definiram-se as sub-redes privadas como parte do mesmo, assegurando que o acesso à instância seja restrito à rede privada executada dentro da VPC. Ainda, como uma camada extra de segurança e agindo de maneira semelhante à um *firewall*, isto é, restringindo o acesso ao servidor em determinadas portas e protocolos, definiu-se a utilização de um *security group*.

Visto que o balanceador de carga, responsável pelo acesso à aplicação, será executado em uma sub-rede privada, faz-se necessário a utilização de meios alternativos a fim de possibilitar que requisições sejam enviadas ao sistema. Deste modo, o serviço Amazon API Gateway apresenta-se como componente capaz de realizar não somente a integração mas também o controle de autorização e acesso aos recursos, assegurando uma camada

**Tabela 4:** *Recursos utilizados na arquitetura de implantação de banco de dados.*

| Recurso | Quantidade | Descrição |
|---|---|---|
| RDS *Instance* | 1 | Instância RDS responsável pela execução do banco de dados |
| *Subnet Group* | 1 | Grupo de sub-redes onde a instância RDS é executada |
| *Security Group* | 1 | Grupo de segurança que controla as permissões do tráfego de entrada e saída do banco de dados |

Fonte: Elaborada pelo autor.

extra de segurança. Possui suporte a criação de APIs RESTful e também APIs *WebSockets*.

No presente trabalho, definiu-se a utilização de uma API HTTP, agindo como uma interface de comunicação. Para correto funcionamento da mesma, faz-se necessário a criação de um VPC Link, recurso responsável pelo estabelecimento e liberação do acesso da API à rede privada, onde localiza-se a aplicação. Uma vez estabelecida a comunicação, configura-se um recurso de integração responsável pela integração entre a API HTTP e o *Load Balancer* da infraestrutura de execução do sistema e também um servidor de roteamento que atua como um *proxy* para a API. A Tabela 5 descreve recursos utilizados para construção de um API Gateway.

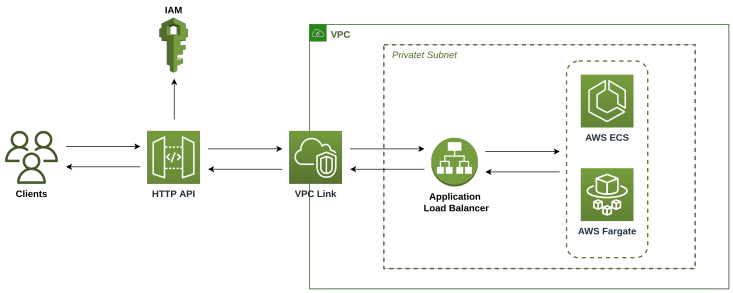
**Tabela 5:** *Recursos utilizados na construção do API Gateway.*

| Recurso | Quantidade | Descrição |
|---|---|---|
| VPC *Link* | 1 | Conexão do API Gateway com a rede privada |
| HTTP API | 1 | API responsável pela execução dos métodos utilizados |
| Route | 1 | Servidor *proxy* configurado com regras de roteamento |
| Integration | 1 | Integração entre a API e o *Load Balancer* privado |

Fonte: Elaborada pelo autor.

Conforme mencionado anteriormente, o serviço apresenta a possibilidade de configuração de autorização e controle de acesso aos recursos. Uma possível configuração pode ser realizada através da utilização do serviço de identidade Amazon IAM, fazendo com que seja necessário a autenticação através da utilização de uma chave de acesso e uma senha geradas pelo mesmo. Através da sua utilização, uma nova camada de segurança é adicionada, restringindo o acesso aos recursos de maneira a assegurar ainda mais segurança e controle, onde apenas pessoas ou dispositivos autenticados podem realizar o acesso. A integração proposta entre o API Gateway e os demais recursos é apresentada na Figura 9.

**Figura 9:** *Integração do API Gateway com o serviço do AWS ECS em uma rede privada.*




Fonte: Elaborada pelo autor.

## 3.3 Automação do Provisionamento da Infraestrutura

Para implementação da automação do provisionamento de infraestrutura, optou-se pela separação da mesma em módulos Terraform. Os módulos, que representam um agrupamento de recursos, foram desenvolvidos com o intuito de proporcionar sua reutilização de acordo com a necessidade e particularidade de cada sistema. Isto é, os módulos podem ser alocados de maneira que supra a necessidade da infraestrutura proposta. Desta forma, permite que, através de sua utilização, diferentes arquiteturas possam ser construídas de maneira automatizada. No caso específico do sistema a ser implantado neste trabalho, todos os módulos foram utilizados. Os módulos desenvolvidos são apresentados na Tabela 6.

**Tabela 6:** *Módulos Terraform desenvolvidos e utilizados para automação do provisionamento da infraestrutura.*

| Módulo | Descrição |
|---|---|
| vpc-module | Responsável pelo provisionamento da infraestrutura de redes do sistema |
| ecs-module | Responsável pelo provisionamento da infraestrutura de execução da aplicação |
| ecr-module | Provisiona o repositório utilizado para armazenamento de imagens da aplicação |
| rds-module | Provisionamento da infraestrutura do serviço de banco de dados |
| api-gateway-module | Responsável pela criação de um *gateway* utilizado para acesso à API privada |

Fonte: Elaborada pelo autor.

Cada módulo apresenta valores de entrada (variáveis de entrada), responsáveis pela configuração dos recursos implementados, e valores de saída (variáveis de saída), que consistem em atributos dos recursos provisionados. Todos os módulos desenvolvidos e seus respectivos códigos podem ser encontrados no Apêndice A.1 deste trabalho. Nesta seção será apresentado, de maneira breve, as definições de construção à cerca dos *scripts* desenvolvidos e suas integrações. Posteriormente, na Seção 4.3 será apresentado sua implementação de maneira prática.

### 3.3.1 Infraestrutura de redes

O módulo responsável pelo provisionamento da infraestrutura de redes é responsável pela construção dos recursos descritos na Seção 3.1. Para isto, desenvolveram-se *scripts*

Terraform associados a cada recurso de maneira independente. O módulo utiliza como parâmetros de projeto as seguintes variáveis:

- `vpc_name`: nome atribuído à rede privada isolada que será provisionada;
- `vpc_cidr`: intervalo de endereços de IPs contemplados pela rede;
- `private_subnets_cidrs`: lista de intervalos de IPs referentes às quatro sub-redes privadas;
- `public_subnets_cidrs`: lista de intervalos de IPs referentes às quatro sub-redes públicas;

Como será necessário a declaração da rede para provisionamento dos demais recursos, faz-se necessário a inserção de valores de saída para o módulo de redes, possibilitando que posteriores integrações possam ser realizadas. Desse modo, declararam-se três valores de saída como *outputs* do módulo:

- `vpc_id`: identificador da VPC criada dentro da AWS;
- `public_subnets_ids`: lista de identificadores referente às sub-redes públicas criadas;
- `private_subnets_ids`: lista de identificadores referente às sub-redes privadas criadas.

### 3.3.2 Infraestrutura de sistemas

A infraestrutura de execução de sistemas diz respeito à configuração principal da arquitetura. O módulo responsável pelo seu provisionamento contempla não somente a criação do serviço de execução de containers, mas também a criação dos demais recursos responsáveis por assegurar a resiliência e alta disponibilidade da mesma. Recursos como balanceador de carga e políticas de escalabilidade estão contidas no módulo. Além do serviço de execução de containers e dos recursos citados, o módulo é responsável pelo provisionamento de todos os componentes apresentados na Tabela 3.

Na construção do mesmo, propõe-se a utilização de variáveis que possibilitarão com que a infraestrutura seja alterada de acordo com a necessidade específica de cada projeto. Os parâmetros utilizados e necessários para configuração são:

- `container_port`: porta de execução da aplicação no container;
- `task_cpu`: valor total de recursos de CPU alocados para execução da aplicação;
- `task_memory`: valor total de recursos de memória alocados para execução da aplicação;
- `private_access`: variável booleana de controle de acesso. Indica se os recursos serão executados em uma sub-rede pública ou privada;
- `allowed_cidrs`: bloco de intervalo de IPs que será utilizado como restrição no grupo de segurança a fim de permitir acesso aos recursos;
- `healthcheck_url`: caminho de validação do estado da saúde da aplicação utilizado pelo balanceador de carga para verificações periódicas.

As demais configurações foram realizadas diretamente na implementação do módulo a partir da definição de valores padrões utilizados. Como principais configurações pré-definidas, destacam-se as políticas de escalabilidade. As mesmas foram implementadas com os valores de 60% de utilização de CPU e 80% de utilização de memória, isto é, uma vez atingido algum desses valores, novas instâncias serão provisionadas para atender a alta demanda de requisições ao sistema.

A fim de possibilitar integrações futuras com o API Gateway, fez-se necessário a declaração de dois valores de saída para o módulo. Os valores dizem respeito ao ARN (*Amazon Resource Name*) do *listener* criado para o balanceador de carga e ao identificador do grupo de segurança, também do balanceador de carga. Os valores são apresentados abaixo:

- `alb_listener_arn`: nome do recurso referente ao *listener* criado pelo balanceador de carga;
- `sg_alb_id`: identificador do grupo de segurança do balanceador de carga.

### 3.3.3 Serviço de armazenamento de imagens

Visto que a proposta faz uso do serviço de armazenamento de imagens da AWS, fez-se necessário o desenvolvimento de um módulo responsável pela criação de um repositório privado dentro do ECR. Este módulo é responsável unicamente pelo provisionamento do repositório e pela criação de uma política de tempo de vida das imagens armazenadas. A política implementada teve como intuito a aplicação de uma regra para armazenamento das 10 imagens mais recentes, evitando um acúmulo excessivo de versões que geraria possíveis custos de armazenamento. O parâmetro necessário de projeto é o nome do repositório a ser criado, definido pela variável *ecr_name*.

Ainda, apresenta um valor de saída, que consiste na variável *repository_url*, que informa a URL gerada para o repositório criado. Posteriormente, este valor será consumido pelo módulo responsável pela criação do ECS para mapeamento da imagem de execução da aplicação.

