**Disponibilidade e resiliência de sistemas web com Chaos Engineering.**

Matheus de Oliveira Manhães – 5306315  
Felipe Barbosa de Oliveira – 5306

Projeto Final de Curso apresentado à Universidade do Grande Rio “*Prof. José de Souza Herdy*” (UNIGRANRIO) como parte dos requisitos para conclusão do curso de Bacharelado em Sistemas de Informação.

**BANCA EXAMINADORA**

Prof. Doutor Thiago Silva – Orientador – UNIGRANRIO

Dedico este trabalho...

Matheus de Oliveira Manhães

Dedico este trabalho...

Felipe Barbosa de Oliveira

**AGRADECIMENTOS**

Somos gratos...

*“Frase inspiradora... ”*

(Autor conhecido)

**RESUMO**

O que antes se considerava um tabu em questões de segurança e privacidade, hoje é possível observar que o uso de aplicativos e sistemas tem sido cada vez mais inerente à cultura e à rotina do ser humano. Vemos, por exemplo, o quão comum é uso do Netflix. A plataforma de vídeos sob demanda tem crescido muito e cada vez mais conquista clientes, ou usuários, ao redor do mundo. É possível perceber o quanto é investido para que os serviços da Netflix permaneçam disponíveis para seus usuários, em situações e dispositivos diversos. Estudar a estrutura de grandes sistemas na *World Wide Web* pode auxiliar na maneira como novos aplicativos e sistemas podem ser construídos garantindo qualidade e disponibilidade para os usuários. Uma das formas em que grandes empresas como Netflix, Amazon, Google, Microsoft, Uber e Dropbox conseguem manter seus serviços com qualidade é usando *Chaos Engineering.*

*Chaos Engineering* é o foco deste trabalho, que mostra como o processo de Engenharia do Caos funciona na execução e criação de ataques simulando rotinas de trabalho comuns, na tentativa de gerar turbulência a fim de preparar os engenheiros de software para as situações onde o *software* pode se tornar indisponível como interrupções de energia, congestionamento de requisições e demora em relação ao tempo de resposta afim de gerar resiliência e descobrir pontos fracos.

Este documento mostra como é feito o processo de Engenharia do Caos através de quatro pequenos sistemas que criamos: um buscador de repositórios de usuários do GitHub, uma API RESTful de cadastro de termos úteis relacionados a *Chaos Engineering,* uma calculadora comum e um gerenciador de tarefas. A ideia é usar essas aplicações como alvo de nossos experimentos de Caos.

**Palavras-chave:** Aplicativos, Sistemas, Web, Internet, Ataques, Experimento, Netflix, Engenharia do Caos.

**ABSTRACT**

What was once considered a taboo on security and privacy issues, today we can see that the use of applications and systems was increasingly inherent in the culture and routine of the human being. We see, for example, how common is the use of Netflix. The on-demand video platform has grown a lot and is increasingly catching customers, or users, around the world. You can see how much is invested for Netflix services that are permanently available to your users in situations and across devices. Studying a large system structure on the World Wide Web can help in how new applications and systems can be used with quality and availability for users. One of the ways in big companies like Netflix, Amazon, Google, Microsoft, Uber and Dropbox keeps their services quality using Chaos Engineering.

Chaos Engineering is the focus of this paper, which shows how the Chaos Engineering process works to create and execute attacks simutating common work routine in an attempt to generate turbulence to finalize software engineers' preparation for situations where software can become unavailable such as power outages, requirements congestion and response time delays in order to generate resilience and uncover weaknesses.

This document shows how the Chaos Engineering process is done through four small systems it creates: a GitHub users repositories search engine, a RESTful Chaos Engineering term record API, a common calculator, and a task manager. The idea is to use these applications as a target for our Chaos experiments.