### 3.3.4 Serviço de banco de dados

Visto que o serviço de banco de dados diz respeito à uma particularidade de sistemas, isto é, não necessariamente faz-se seu uso, a melhor estratégia foi o desenvolvimento de um módulo independente responsável pelo seu provisionamento. Deste modo, a partir de sua utilização, pode-se incorporar ao projeto um banco de dados executado dentro do serviço gerenciado RDS, da AWS. Para sua criação, utilizaram-se os seguintes parâmetros:

- `database_name`: nome associado ao banco de dados criado;
- `allowed_cidrs`: lista dos intervalos de IPs cujo acesso é permitido ao banco;
- `engine`: sistema gerenciador de banco de dados que deseja ser utilizado;
- `engine_version`: versão do sistema gerenciador do banco de dados utilizado;
- `db_username`: nome do usuário de acesso ao banco de dados;
- `db_password`: senha de acesso ao banco de dados;
- `db_port`: porta de execução do banco de dados;

- `deletion_protection`: variável de controle de proteção para evitar remoções indesejadas;
- `storage_type`: tipo do disco de armazenamento utilizado no servidor;
- `allcoated_storage`: quantidade de memória de armazenamento alocada ao disco;
- `instance_class`: classe da instância de execução do servidor;
- `publicly_accessible`: variável booleana de controle de restrição de acesso público.

### 3.3.5 API Gateway

O módulo API Gateway foi desenvolvido a fim de proporcionar o acesso à aplicações que são executadas em redes privadas dentro da AWS. Para isto, implementa recursos responsáveis pela criação de métodos de integração que funcionam como um *proxy*, roteando o tráfego para o balanceador de cargas da aplicação. Os recursos são melhores descritos e apresentados na Tabela 5.

Como valores de entrada de projeto, declara parâmetros referente ao nome da aplicação e às configurações de seu balanceador de carga. São eles:

- `alb_listener_arn`: nome do recurso referente ao listener que deseja-se mapear;
- `aws_subnet_ids`: identificadores das sub-redes utilizadas para integração à rede privada;
- `sg_alb_id`: grupo de segurança do balanceador de cargas que será utilizado.

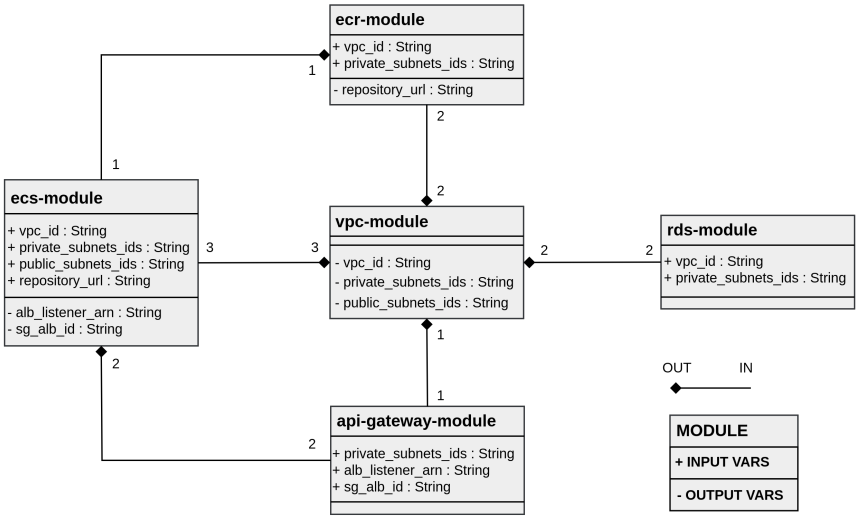
### 3.3.6 Integração e estrutura final dos módulos

Construídos os cinco módulos descritos anteriormente, faz-se possível a construção de uma arquitetura completa a partir de suas integrações. Os módulos podem ser integrados facilmente, visto que os parâmetros de entrada e saída que relacionam-os foram implementados de modo a proporcionar uma maior facilidade. Uma visão geral da integração entre os módulos e os respectivos parâmetros responsáveis pela mesma é apresentada na Figura 10.

A grande vantagem da utilização de infraestrutura subdividida em módulos é a possibilidade de construção simplificada. Cada componente pode ser incorporado ao projeto através da referenciação de seu respectivo módulo, fazendo com que diferentes arquiteturas possam ser aplicadas à diferentes projetos com a utilização da mesma automação implementada.

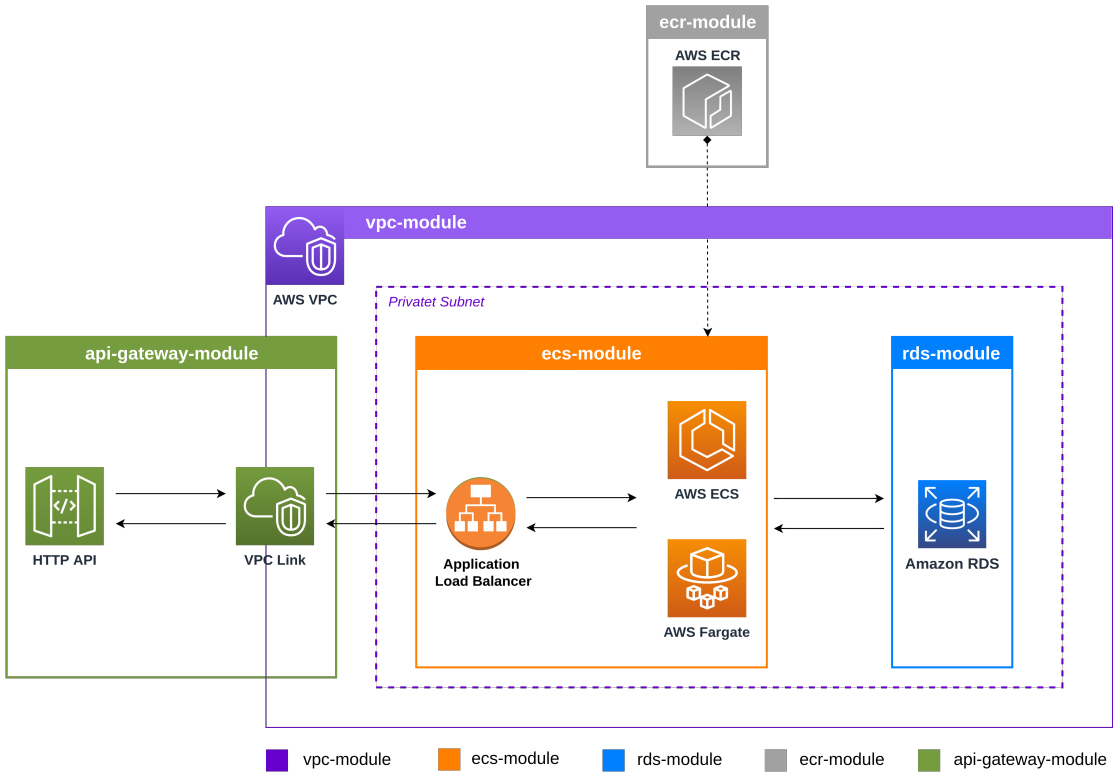
Para o estudo de caso que será proposto no presente trabalho, todos os módulos serão utilizados, acarretando na construção da arquitetura completa descrita anteriormente. Para melhor entendimento e visualização da implementação da automação utilizando módulos Terraform, é apresentada a Figura 11. Destaca-se que alguns recursos foram omitidos para melhor elucidação, porém, pode-se notar uma separação lógica entre cada componente da infraestrutura definida.

**Figura 10:** *Visão geral da integração entre os módulos implementados.*




Fonte: Elaborada pelo autor.

**Figura 11:** *Visão geral da arquitetura completa implementada utilizando os módulos Terraform desenvolvidos.*




Fonte: Elaborada pelo autor.

# 4 ESTUDO DE CASO

Como forma de validar a arquitetura desenvolvida e suas integrações, propõe-se a implantação de um sistema real à ser analisado em um estudo de caso. Deste modo, têm-se como objetivo validar a arquitetura desenvolvida e a automação implementada no processo de provisionamento da infraestrutura. Neste capítulo é apresentado o sistema a ser utilizado e o procedimento necessário para sua implantação.

## 4.1 Definição do Sistema

O sistema a ser implantado como forma de validar a automação e a arquitetura da infraestrutura proposta é um sistema de gerenciamento de ponto eletrônico. Dentre as integrações que o mesmo possui, faz uso de um banco de dados relacional, responsável pelo armazenamento de informações referentes a empregados e suas jornadas de trabalho.

A definição do sistema a ser utilizado neste estudo de caso se deve ao envolvimento prévio do autor com o mesmo. Ainda, visto que o sistema faz uso de dispositivos IoT, responsáveis pela aquisição de dados e posterior envio dos mesmos, apresenta a possibilidade da exploração de tópicos relativos ao curso de Engenharia de Controle e Automação, que serão abordados durante o desenvolvimento de um protótipo.

### 4.1.1 Componentes do sistema

O sistema definido para utilização neste estudo de caso faz uso de dois componentes principais, sendo estes uma API desenvolvida em ASP.NET Core e um banco de dados PostgreSQL. A API apresenta um conjunto de rotinas responsáveis pelo acesso às funcionalidades do sistema e comunicação com o banco de dados, responsável pelo armazenamento das informações referentes aos empregados e à gestão de suas jornadas de trabalho. A Tabela 7 apresenta os componentes do sistema, suas tecnologias e a respectivas descrições.

**Tabela 7:** *Componentes do sistema de gerenciamento de ponto eletrônico e suas tecnologias.*

| Componente | Tecnologia | Descrição |
|---|---|---|
| API | ASP.NET Core | Acesso às funcionalidades do sistema e suas integrações |
| Banco de Dados | PostgreSQL | Armazenamento de dados |

Fonte: Elaborada pelo autor.

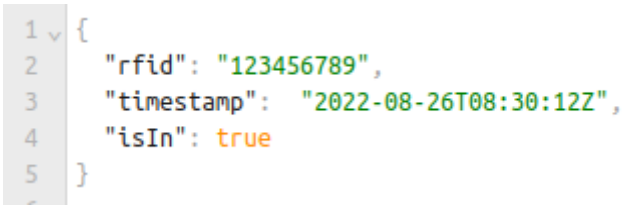
Um dos métodos implementados pela API e instrumento do escopo deste trabalho é o método do tipo POST denominado *WorkDay*. O método tem como objetivo a inserção

da informação referente à entrada ou saída do empregado na empresa no banco de dados. Utiliza em seu corpo uma mensagem no formato JSON, contendo os seguintes atributos:

- `rfid`: Atributo referente ao identificador RFID. Relaciona cada identificador ao respectivo empregado cadastrado previamente no sistema;
- `timestamp`: Instante de tempo em que ocorreu a aquisição dos dados para lançamento;
- `IsIn`: Atributo booleano responsável por informar se o lançamento diz respeito à entrada ou saída de um empregado, respectivamente.

Um exemplo de corpo de mensagem da requisição HTTP é apresentado na Figura 12.

**Figura 12:** *Exemplo de corpo de mensagem da requisição HTTP utilizada.*




Fonte: Elaborada pelo autor.