**Keywords**: Apps, Systems, Web, Internet, Chaos Engineering, Attacks, Experiment, Netflix.

**SUMÁRIO**

[1 – INTRODUÇÃO 4](#_Toc24756114)

[1.1 – Justificativa da Pesquisa 4](#_Toc24756115)

[1.2 – Problema de Pesquisa 4](#_Toc24756116)

[1.3 – Hipótese 4](#_Toc24756117)

[1.4 – Objetivo 4](#_Toc24756118)

[1.5 – Objeto 4](#_Toc24756119)

[1.6 – Organização do Trabalho 4](#_Toc24756120)

[2 – REFERENCIAL TEÓRICO 4](#_Toc24756121)

[2.1 – Conceito de Sistemas Web 4](#_Toc24756122)

[2.2 – Netflix 4](#_Toc24756123)

[2.3 – Chaos Engineering 4](#_Toc24756124)

[3 – TECNOLOGIAS UTILIZADAS 4](#_Toc24756125)

[3.1 – Chaos Toolkit 4](#_Toc24756126)

[3.2 – Gremlin 4](#_Toc24756127)

[3.3 – HTML5 4](#_Toc24756128)

[3.4 – CSS3 4](#_Toc24756129)

[3.5 – JavaScript 4](#_Toc24756130)

[3.6 – Node.js 4](#_Toc24756131)

[3.7 – Angular 4](#_Toc24756132)

[3.8 – Docker 4](#_Toc24756133)

[3.9 – Kubernetes Minikube 4](#_Toc24756134)

[3.10 – AWS 4](#_Toc24756135)

[4 – APRESENTAÇÃO DA SOLUÇÃO 4](#_Toc24756136)

[4.1 – Descrição do Problema 4](#_Toc24756137)

[4.2 – Solução Proposta 4](#_Toc24756138)

[4.2.1 – Gremlin na Prática 4](#_Toc24756139)

[4.3 – Arquitetura do Projeto 4](#_Toc24756140)

[4.3.1 – Calculadora 4](#_Toc24756141)

[4.3.2 – Gerenciador de Tarefas 4](#_Toc24756142)

[4.3.3 – Buscador de Repositórios 4](#_Toc24756143)

[4.3.4 – API REST de Termos Úteis 4](#_Toc24756144)

[4.3.5 – Contêinerização das Aplicações 4](#_Toc24756145)

[4.3.6 – Osquestração dos Contêineres 4](#_Toc24756146)

[4.3.7 – Deploy dos Pods 4](#_Toc24756147)

[4.3.8 – Execução dos Testes 4](#_Toc24756148)

[4.3.9 – Criação dos Testes 4](#_Toc24756149)

[5 – CONCLUSÃO 4](#_Toc24756150)

[5.1 – Contribuições 4](#_Toc24756151)

[5.2 – Trabalhos Futuros 4](#_Toc24756152)

[REFERÊNCIAS BIBLIOGRÁFICAS 4](#_Toc24756153)

**LISTA DE FIGURAS**

[**Figura 1 - Arquitetura do Chaos Tookit** 4](#_Toc24744946)

[**Figura 2 - Arquitetura do Projeto** 4](#_Toc24744947)