## 4.2 Desenvolvimento do Protótipo IoT

Visto que o sistema a ser utilizado faz uso de dispositivos responsáveis pela aquisição e inserção de dados em seu banco de dados, faz-se necessário o desenvolvimento de um protótipo que possibilite sua correta integração. Sendo assim, propõe-se a construção de um dispositivo IoT responsável pela aquisição de dados a partir de um identificador RFID e posterior inserção dos mesmos no sistema, capaz de assegurar que os dados referentes à entrada e saída sejam corretamente armazenados em local seguro e altamente disponível para acesso.

### 4.2.1 Definição do microcontrolador

Visto que o sistema necessita realizar o envio de dados através da Internet, a possibilidade de acesso à rede Wi-Fi integrado com o microcontrolador apresenta-se como parâmetro crucial a ser analisado. Ainda, devido a necessidade de conexão de sensores e outros componentes, a quantidade de portas I/O deve ser considerada, assegurando que a demanda do sistema possa ser suprida.

Conforme apresentado na Seção 2.5, a utilização da plataforma NodeMCU vêm ganhando destaque, sendo atualmente uma das plataformas IoT mais utilizadas pela comunidade. Tendo a vista a disponibilidade do módulo ESP8266 NodeMCU para execução do projeto e, uma vez o mesmo atendendo aos requisitos, definiu-se o mesmo como microcontrolador à ser utilizado.

### 4.2.2 Escolha dos componentes

Uma vez definido o microcontrolador a ser utilizado no projeto, o próximo passo consistiu na definição dos demais componentes empregados no protótipo.

Visto que o mesmo tem como principal funcionalidade a aquisição de dados através de um leitor, faz-se extremamente importante a escolha da tecnologia e do componente a ser utilizado. Para aplicações eletrônicas encontramos no mercado módulos NFC (*Near Field Communication*), desenvolvidos a fim de proporcionar uma maior praticidade em sua utilização. NFC é uma tenologia que apresenta-se como emergente no cenário atual e permite a troca de informações entre dispositivos que estejam próximos uns aos outros, sendo esta uma subdivisão do RFID, que visa melhorar a transmissão de dados ponto-a-ponto. Devido à isto, optou-se como componente responsável pela leitura dos dados de entrada a utilização de módulo NFC. Em análise realizada, observou-se a existência da biblioteca *Adafruit_PN532*, que possibilita integração do módulo NFC PN532 ao microcontrolador utilizado. Tendo em vista a possibilidade de integração apresentada e a disponibilidade do módulo para utilização, optou-se pelo NFC PN532 como leitor de dados de entrada.

De modo a proporcionar uma melhor experiência e interação com o usuário, utilizou-se um display LCD 16x2. O mesmo foi escolhido por sua ampla utilização em projetos eletrônicos e também pela possibilidade de utilização do módulo de expansão LMC1602 IIC, que facilita sua implementação.

Ainda, conforme descrito na Seção 4.1.1, no corpo da mensagem enviada na requisição ao método de entrada de dados deve ser informado se a mesma diz respeito à um lançamento de entrada ou saída, sendo este configurado de acordo com o parâmetro *IsIn*. Desta forma, faz-se necessário a inserção de um componente capaz de realizar a seleção do tipo de lançamento informado. Para isto, utilizou-se um *push button* 4 pinos como forma de seleção. A Tabela 8 apresenta a listagem final de componentes utilizados no projeto.

**Tabela 8:** *Listagem final de componentes utilizados no projeto.*

| Componente | Descrição |
|---|---|
| ESP8266 NodeMCU | Microcontrolador |
| NFC PN532 | Aquisição de dados de entrada |
| Display LCD 16x2 | Visor de exibição |
| LMC1602 IIC | Módulo I2C |
| *Push Button* | Seleção de tipo de dado |

Fonte: Elaborada pelo autor.

### 4.2.3 Elaboração do circuito

O módulo NFC PN532 possui suporte à comunicação I2C, SPI (*Serial Peripheral Interface*) e HSU (*High Speed UART*). Apresenta também suporte à leitura e escrita RFID e tecnologia do tipo *plug-and-play*, compatível com o microcontrolador utilizado. Quando operando no modo SPI, apresenta tensão de alimentação de 3,3 V com resistor de 100 Ω em série, podendo ser conectado diretamente à interface de tensão do microcontrolador. Tem como grande vantagem a possibilidade de utilização da biblioteca *Adafruit_PN532*, compatível com NodeMCU. Para implementação de comunicação SPI, faz-se uso de quatro conexões de barramento, sendo elas SCLK (*Serial Clock*), MOSI (*Master Out Slave In*), MISO (*Master In Slave Out*) e SS (*Slave Select*). Por padrão, a plataforma NodeMCU

implementa as conexões de barramento das interfaces em portas de I/O de acordo com o mapeamento apresentado na Tabela 9.

**Tabela 9:** *Mapeamento de conexões do módulo NFC PN532 para comunicação SPI.*

| NFC PN532 Pins | ESP8266 NodeMCU I/O |
|---|---|
| SCLK (*Serial Clock*) | D5 |
| MISO (*Master In Slave Out*) | D6 |
| MOSI (*Master Out Slave In*) | D7 |
| SS (*Slave Select*) | D4 |

Fonte: Elaborada pelo autor.

A conexão entre o NodeMCU e o display LCD é efetuada através da utilização do módulo de expansão LMC1602 IIC e a comunicação é realizada pelo protocolo I2C. O protocolo, desenvolvido para conectar periféricos de baixa velocidade a outros de maior velocidade, é nativamente suportado pela plataforma e tem sua implementação física facilitada, sendo necessário apenas um par de fios para conexão dos barramentos, denominados SDA (*Serial Data*) e SCL (*Serial Clock*). A plataforma NodeMCU implementa a biblioteca *LiquidCrystal_I2C*, que, por padrão, apresenta a conexão do par de fios dos barramentos de acordo com o mapeamento apresentado na Tabela 10.

**Tabela 10:** *Mapeamento de conexões do módulo LMC1602 para comunicação I2C.*

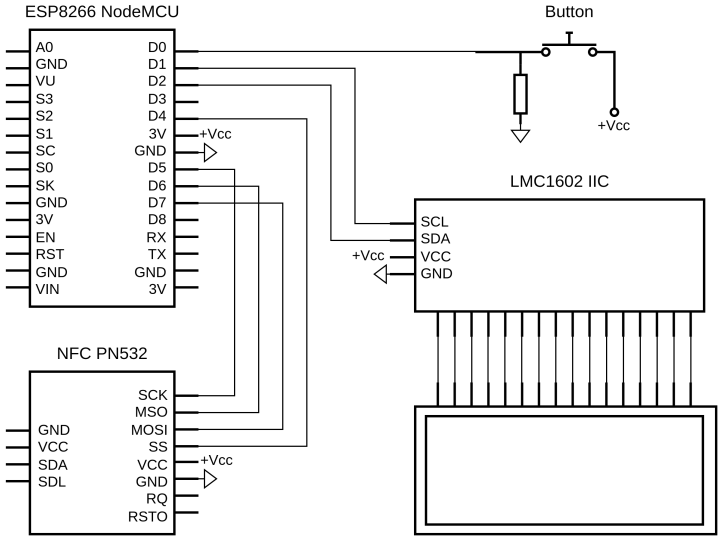
| LMC1602 IIC Pins | ESP8266 NodeMCU I/O |
|---|---|
| SCL (*Serial Clock*) | D1 |
| SDA (*Serial Data*) | D2 |

Fonte: Elaborada pelo autor.

O último componente a ser conectado junto ao microcontrolador é um *push button* 4 pinos. Para sua implementação, utilizou-se um resistor de *pull-down*, que faz com que o nível lógico alto seja gerado na entrada da porta de I/O no momento em que o botão encontra-se pressionado. Para o botão, a porta utilizada foi a D0.

O circuito final resultante implementado é apresentado na Figura 13. A montagem final do dispositivo pode ser visualizada no Anexo B.1.

**Figura 13:** *Esquemático de ligações entre o NodeMCU e os demais componentes.*




Fonte: Elaborada pelo autor.

### 4.2.4 Implementação do algoritmo

Após apresentada a construção física do protótipo desenvolvido, isto é, definido o microcontrolador, os componentes e a forma como os mesmos encontram-se fisicamente conectados, nesta seção será apresentado o algoritmo implementado no *firmware*. O objetivo do algoritmo implementado é realizar o comando e controle do módulo de aquisição, realizando o envio dos dados coletados diretamente para o sistema descrito na Seção 4.1.

A fim de garantir acesso à Internet ao dispositivo, o primeiro passo do algoritmo implementado consiste no estabelecimento de conexão WiFi a partir da utilização do módulo ESP8266. Para isso, fez-se o uso da biblioteca *ESP8266WiFi* que implementa uma rotina responsável pelo estabelecimento da conexão. Definiram-se também duas constantes denominadas *SSID* e *PASSWORD*, cujo valores dizem respeito ao respectivo identificador e senha da rede WiFi utilizada.

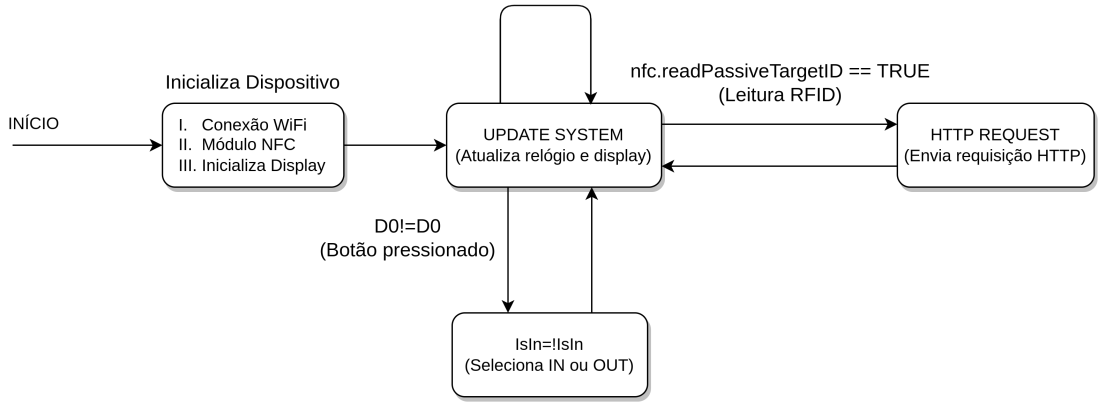
O componente principal é o módulo NFC, responsável pela aquisição de dados. Para sua implementação utilizou-se a biblioteca *Adafruit_PN532*, que possui compatibilidade com a plataforma utilizada. Uma vez inicializado o módulo, implementou-se uma rotina de leitura a partir da verificação cíclica da presença de algum dispositivo compatível com NFC dentro da faixa de proximidade do módulo de aquisição.

Realizada uma leitura pelo módulo NFC, o processo de envio de uma requisição HTTP é inicializado. Para isto, utilizou-se a biblioteca *ESP8266HTTPClient*, onde, inicialmente, é instanciado um novo *client*, responsável pelo estabelecimento de comunicação com a API. Para comunicação com a API, definiu-se uma constante denominada *BASE_URL*, que realiza o correto apontamento para URL gerada pelo API Gateway provisionado. O corpo da mensagem de requisição é construído de acordo com o formato apresentado anteriormente e o envio é realizado.

Ainda, fazem parte do ciclo de funcionamento do dispositivo os métodos responsáveis

pela inicialização do *display* LCD e de sua respectiva atualização, com base no horário obtido a partir de um relógio que implementa a biblioteca *NTPClient*. Para proporcionar um melhor entendimento do funcionamento do algoritmo implementado no microcontrolador, a máquina de estados simplificada do mesmo é apresentada na Figura 14.

**Figura 14:** *Máquina de estados simplificada do algoritmo implementado no dispositivo.*




Fonte: Elaborada pelo autor.

## 4.3 Provisionamento da Infraestrutura

Para provisionamento da infraestrutura do sistema serão utilizados os módulos Terraform desenvolvidos. Deste modo, faz-se necessário a utilização de três arquivos de configuração:

- `main.tf`: arquivo que contém a configuração dos módulos utilizados pelo projeto e demais requisitos necessários, sendo este o arquivo utilizado pelo projeto para execução do provisionamento da infraestrutura;
- `variables.tf`: arquivo que irá conter a declaração das variáveis utilizadas pelo projeto para configuração dos módulos;
- `terraform.tfvars`: declaração dos parâmetros de projeto que serão atribuídos posteriormente às respectivas variáveis utilizadas pelos módulos na automação do provisionamento da infraestrutura.

Para implantação do sistema proposto, definiu-se a alocação de 1024 MiB de memória e a utilização de 0,5 vCPU (*Virtual Central Processment Unit*), que consistem na quantidade de memória e de núcleos virtuais designados para a execução do container. Os valores foram escolhidos por serem suficientes para execução do mesmo. Ainda, para execução do banco de dados, o servidor escolhido foi o da classe *db.t3.micro*, que apresenta-se como um servidor de entrada com um nível básico de performance. Para configuração de rede, definiu-se a alocação de 8192 endereçamentos a partir da aplicação de uma máscara de sub-rede ao IP de rede informado, sendo este 10.0.0.0. Uma vez definido o endereçamento 10.0.0.0/19 para utilização na rede criada, subdividiu-se o mesmo de maneira igualitária

entre as quatro sub-redes públicas e as quatro sub-redes privadas. Ainda, definiu-se a restrição de permissão de acesso aos recursos somente para origens provenientes da rede criada, isto é, uma vez provisionado os recursos os mesmos encontram-se restritos apenas para acesso privado. A Figura 15 apresenta a configuração utilizada no projeto.

**Figura 15:** *Variáveis de projeto configuradas no arquivo* `terraform.tfvars` *para implantação do sistema.*

```
app_name = "atreviso-tcc-cca"

vpc_name             = "atreviso-tcc-cca"
vpc_cidr             = "10.0.0.0/19"
private_subnets_cidrs = ["10.0.0.0/22", "10.0.4.0/22", "10.0.8.0/22", "10.0.12.0/22"]
public_subnets_cidrs  = ["10.0.16.0/22", "10.0.20.0/22", "10.0.24.0/22", "10.0.28.0/22"]
region               = "us-east-1"
env                  = "dev"

database_name       = "atrevisotcc"
allowed_cidrs       = ["10.0.0.0/19"]
engine              = "postgres"
engine_version      = "13.4"
db_username         = "atrevisotcc"
db_password         = "admin1234"
db_port             = "5432"
deletion_protection = true
storage_type        = "gp2"
instance_class      = "db.t3.micro"
publicly_accessible = false
allocated_storage   = "30"

ecr_name = "atreviso-tcc-cca"

container_port  = 80
task_memory     = 1024
task_cpu        = 512
healthcheck_url = "/swagger/index.html"
private_access  = true
```

Fonte: Elaborada pelo autor.

Como pode ser observado, também foi declarado a utilização de um banco de dados do tipo PostgreSQL, conforme necessidade apresentada na Seção 4.1.1. No que diz respeito ao banco de dados, ainda foram informados parâmetros de segurança de acesso referentes à criação de usuário e senha utilizado.

Uma vez declarados os parâmetros de entrada necessários para execução dos módulos, realizou-se a construção do arquivo de configuração Terraform. Para sua utilização, declaram-se os módulos e os respectivos valores são atribuídos às variáveis de entrada dos mesmos. A Figura 16 apresenta a declaração do módulo ECS e a atribuição de variáveis junto ao mesmo, de forma a ilustrar o processo. Todos os módulos foram configurados e sua integração realizada com base no apresentado na Seção 3.3.6.

Uma vez devidamente configurado, a infraestrutura pôde ser provisionada através da utilização da plataforma Terraform Cloud.

**Figura 16:** *Declaração de módulo referente ao ECS no arquivo* `main.tf` *utilizado.*

```
65  module "ecs" {
66    source             = "app.terraform.io/atreviso/ecs/aws"
67    version            = "2.0.1"
68    app_name           = var.app_name
69    env                = var.env
70    region             = var.region
71    vpc_id             = module.vpc.vpc_id
72    public_subnets_ids  = module.vpc.public_subnets
73    private_subnets_ids = module.vpc.private_subnets
74    container_port     = var.container_port
75    task_memory        = var.task_memory
76    task_cpu           = var.task_cpu
77    allowed_cidrs      = var.allowed_cidrs
78    healthcheck_url    = var.healthcheck_url
79    private_access     = var.private_access
80    repository_url     = module.ecr.repository_url
81  }
```

Fonte: Elaborada pelo autor.

## 4.4 Análise de Custos

Após implantado o sistema e suas respectivas integrações, realizou-se uma análise de custos da infraestrutura envolvida. Para isso, utilizou-se a ferramenta de planejamento AWS Pricing Calculator, que possibilita o cálculo de estimativas de custos de soluções. A Tabela 11 apresenta a síntese dos custos mensais referentes à infraestrutura utilizada.

**Tabela 11:** *Estimativa de custos mensal obtida para implantação da arquitetura proposta.*

| **Serviço** | **Custo Mensal [USD]** | |
|---|---|---|
| | **Primeiros 12 meses** | **Após 12 meses** |
| AWS Fargate | 0,00 | 17,77 |
| RDS PostgreSQL | 0,00 | 15,44 |
| Load Balancer | 0,00 | 16,44 |
| Amazon ECR | 0,00 | 0,05 |
| API Gateway | 0,00 | 1,00 |
| **Total** | 0,00 | 50,70 |

Fonte: Elaborada pelo autor.

Ressalta-se que a plataforma AWS apresenta um plano gratuito de 12 meses para novos usuários, contemplado nos serviços e recursos utilizados. Após a expiração do período de teste gratuito, o custo resultante mensal estimado foi de USD 50,70 que, quando comparado à um investimento mínimo inicial para implantação de uma infraestrutura local, é expressivamente menor. Segundo levantamento realizado pelo autor, o custo atual estimado para implantação de servidores de entrada encontra-se na faixa de R$ 10.000,00. Ainda, a implantação de servidores locais geraria como consequência custos ligados à manutenções periódicas. Dessa forma, considera-se como satisfatório a estimativa de custo de implantação de infraestrutura em ambiente *cloud*.

# 5 RESULTADOS

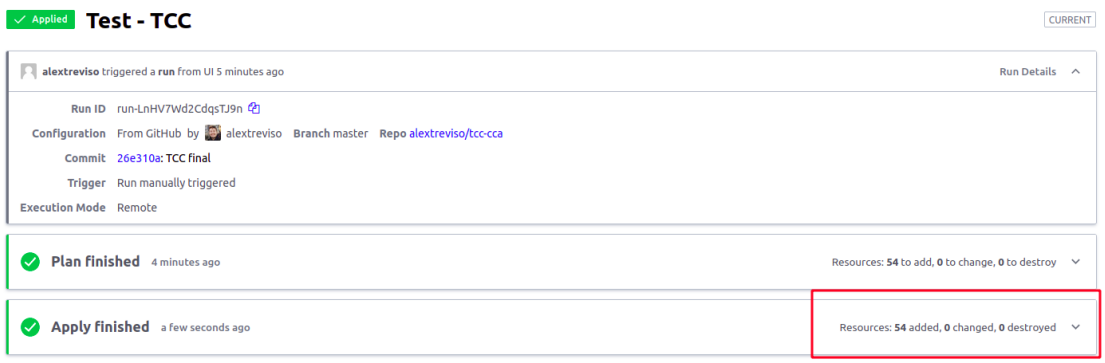
## 5.1 Automação de Infraestrutura

Através da utilização dos módulos Terraform desenvolvidos, deve-se prover a capacidade de realizar, de maneira automatizada, o provisionamento de infraestrutura de acordo com a arquitetura desejada. Para sua correta validação, propôs-se a implantação de um estudo de caso que fez uso dos cinco módulos desenvolvidos em sua totalidade.

O ambiente de execução da automação proposta foi estruturado na ferramenta Terraform Cloud, que consiste no serviço gerenciado de execução da HashiCorp. Através da sua utilização, elimina-se a necessidade de execução local do Terraform, apresentando também a facilidade de gestão de módulos, que podem ser armazenados e disponibilizados através do registro na plataforma. A utilização do registro permite ainda o versionamento dos módulos, que traz como benefício a possibilidade da utilização e desenvolvimento colaborativo dos mesmos. A ferramenta também se encarrega da gestão do estado da infraestrutura, componente fundamental para rastreabilidade e confiabilidade da mesma.