**LISTA DE ABREVIATURAS E SIGLAS**

1. HTML5 –
2. CSS3 –
3. AWS –
4. API –
5. FaaS –

# 1 – INTRODUÇÃO

## 1.1 – Justificativa da Pesquisa

## 1.2 – Problema de Pesquisa

## 1.3 – Hipótese

## 1.4 – Objetivo

## 1.5 – Objeto

## 1.6 – Organização do Trabalho

# 

# 

# 2 – REFERENCIAL TEÓRICO

## 2.1 – Conceito de Sistemas Web

## 2.2 – Netflix

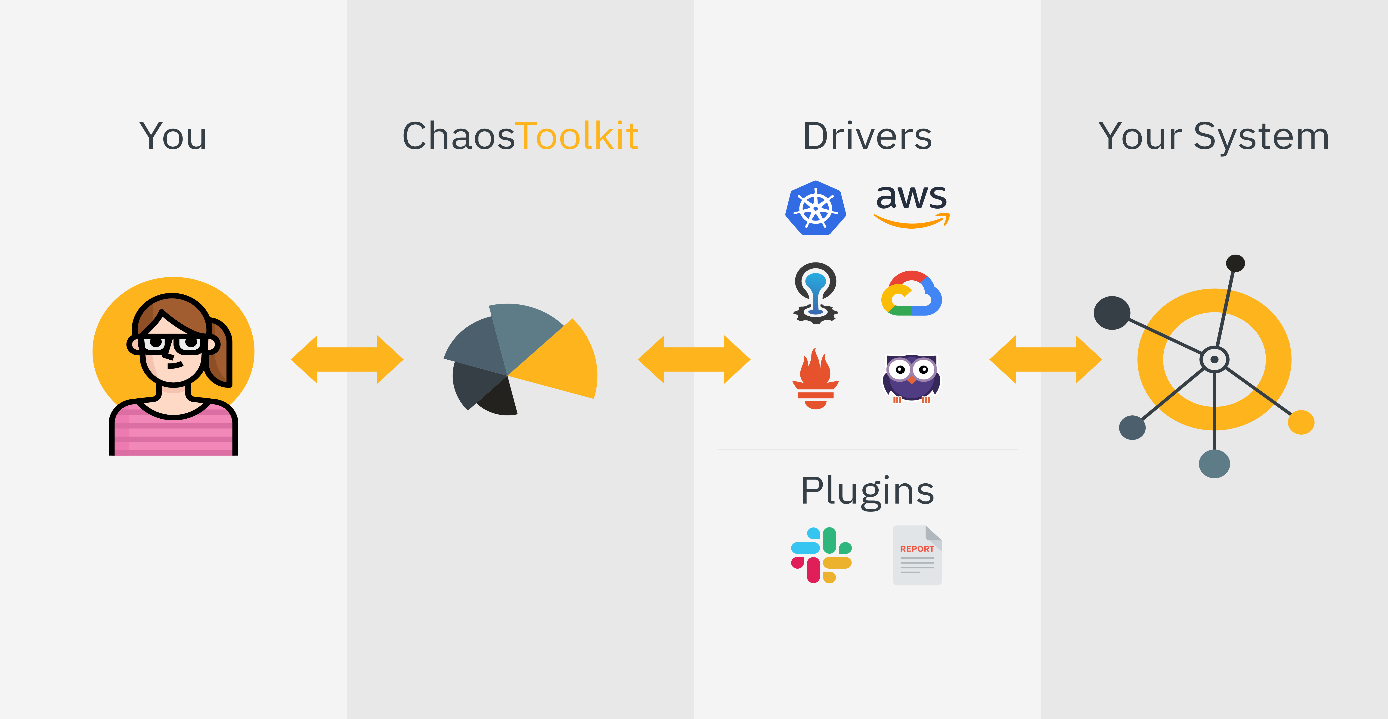
## 2.3 – Chaos Engineering

# 3 – TECNOLOGIAS UTILIZADAS

Nesta seção serão descritas as tecnologias Chaos Toolkit, Gremlin, HTML5, CSS3, JavaScript, NodeJS, Angular, Docker, Kubernetes Minikube e AWS.

## 3.1 – Chaos Toolkit

Chaos Toolkit é um framework de código aberto que disponibiliza um kit de ferramentas para execução e criação de experimentos de Chaos Engineering. O framework tem dois principais propósitos: prover uma ferramenta completa, porém simples a fim de fornecer um ponto de partida fácil para a aplicação da disciplina e disponibilizar uma API aberta para a comunidade para que qualquer experimento de Caos possa ser executado consistentemente usando integrações com outras ferramentas que estão surgindo.



**Figura 1 - Arquitetura do Chaos Tookit**

Fonte: Documentação do Chaos Tookit

O Chaos Toolkit visa tornar simples e direto, seguindo os Princípios da Engenharia do Caos, a execução de experimentos contra o seu sistema ativo, a fim de criar confiança em seu comportamento e aprender sobre possíveis pontos fracos.

Foi escolhida esta ferramenta para poder escrever casos de testes similares ao Gremlin. O uso do Chaos Toolkit permite a criação dos experimentos, execução dos mesmos e integração com os outros serviços como o Gremlin, Docker, Kubernetes e AWS.

## 3.2 – Gremlin

O Gremlin é uma ferramenta paga, mas oferece alguns recursos limitados grátis. Criada a fim de “Transformar o fracasso em resiliência”, como diz em sua documentação, o Gremlin oferece uma estrutura a fim simular interrupções reais com segurança e com uma crescente biblioteca de ataques. Gremlin se encaixa na categoria “Failure as a Service”, onde oferece uma plataforma para os usuários encontrarem pontos fracos em seus sistemas antes que eles possam causar problemas.

Escolhemos o Gremlin, pois é possível realizar alguns experimentos que estão divididos pelas categorias “Resource”, “State” e “Network”. Cada categoria possui uma lista de ataques de acordo com cada perfil. Na versão grátis estão disponíveis dois ataques: CPU na categoria “Resource” e “Shutdown” na categoria “State”. Escolhendo o Gremlin como ferramenta deste trabalho podemos executar experimentos de aumento do consumo de CPU do servidor assim como desligá-lo programadamente com alguns passos de configuração.

## 3.3 – HTML5

HTML é sigla para HyperText Markup Language, que significa Linguagem de Marcação de Hipertexto. HTML é a linguagem de marcação padrão para páginas Web. Com ela é possível criar páginas web estáticas e dinâmicas. HTML descreve a estrutura de uma página da Web que consiste em uma série de elementos que informam ao navegador como exibir o conteúdo a quem está acessando a página.

Os elementos HTML são representados por *tags* que rotulam partes do conteúdo, como "cabeçalho", "parágrafo", "tabela" e assim por diante. Os navegadores não exibem as tags HTML, mas as usam para exibir o conteúdo da página.

Para este projeto escolhemos utilizar o HTML5 que é a versão mais atual da linguagem com novos elementos, atributos e comportamentos. Com ela, segundo sua documentação na página do Mozilla, é possível descrever mais precisamente o nosso conteúdo por conta da inserção das *tags* semânticas, além de permitir que páginas web armazenem dados localmente em no lado do cliente (navegador)entre outras muitas funcionalidades que auxiliam e muito na rotina do programador.

## 3.4 – CSS3

CSS é uma sigla para Cascading Style Sheets, Folhas de Estilo em Cascata na tradução pra Língua Portuguesa, que é a linguagem que define o estilo (formatação, cores, espaçamento, posição de elementos etc.) de um documento HTML, ou seja, descreve como os elementos HTML devem ser exibidos.

O estilo da página com CSS pode ser feito de várias maneiras: usando uma *tag* de estilono documento inteiro, usando uma propriedade de estilo para cada *tag* ou criando um link para um arquivo *.css* externo ao documento de conteúdo (HTML).

Neste projeto usamos 3 aplicações que possuem páginas *web* com HTML, então decidimos utilizar CSS3 que é a versão mais atual da ferramenta junto com o framework do Twitter chamado Bootstrap, que nada mais é do que um conjunto de estilos previamente definidos e configurados onde podemos ter acesso usando sua documentação na internet. Optamos por utilizar CSS3 com Bootstrap pelo fato de ser uma biblioteca muito utilizada em muitos sites renomados e por possuir estilos criativos e funcionais.

## 3.5 – JavaScript

Uma linguagem de programação que surgiu em 1995 que tinha como objetivo ser linguagem de programação para o lado do cliente (navegador), algo que nos últimos anos vem sendo resignificado desde o surgimento do Node.js, um interpretador javascript para o lado do servidor.