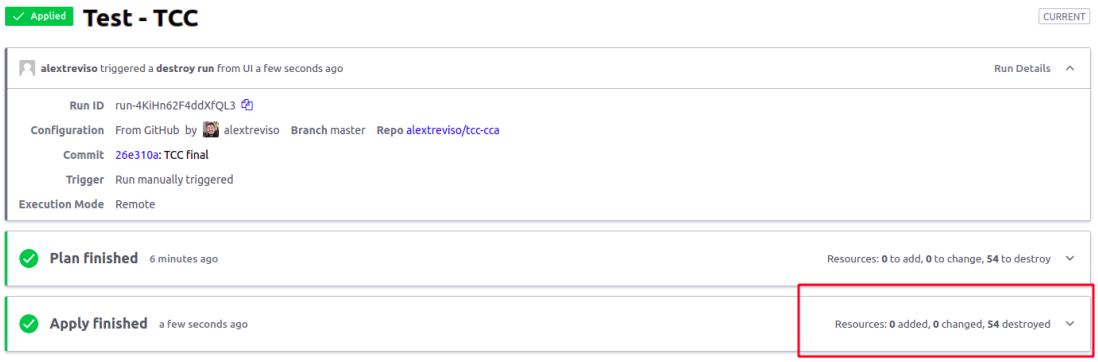
Um dos principais conceitos relacionados à Infraestrutura como Código diz respeito a capacidade da infraestrutura ser facilmente destruída e replicada em distintos ambientes. Deste modo, realizaram-se dois testes, um referente ao provisionamento dos recursos contemplados na arquitetura e um referente à destruição dos mesmos. As figuras 17 e 18 apresentam os resultados do provisionamento e destruição da infraestrutura utilizando Terraform.

**Figura 17:** *Execução do provisionamento da infraestrutura utilizando Terraform.*




Fonte: Elaborada pelo autor.

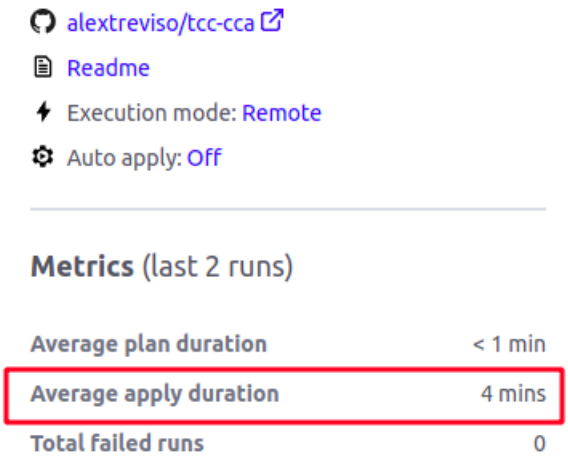
**Figura 18:** *Execução da destruição da infraestrutura provisionada utilizando Terraform.*




Fonte: Elaborada pelo autor.

É possível visualizar que, em ambos os casos, a execução foi concluída com sucesso. Através da utilização do Terraform e dos módulos desenvolvidos, foi possível realizar o provisionamento da arquitetura proposta. A quantidade total de recursos gerenciados via Terraform foi de 54. Ainda, o tempo médio de execução obtido para o provisionamento da infraestrutura proposta foi de quatro minutos, conforme apresentado na Figura 19.

**Figura 19:** *Tempo médio de execução obtido para o provisionamento da infraestrutura proposta.*




Fonte: Elaborada pelo autor.

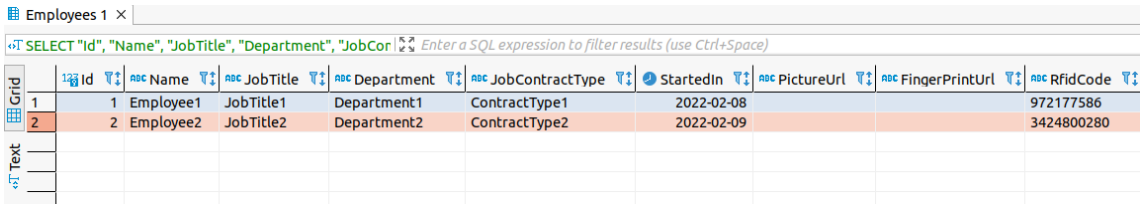
## 5.2 Análise do Sistema

Como forma de validar que o sistema encontrava-se acessível e disponível conforme o esperado, uma vez construído o protótipo do dispositivo IoT, realizaram-se testes a fim de validar a correta inserção dos dados no mesmo. Deste modo, configurou-se o dispositivo a fim de realizar o apontamento à URL gerada pelo API Gateway, responsável pelo acesso à

API e suas funcionalidades.

A fim de proporcionar a correta visualização dos dados, utilizou-se a ferramenta de gerenciamento de banco de dados DBeaver, onde, após devidamente conectado, pôde-se acessar e visualizar os dados presentes em suas tabelas. Para fins de teste, realizou-se a inserção de dois empregados teste junto à base de dados, denominados *Employee1* e *Employee2*, respectivamente. A Figura 20 apresenta uma captura de tela onde é possível visualizar os dois empregados inseridos no sistema para realização dos testes. Ressalta-se que para cada um dos empregados, atribuiu-se um identificador RFID referente à duas diferentes credenciais utilizadas.

**Figura 20:** *Captura de tela referente aos empregados inseridos na tabela Employees para realização dos testes.*

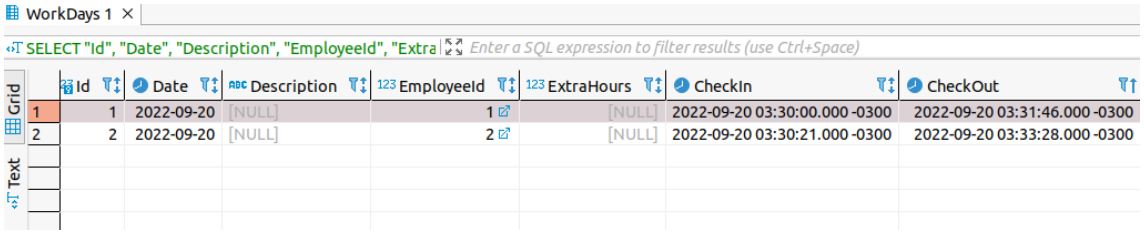



| | Id | Name | JobTitle | Department | JobContractType | StartedIn | PictureUrl | FingerPrintUrl | RfidCode |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Employee1 | JobTitle1 | Department1 | ContractType1 | 2022-02-08 | | | 972177586 |
| 2 | 2 | Employee2 | JobTitle2 | Department2 | ContractType2 | 2022-02-09 | | | 3424800280 |

Fonte: Elaborada pelo autor.

Devidamente cadastrados dois empregados fictícios junto à base de dados, realizou-se o teste do protótipo do dispositivo IoT desenvolvido. Para isto, a aquisição de dados referente à entrada (*CheckIn)* e saída (*CheckOut*) de ambos os empregados foi realizada a partir da utilização do mesmo. A Figura 21 apresenta uma captura de tela referente à tabela denominada *WorkDays* onde é possível visualizar a correta inserção dos dados gerados através do protótipo do dispositivo IoT.

**Figura 21:** *Captura de tela referente à inserção dos dados gerados através do dispositivo IoT na tabela WorkDays durante os testes.*




| | Id | Date | Description | EmployeeId | ExtraHours | CheckIn | CheckOut |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 2022-09-20 | [NULL] | 1 | [NULL] | 2022-09-20 03:30:00.000 -0300 | 2022-09-20 03:31:46.000 -0300 |
| 2 | 2 | 2022-09-20 | [NULL] | 2 | [NULL] | 2022-09-20 03:30:21.000 -0300 | 2022-09-20 03:33:28.000 -0300 |

Fonte: Elaborada pelo autor.

## 5.3 Análise de Segurança

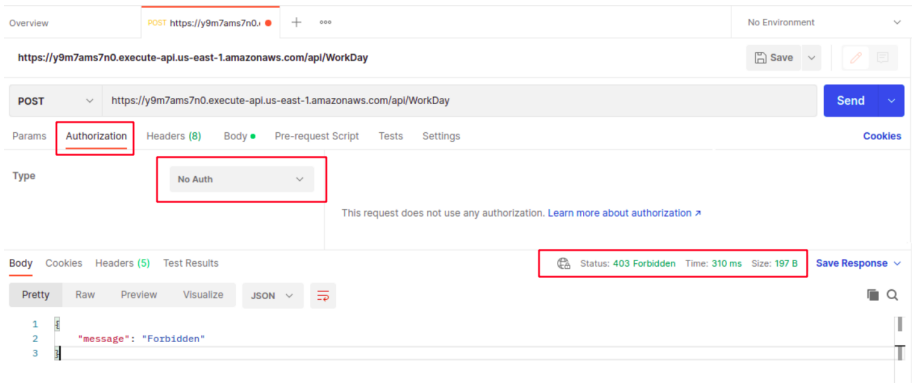
### 5.3.1 Controle de acesso à API

Conforme mencionado na Seção 4.1.1, é possível realizar a configuração de autorização e controle de acesso à API através da integração do serviço de identidade IAM junto ao *gateway*. Uma vez configurado, faz-se necessário realizar a autenticação através da utilização de uma chave de acesso e senha geradas e associadas à um usuário, que por sua vez deve conter as devidas permissões para execução de requisições à API. Sabendo disso,

realizou-se uma análise de segurança voltada à validação do controle de acesso à API a partir da utilização de um usuário teste.

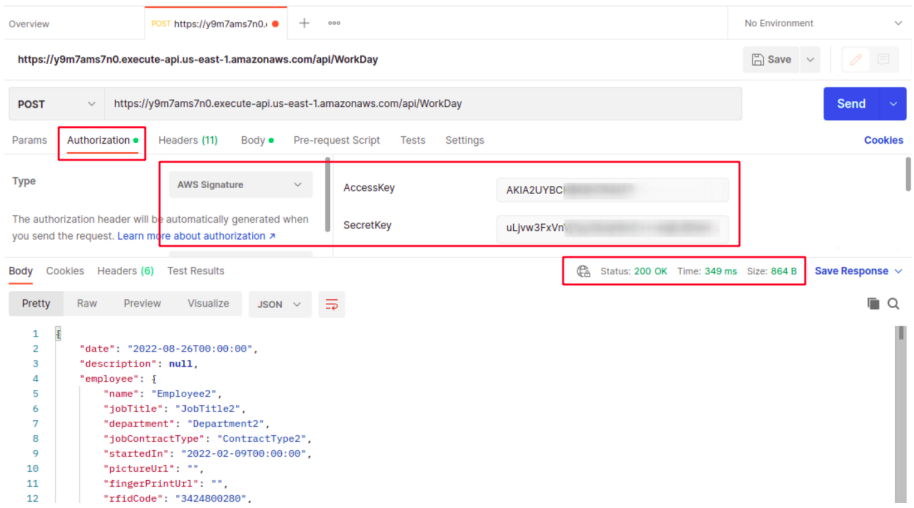
Para realização da validação utilizou-se o ambiente de teste de API da ferramenta Postman, onde estruturou-se uma requisição HTTP do tipo POST ao método utilizado e implementado pelo dispositvo IoT. A URL de destino utilizada foi a gerada pelo API Gateway. Dois casos foram analisados, sendo o primeiro realizando a requisição sem a utilização de nenhuma autenticação e o segundo utilizando autenticação do tipo AWS Signature, responsável pela inserção de informações de autenticação às solicitações da AWS enviadas por HTTP. A Figura 22 apresenta a resposta obtida para uma requisição sem a utilização de autenticação e a Figura 23 a resposta obtida para o caso em que a autenticação do tipo AWS Signature foi utilizada. Nas figuras estão salientados, em vermelho, os pontos de interesse a serem observados e mencionados anteriormente.