JavaScript é uma linguagem multi-paradigma de tipagem dinâmica que não faz uso de compilador por ser uma linguagem interpretada, portanto a maioria de seus erros podem apenas ser descobertos em tempo de execução.

Foi o escolhido o JavaScript como linguagem de programação, pois faz o casamento perfeito com as tecnologias HTML5 e CSS3. JavaScript é a linguagem mais utilizada na web para fazer a interação dos elementos HTML e estilo CSS como botões, barras de navegações, fundos animados e muitas outras possibilidades.

## 3.6 – Node.js

É um interpretador de código que tem como propósito interpretar o JavaScript no lado do servidor, coisa que só era possível no lado do cliente (navegador). Isso é possível hoje, pois Ryan Dahl, engenheiro por trás do Node.js, utilizou o motor *open-source* do Google Chrome, chamado V8, no lado do servidor, assim sendo capaz de interpretar código JavaScript não mais somente no lado do cliente, como era até 2009.

Foi o escolhido o Node.js como tecnologia para o desenvolvimento da API de termos úteis de Chaos Engineering pela sua simplicidade para subir um servidor e, como a lógica desta API é algo simples, ou seja, somente um CRUD de termos úteis com ID, título e descrição, foi escolhido o Node.js, junto com a biblioteca Express e o banco de dados não relacional MongoDB para construção.

## 3.7 – Angular

Angular é um framework usado para construção da interface de aplicações Web usando HTML, CSS e, principalmente JavaScript. O framework é mantido pela Google e a sua primeira versão chamada AngularJS foi totalmente reescrita e passou, a partir da versão 2, a ser chamado somente de “Angular”. A partir da versão 2 fizeram um framework totalmente novo. Entre as muitas mudanças do AngularJS para as versões superiores, destaca-se a linguagem padrão de desenvolvimento do Angular que passou a ser o TypeScript, um superconjunto da Microsoft que insere tipagem no JavaScript.

Para este projeto foi escolhido o uso do Angular para o desenvolvimento da calculadora e do gerenciador de tarefas, pois é um framework mantido por uma empresa de alta credibilidade, além de ser um framework excelente para criação de SPAs, ou Single Page Applications.

## 3.8 – Docker

O Docker é uma plataforma aberta de desenvolvimento, envio e execução de aplicações. Fornece a capacidade de empacotar e executar um aplicativo em um ambiente pouco isolado chamado contêiner. O isolamento e a segurança permitem executar muitos contêineres simultaneamente em um determinado host.

Escolhemos o Docker para este projeto, pois com o Docker é possível criar imagens (containers prontos para *deploy*) a partir de arquivos de definição chamados Dockerfiles com facilidade, além do Docker utilizar como *backend* *default* o LXC, sendo possível definir limitações de recursos por container (memória, CPU e I/O, por exemplo). Sendo assim, o Docker “caiu como uma luva” para que pudéssemos subir as 4 aplicações em contêineres, visto que também seria fácil a integração com o Kubernetes e o AWS, que serão tratados nos próximos tópicos.

## 3.9 – Kubernetes Minikube

O Kubernetes, também chamado de k8s, é uma plataforma de código aberto que automatiza as operações dos containers Linux. Essa plataforma elimina grande parte dos processos manuais necessários para implantar e escalar as aplicações em containers. O Kubernetes oferece um serviço para que os pods (grupo de um ou mais contêineres) sejam facilmente gerenciados.

Já o Minikube é uma ferramenta que facilita a execução local do Kubernetes. O Minikube executa um cluster Kubernetes de nó único dentro de uma Máquina Virtual (VM).

Esse projeto faz uso do Kubernetes com Minikube, pois ele tem a responsabilidade de gerenciar as aplicações com facilidade e eficiência. O Kubernetes foi feito para gerenciar *clusters* que podem incluir hosts em *clouds* públicas, privadas ou híbridas. Por isso, o Kubernetes é a plataforma ideal para hospedar aplicações nativas em *cloud* que exigem escalabilidade rápida, algo que é essencial para aproveitarmos, visto que iremos realizar testes e experimentos de Chaos Engineering, onde precisaremos subir serviços que foram derrubados, analisar indisponibilidade dos serviços, configurar redirecionamento de fluxo entre outras coisas que o Kubernetes irá ajudar e muito.

## 3.10 – AWS

Amazon Web Services, também conhecido como AWS, é uma plataforma de serviços de computação em nuvem. É a maior empresa fornecedora de serviços em nuvem, oferecendo cerca de 165 serviços completos incluindo série de aplicativos, incluindo computação, armazenamento, bancos de dados, redes, análises, machine learning e inteligência artificial (IA), Internet das Coisas (IoT), segurança e desenvolvimento, implantação e gerenciamento de aplicativos.

Foi escolhido o uso do AWS para este projeto para não precisar fazer o *setup* e o *deploy* de toda a configuração e recursos do projeto em uma máquina física. Seria extremamente complicado ter que deixar um computador disponível e ligado para que pudéssemos realizar as atividades deste trabalho, além de termos a possibilidade de analisarmos os gráficos de monitoramento do serviço da Amazon, visto que o AWS oferece serviços *cloud* de maneira excepcional, onde grandes empresas utilizam de seus recursos, além de oferecer serviços grátis com fins experimentais.