**Figura 22:** *Resposta obtida para uma requisição HTTP à API sem a utilização de autenticação.*




Fonte: Elaborada pelo autor.

**Figura 23:** *Resposta obtida para uma requisição HTTP à API com a utilização de autenticação do tipo AWS Signature.*




Fonte: Elaborada pelo autor.

É possível visualizar que no caso em que não foi utilizado autenticação na requisição,

obteve-se o código de resposta HTTP 403. O código de resposta obtido indica que o servidor recebeu a requisição porém não autorizou a emissão de uma resposta, de acordo com o esperado. Por outro lado, uma vez utilizada autenticação do tipo AWS Signature e fazendo uso dos dados do usuário de teste criado, pode-se validar seu correto funcionamento, visto que a requisição HTTP apresentou sucesso.

### 5.3.2 Segurança do banco de dados

Como parte fundamental da infraestrutura proposta encontra-se a segurança de dados. Desta forma, faz-se necessário validar as camadas de segurança implementadas que visam impedir possíveis acessos provenientes de outras redes ao servidor de execução do banco de dados. Como o servidor é executado em sub-redes privadas da arquitetura, o mesmo deve estar inacessível para origens que encontrarem-se fora da rede. Ainda, conta com um grupo de segurança que restringe o acesso em suas regras de ingresso apenas para a porta 5432, responsável pela execução do serviço PostgreSQL, e para a faixa de IPs dos recursos da rede privada.

Para realização do teste de segurança utilizou-se o protocolo de rede Telnet, empregado para acessar virtualmente um servidor. Analisaram-se os casos de tentativas de acesso provenientes de origem que encontram-se fora da rede e também de conexões proveniente da mesma rede. Neste caso, utilizou-se um *host* bastion, que consiste em uma instância que é provisionada com um endereço IP público e pode ser acessada por meio de SSH. A Figura 24 apresenta os resultados obtidos para as tentativas de acesso ao servidor realizadas.

**Figura 24:** *Tentativas de acesso ao servidor utilizando o protocolo Telnet para ambos os casos.*


Fonte: Elaborada pelo autor.

Através do teste, foi possível comprovar a possibilidade de acesso ao servidor apenas para o caso em que o *host* bastion foi utilizado, assegurando a restrição de acesso exclusivamente para origens que encontram-se dentro da mesma rede. Ainda, utilizando o *host* bastion validou-se o funcionamento do grupo de segurança implementado, visto que não foi possível realizar o acesso em uma porta distinta.

## 5.4 Análise de Escalabilidade

A fim de validar o correto funcionamento da infraestrutura proposta e a capacidade da mesma suportar variações expressivas nos *workloads*, mantendo-se disponível a partir do aumento de recursos devido à escalabilidade, propôs-se a implementação de um teste de estresse. O teste teve como objetivo a análise do comportamento do sistema quando submetido à elevadas cargas de trabalho.

Para desenvolvimento do teste, utilizou-se o Loadster, ferramenta voltada ao desenvolvimento de testes de carga e testes de estresse de alta performance. Possui a possibilidade de criação de diferentes cenários de teste voltados para APIs, fazendo uso de robôs que executam *scripts* pré-configurados, responsáveis pelo envio de milhares de requisições HTTP.

No caso do sistema implantado, os robôs representariam o comportamento dos dispositivos de aquisições de dados que interagem com a API a partir de requisições HTTP ao método de inserção de dados, da mesma forma como apresentado na Seção 4.2. Para isto, na ferramenta Loadster desenvolvou-se o script *"API Stress Test"*, que implementa uma requisição HTTP à URL do API Gateway. Desenvolveram-se cinco diferentes cenários de teste, apresentados na tabela abaixo.

**Tabela 12:** *Cenários desenvolvidos para o teste de estresse.*

| Cenário | Bots Simultâneos | Duração [min] |
|---|---|---|
| 1 | 25 | 5 |
| 2 | 50 | 5 |
| 3 | 75 | 5 |
| 4 | 100 | 5 |
| 5 | 100 | 15 |

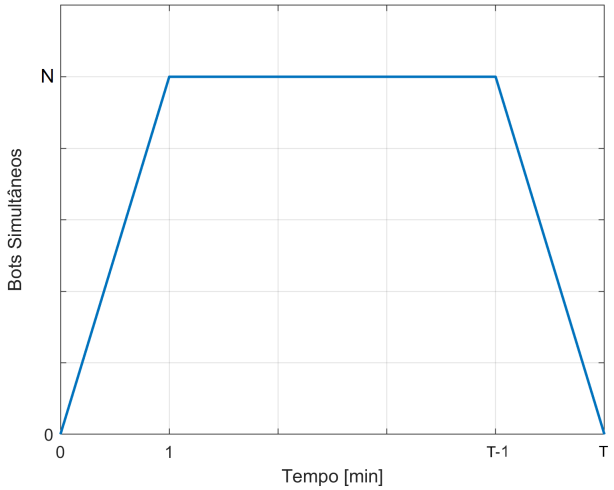
Fonte: Elaborada pelo autor.

Ambos os cenários desenvolvidos apresentam um acréscimo linear no número de robôs ao longo do primeiro minuto. Após o primeiro minuto, permanecem com a quantidade máxima de robôs enviando requisições até o instante T-1, onde T é definido pela duração, em minutos, do cenário de teste implementado. Por fim, no minuto final, ocorre o decréscimo no número de robôs. A Figura 25 apresenta o perfil implementado nos cenários, sendo N o número de robôs simultâneos definidos nos mesmos.

Ao final do teste, o relatório gerado apresenta as métricas obtidas ao longo da execução do mesmo. A análise obtida consiste em parâmetros como número total de iterações entre os robôs e a API e os devidos êxitos e erros obtidos. Ainda, o tempo de resposta obtido para as requisições HTTP foram analisados. A Tabela 13 apresenta os resultados obtidos durante a realização do teste.

Analisando os resultados, percebe-se que os testes com 25, 50 e 75 robôs simultâneos não apresentaram erros. Ainda, ambos os cenários apresentaram tempo médio de resposta de 0,02 segundos. No cenário com 100 robôs simultâneos e com tempo de duração de 5 minutos, foi possível visualizar a ocorrência de apenas 1 erro, porém observou-se um acréscimo no tempo médio de resposta, que foi de 0,04 segundos. Esse aumento do tempo médio atribui-se ao fato de algumas respostas terem resultado num tempo expressivamente superior, sendo o tempo máximo obtido de 5,29 segundos. Já para o cenário extremo, com 100 robôs simultâneos enviando requisições por 15 minutos, observou-se a ocorrência de 1785 erros nas 39952 requisições enviadas. Essa quantidade de erros corresponde à

**Figura 25:** *Perfil implementado nos cenários de testes planejados.*




Fonte: Elaborada pelo autor.

**Tabela 13:** *Resultados obtidos para cinco diferentes cenários analisados.*

| Bots Simultâneos | Duração [min] | Iterações | Êxitos | Erros | Tempo de Resposta [s] | | |
|---|---|---|---|---|---|---|---|
| | | | | | Máximo | Médio | Mínimo |
| 25 | 5 | 2978 | 2978 | 0 | 0,16 | 0,02 | 0,01 |
| 50 | 5 | 5945 | 5945 | 0 | 0,28 | 0,02 | 0,01 |
| 75 | 5 | 8911 | 8911 | 0 | 0,27 | 0,02 | 0,01 |
| 100 | 5 | 11787 | 11786 | 1 | 5,29 | 0,04 | 0,01 |
| 100 | 15 | 39952 | 38167 | 1785 | 25,72 | 0,09 | 0,01 |

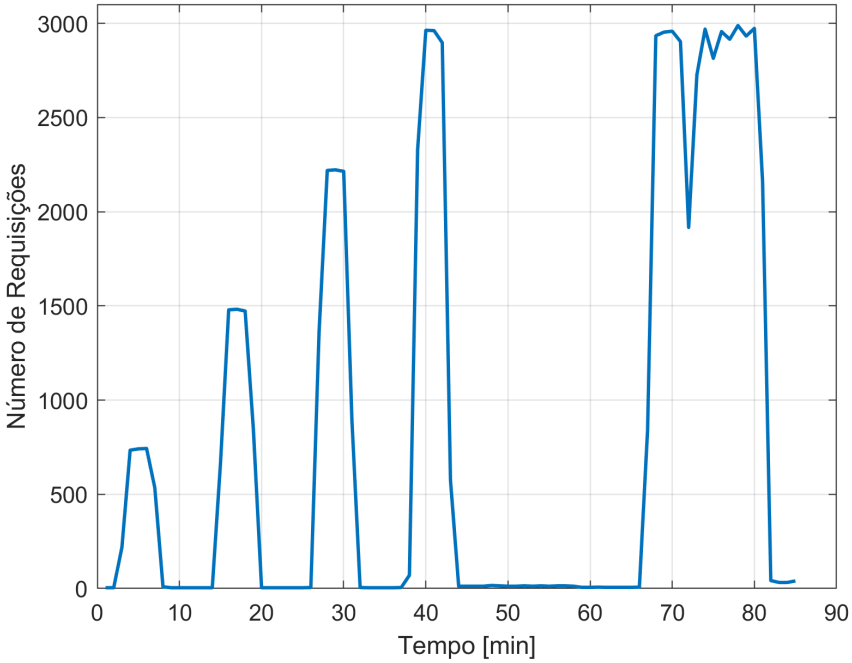
Fonte: Elaborada pelo autor.

um total de 4,468% das requisições enviadas. Ainda, visualizou-se um tempo de resposta médio de 0,09 segundos tendo o tempo de resposta máximo atingido 25,72 segundos.

A fim de proporcionar uma análise completa a respeito dos cenários propostos no que diz respeito ao comportamento da infraestrutura, analisaram-se também as métricas de utilização de CPU e memória da mesma. Ainda, analisaram-se métricas referentes ao número de requisições recebidas pela API durante o intervalo de tempo de realização do teste. Ambas as métricas foram retiradas do serviço de monitoramento de recursos AWS CloudWatch, com um período de amostragem de 1 minuto.

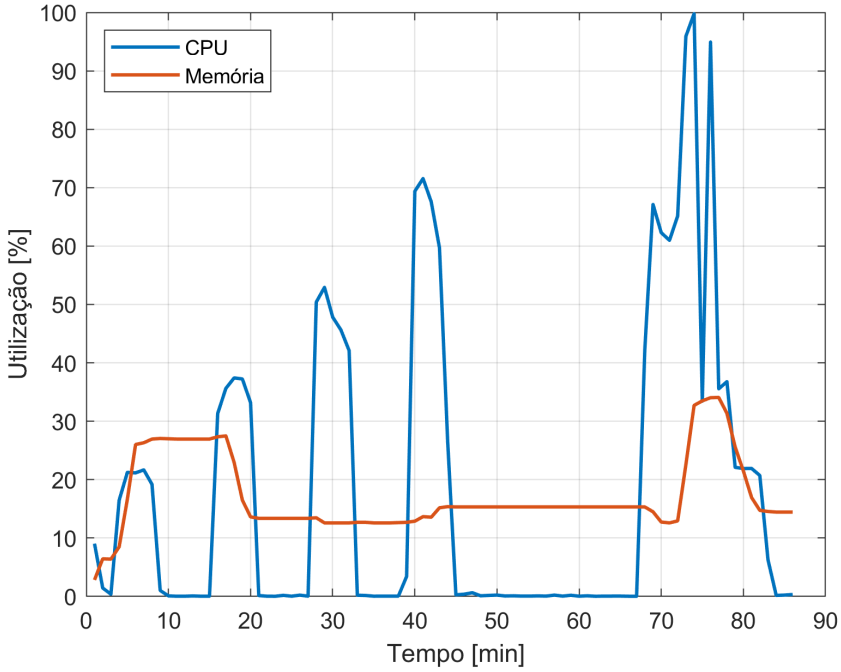
Primeiramente, analisaram-se as métricas referentes a contagem do número de requisições recebidas pela API durante o teste. É possível visualizar o acréscimo no número de contagens para os respectivos períodos, referentes aos cenários aplicados ao teste. Ainda, nota-se que durante o período em que tem-se a ocorrência do número máximo de robôs simultâneos enviando requisições para a API, o número de requisições permanece aproximadamente constante, comprovando a ocorrência de uma taxa de requisições constante ao longo do tempo de duração do teste. A Figura 26 apresenta as métricas do número de requisições recebidas pela API durante a realização do teste. Para correto entendimento e validação do comportamento da infraestrutura, analisou-se a utilização dos recursos da mesma. A Figura 27 apresenta as métricas de utilização de CPU e memória durante a realização do teste.

**Figura 26:** *Contagem de requisições recebidas pela API durante o teste de estresse.*




Fonte: Elaborada pelo autor.

**Figura 27:** *Análise da utilização de CPU e memória durante o teste de estresse.*




Fonte: Elaborada pelo autor.

É possível visualizar que o consumo de memória permaneceu inferior a 35% ao longo do teste. No caso da utilização de CPU, apenas para os cenários com 100 robôs simultâneos visualizou-se a ocorrência de um consumo superior a 60%, que foi o valor utilizado para disparo do alarme e consequentemente escalonamento da infraestrutura. Em ambos os casos pôde-se comprovar a ocorrência de escalonamento, conforme figuras disponíveis no Anexo B.2.

A partir da análise das métricas de utilização de CPU pode-se também visualizar o exato momento de ocorrência dos erros para o caso do cenário 5, que foi o instante de tempo por volta do minuto 74, em que o consumo de CPU atingiu sua capacidade máxima. Pôde-se observar também uma redução gradual em sua utilização após esse instante, acarretada pelo provisionamento de novas instâncias devido ao escalonamento horizontal da infraestrutura, comprovando o funcionamento implementado. Ressalta-se que, em um cenário ideal, possivelmente a ocorrência de erros não ocorreriam visto que o aumento da utilização dos recursos possivelmente não se daria de maneira abrupta. Porém, mesmo para o caso extremo analisado, observou-se que o sistema não apresentou a ocorrência de *downtimes* e rapidamente teve seu funcionamento estabilizado a partir da capacidade da escalabilidade horizontal da infraestrutura, comprovando sua eficiência para utilização em inúmeros sistemas que necessitam alta disponibilidade.

# 6 CONCLUSÕES

Considerando o objetivo principal deste trabalho, de realizar a automação do processo de provisionamento de infraestrutura em ambiente *cloud*, pôde-se concluir, através da implantação de um estudo de caso, que o projeto mostrou-se eficiente em sua proposta. Durante sua realização, implantou-se um sistema de gerenciamento de ponto eletrônico na arquitetura desenvolvida, que visou proporcionar um ambiente escalável, seguro e altamente disponível.

A automação do provisionamento da infraestrutura foi desenvolvida utilizando Terraform. A arquitetura proposta foi subdivida em cinco distintos módulos, fazendo com que diferentes arquiteturas possam ser aplicadas à diferentes projetos. No caso da infraestrutura proposta para o estudo de caso realizado, foi possível realizar o provisionamento da mesma com um tempo médio de 4 minutos, que apresenta-se satisfatório visto que a mesma diz respeito ao gerenciamento de uma totalidade de 54 recursos.

A correta validação do sistema e suas integrações foi realizada a partir do desenvolvimento de um protótipo de dispositivo IoT. O dispositivo, responsável pela aquisição de dados a partir de um sensor NFC e inserção dos mesmos em um banco de dados por meio da utilização de uma API HTTP, apresenta-se como solução para implantação do sistema em casos reais.

No que diz respeito à segurança, a infraestrutura proposta apresentou-se robusta à partir dos testes realizados. Validou-se a possibilidade de utilização de controle de acesso à aplicação através da integração do serviço de gerenciamento de identidade, onde através dos testes realizados foi constatado que somente requisições que utilizam autenticação em seu corpo seriam capazes de enviar informações ao sistema. Desta forma, a proposta apresenta-se como alternativa ao gerenciamento de dispositivos autorizados. Ainda, foi possível validar a segurança dos dados armazenados no banco de dados, visto que não foi possível realizar a conexão ao servidor a partir de uma origem externa sem a utilização de um *host* bastion hospedado na mesma rede.

Realizou-se também um teste de estresse a fim de analisar o correto funcionamento da escalabilidade da infraestrutura proposta. Durante o teste, analisaram-se cinco diferentes cenários que visavam verificar o correto funcionamento do sistema em casos de elevada utilização de recursos computacionais da infraestrutura. No caso extremo, pôde-se comprovar o correto funcionamento da escalabilidade sob demanda implementada a partir da visualização da ocorrência do provisionamento de novas instâncias no momento em que o limite de utilização foi atingido. Deste modo, assegura-se que, mesmo submetido à uma forte demanda de requisições e carga de trabalhos, o sistema possuirá a capacidade de escalar sua infraestrutura de maneira horizontal e distribuir o tráfego de entrada, evitando possíveis períodos de *downtimes* e assegurando alta disponibilidade ao mesmo.

A partir dos resultados obtidos no teste de estresse, concluiu-se também que a infra-

estrutura apresentou-se superdimensionada e com recursos subutilizados para o sistema utilizado em estudo de caso. Através do redimensionamento dos recursos computacionais e supressão do balanceador de carga, seria possível obter uma significativa redução no custo de infraestrutura envolvido no projeto e o funcionamento do mesmo não seria impactado.

Como trabalhos futuros, propõe-se a integração de autenticação junto ao dispositivo desenvolvido, uma vez que atualmente o *firmware* desenvolvido não implementa autenticação no envio da requisição HTTP. Ainda, no caso do sistema analisado, a utilização de escalabilidade agendada para os horários de maior utilização poderia ser explorada como alternativa à estratégia de escalabilidade sob demanda implementada.

# REFERÊNCIAS


AWS. AWS Global Infrastructure. In: Disponível em: https://aws.amazon.com/about-aws/global-infrastructure/. Acesso em: 06 ago. 2022.

BALA, R. et al. Magic Quadrant for Cloud Infrastructure and Platform Services. In: GARTNER, INC. Disponível em: https://www.gartner.com/technology/media-products/reprints/AWS/1-271W1OT3-PTB.html. Acesso em: 06 ago. 2022.

BRIKMAN, Y. *Terraform: up & running: writing infrastructure as code*. [S.l.]: O'Reilly Media, 2019.

BROWN, A. B.; HELLERSTEIN, J. L. Reducing the cost of it operations-is automation always the answer? In: HOTOS. [S.l.: s.n.], 2005.

CHELLAPPA, R. Intermediaries in cloud-computing: A new computing paradigm. In: INFORMS Annual Meeting, Dallas. [S.l.: s.n.], 1997. p. 26–29.

COLUMBUS, L. 32% Of IT Budgets Will Be Dedicated To The Cloud By 2021. In: Disponível em: https://www.forbes.com/sites/louiscolumbus/2020/08/02/32-of-it-budgets-will-be-dedicated-to-the-cloud-by-2021. Acesso em: 17 jul. 2022.

DE DONNO, M.; TANGE, K.; DRAGONI, N. Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. *Ieee Access*, Ieee, v. 7, p. 150936–150948, 2019.

FAROOQ, M. U. et al. A review on internet of things (IoT). *International journal of computer applications*, Foundation of Computer Science, v. 113, n. 1, p. 1–7, 2015.

GAJBHIYE, A.; SHRIVASTVA, K. M. P. Cloud computing: Need, enabling technology, architecture, advantages and challenges. In: 2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence). [S.l.: s.n.], 2014. p. 1–7.

GARRISON, G.; KIM, S.; WAKEFIELD, R. L. Success factors for deploying cloud computing. *Communications of the ACM*, Acm New York, NY, USA, v. 55, n. 9, p. 62–68, 2012.

HASHEMIPOUR, S.; ALI, M. Amazon web services (aws)–an overview of the on-demand cloud computing platform. In: SPRINGER. INTERNATIONAL Conference for Emerging Technologies in Computing. [S.l.: s.n.], 2020. p. 40–47.

IBM. What is Virtualization? In: Disponível em: https://www.ibm.com/br-pt/cloud/learn/virtualization-a-complete-guide. Acesso em: 31 jul. 2022.

JAIN, N.; CHOUDHARY, S. Overview of virtualization in cloud computing. In: IEEE. 2016 Symposium on Colossal Data Analysis and Networking (CDAN). [S.l.: s.n.], 2016. p. 1–4.

JIN, H. et al. Cloud types and services. In: HANDBOOK of cloud computing. [S.l.]: Springer, 2010. p. 335–355.

KUMAR, M.; DUBEY, K.; PANDEY, R. Evolution of emerging computing paradigm cloud to fog: applications, limitations and research challenges. In: IEEE. 2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence). [S.l.: s.n.], 2021. p. 257–261.

MAHMOUD, M. M. et al. Enabling technologies on cloud of things for smart healthcare. *IEEE Access*, IEEE, v. 6, p. 31950–31967, 2018.

MARSTON, S. et al. Cloud computing—The business perspective. *Decision support systems*, Elsevier, v. 51, n. 1, p. 176–189, 2011.

MORRIS, K. *Infrastructure as code: managing servers in the cloud*. [S.l.]: "O'Reilly Media, Inc.", 2016.

OPPENHEIMER, D.; GANAPATHI, A.; PATTERSON, D. A. Why do Internet services fail, and what can be done about it? In: 4TH Usenix Symposium on Internet Technologies and Systems (USITS 03). [S.l.: s.n.], 2003.

PRASSANNA, J.; PAWAR, A.; NEELANARAYANAN, V. A review of existing cloud automation tools. *Asian J Pharm Clin Res*, v. 10, p. 471–473, 2017.

SABHARWAL, N.; PANDEY, S.; PANDEY, P. Understanding Terraform Programming Constructs. In: INFRASTRUCTURE-AS-CODE Automation Using Terraform, Packer, Vault, Nomad and Consul. [S.l.]: Springer, 2021. p. 47–83.

SCHEEPERS, M. J. Virtualization and containerization of application infrastructure: A comparison. In: 21ST twente student conference on IT. [S.l.: s.n.], 2014. v. 21.

SIDDIQUI, T.; SIDDIQUI, S. A.; KHAN, N. A. Comprehensive analysis of container technology. In: IEEE. 2019 4th International Conference on Information Systems and Computer Networks (ISCON). [S.l.: s.n.], 2019. p. 218–223.

SKOULOUDI, C.; FERNÁNDEZ, G. Towards secure convergence of Cloud and IoT. In: EUROPEAN Union Agency for Network and Information Security. [S.l.: s.n.], 2018.

VARIA, J.; MATHEW, S. et al. Overview of amazon web services. *Amazon Web Services*, v. 105, 2014.

VILELA, P. H. et al. Performance evaluation of a Fog-assisted IoT solution for e-Health applications. *Future Generation Computer Systems*, Elsevier, v. 97, p. 379–386, 2019.

VOGEL, B. et al. What is an open IoT platform? Insights from a systematic mapping study. *Future Internet*, Multidisciplinary Digital Publishing Institute, v. 12, n. 4, p. 73, 2020.

# APÊNDICE A - SCRIPTS DESENVOLVIDOS

A fim de proporcionar uma melhor compreensão dos códigos desenvolvidos, sem que seja necessária sua completa adição ao documento, os mesmos encontram-se disponibilizados na plataforma GitHub. A listagem completa dos códigos desenvolvidos e as respectivas URLs de acesso, são apresentadas abaixo.

## A.1 Módulos Terraform

- Módulo ECS: https://github.com/alextreviso/terraform-aws-ecs
- Módulo ECR: https://github.com/alextreviso/terraform-aws-ecr
- Módulo VPC: https://github.com/alextreviso/terraform-aws-vpc
- Módulo RDS: https://github.com/alextreviso/terraform-aws-rds
- Módulo API Gateway: https://github.com/alextreviso/terraform-aws-api-gateway

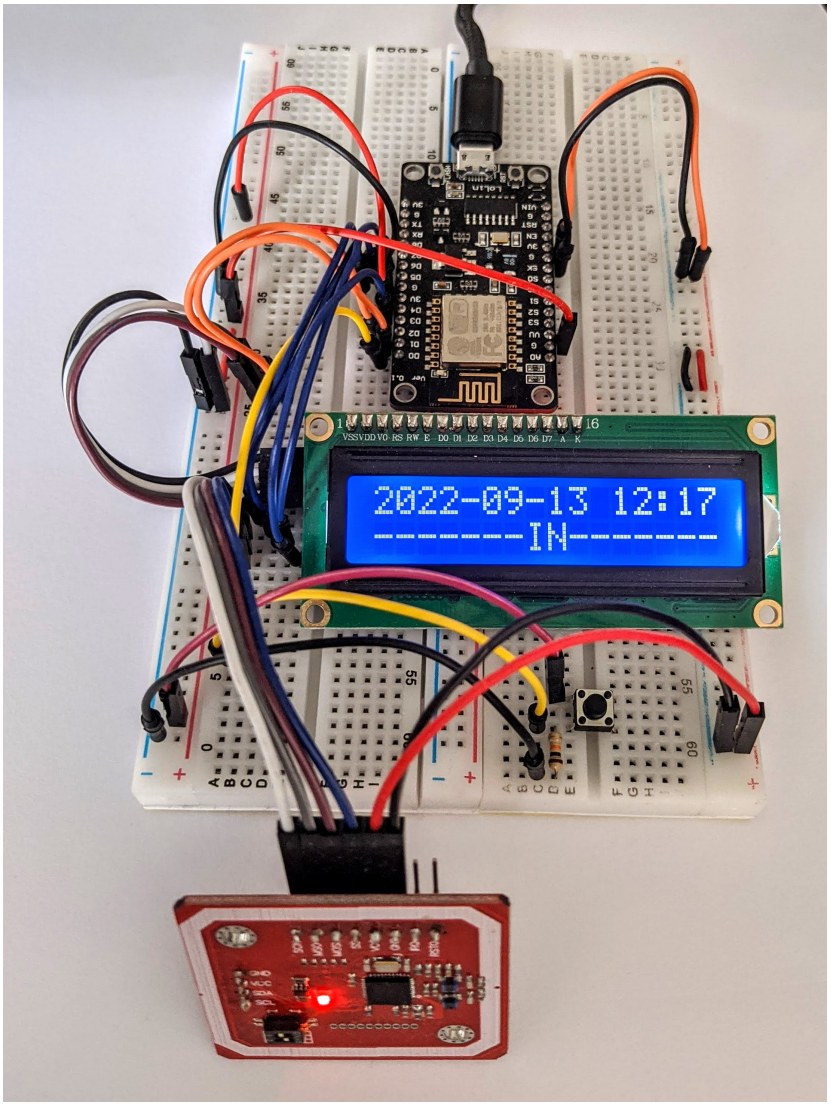
## A.2 Projeto final

Projeto TCC/CCA: https://github.com/alextreviso/tcc-cca

# APÊNDICE B - IMAGENS

## B.1 Protótipo desenvolvido

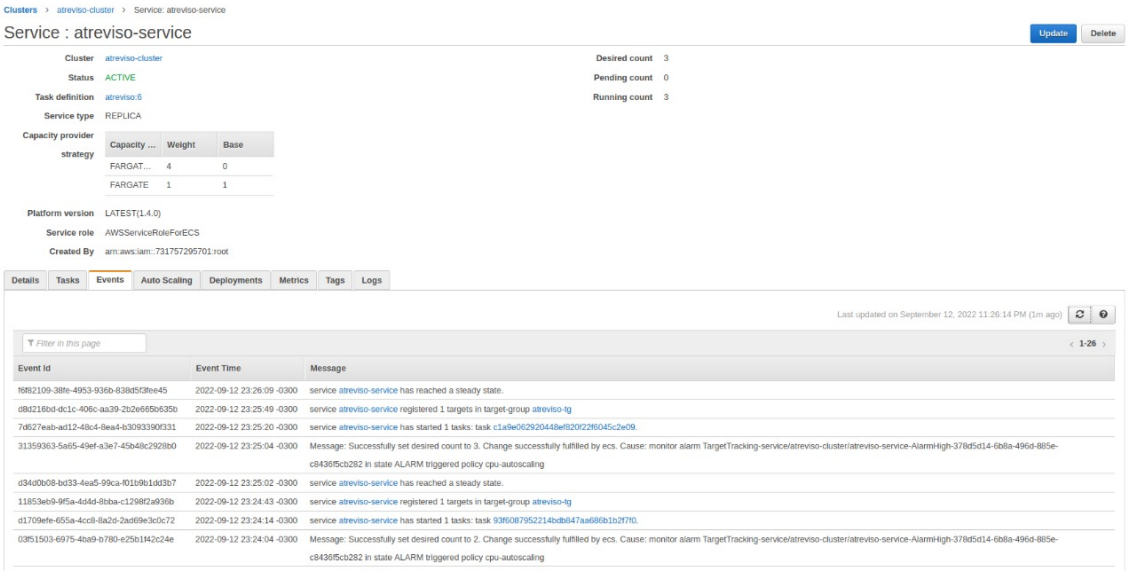
**Figura 28:** *Montagem final do protótipo desenvolvido.*




Fonte: Elaborada pelo autor.

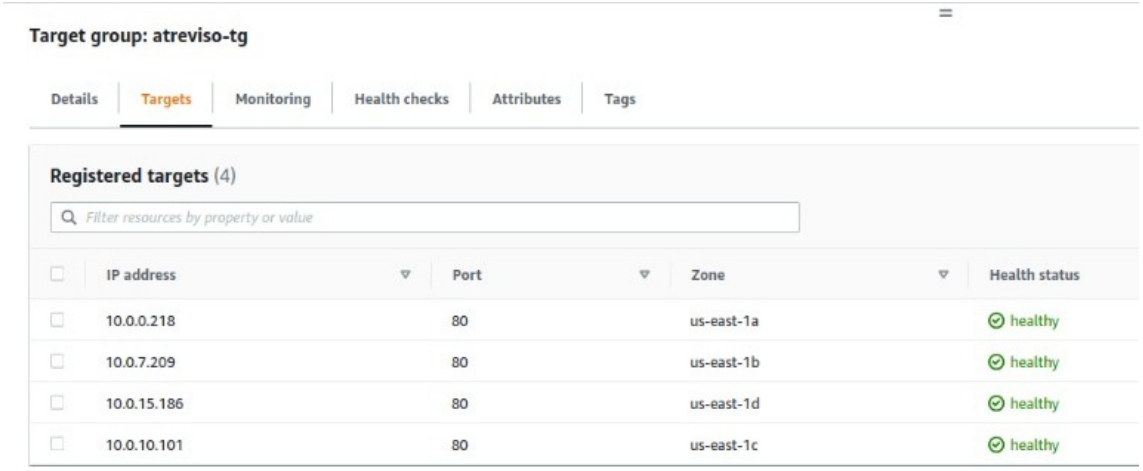
## B.2 Escalabilidade da infraestrutura

**Figura 29:** *Eventos gerados devido ao elevado consumo de CPU e ocorrência de escalabilidade da infraestrutura.*




Fonte: Elaborada pelo autor.

**Figura 30:** *Novos containers provisionados disponíveis para o balanceador de carga.*




Fonte: Elaborada pelo autor.