# 4 – APRESENTAÇÃO DA SOLUÇÃO

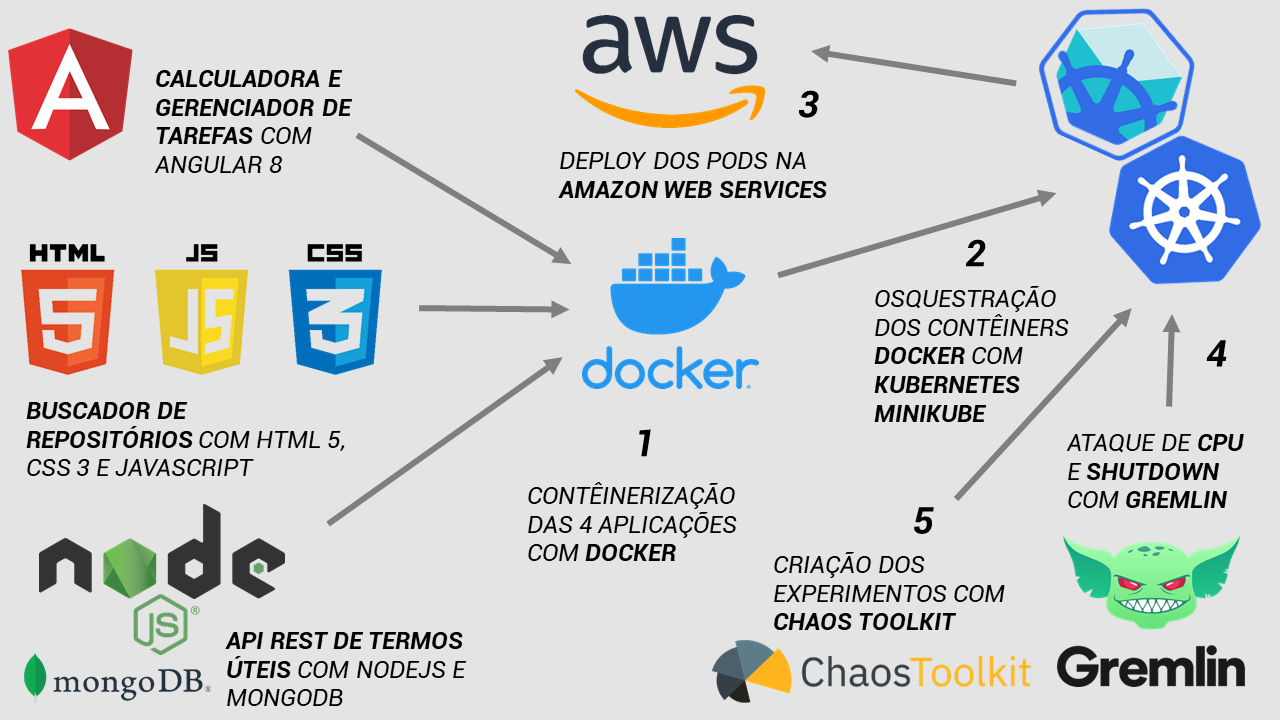
## 4.1 – Descrição do Problema

## 4.2 – Solução Proposta

### 4.2.1 – Gremlin na Prática

## 4.3 – Arquitetura do Projeto

A figura abaixo representa a arquitetura de todo o projeto, desde a criação das aplicações até a execução e criação dos ataques. O processo é dividido em 5 etapas após a criação das 4 aplicações. 1 – Subir as 4 aplicações em contêineres. 2 – Usar o Minikube para instalação do Kubernetes localmente afim de gerir os contêineres Docker. 3 – Subir os *pods* no Amazon Web Services. 4 – Execução de ataques de CPU e Shutdown com Gremlin. 5 – Criação e execução de ataques com o Chaos Toolkit.



**Figura 2 - Arquitetura do Projeto**

Fonte: Autores.

### 4.3.1 – Calculadora

### 4.3.2 – Gerenciador de Tarefas

### 4.3.3 – Buscador de Repositórios

### 4.3.4 – API REST de Termos Úteis

### 4.3.5 – Contêinerização das Aplicações

### 4.3.6 – Osquestração dos Contêineres

### 4.3.7 – Deploy dos Pods

### 4.3.8 – Execução dos Testes

### 4.3.9 – Criação dos Testes

# 5 – CONCLUSÃO

## 5.1 – Contribuições

## 5.2 – Trabalhos Futuros

# REFERÊNCIAS BIBLIOGRÁFICAS

**Links Acessados**

Chaos Toolkit. **Chaos Tookit Site.** Disponível em: <https://docs.chaostoolkit.org/>. Acesso em: 15 de novembro de 2019.

GitHub Chaos Toolkit. **Chaos Tookit Documentation.** Disponível em: <https://github.com/chaostoolkit/chaostoolkit/>. Acesso em: 15 de novembro de 2019.

Gremlin. **Gremlin Documentation.** Disponível em: <https://www.gremlin.com/docs/>. Acesso em: 15 de novembro de 2019.

W3Schools. **HTML Tutorial.** Disponível em:

<https://www.w3schools.com/html/>. Acesso em: 15 de novembro de 2019.

MDN Web Docs. **HTML5 Documentation.** Disponível em:

<https://developer.mozilla.org/pt-BR/docs/Web/HTML/HTML5/>. Acesso em: 15 de novembro de 2019.

W3Schools. **CSS Tutorial.** Disponível em:

<https://www.w3schools.com/css/>. Acesso em: 15 de novembro de 2019.

Bootstrap. **Bootstrap Documentation.** Disponível em:

<https://getbootstrap.com/docs/4.3/getting-started/introduction/>. Acesso em: 15 de novembro de 2019.

MDN Web Docs. **Documentação JavaScript.** Disponível em:

<https://developer.mozilla.org/pt-BR/docs/Aprender/JavaScript/>. Acesso em: 15 de novembro de 2019.

Node.js. **Documentação Node.js.** Disponível em:

<https://nodejs.org/pt-br/about/>. Acesso em: 15 de novembro de 2019.

Angular. **Documentação Angular.** Disponível em:

<https://angular.io/docs/>. Acesso em: 16 de novembro de 2019.

Docker. **Documentação Docker.** Disponível em:

<https://docs.docker.com/>. Acesso em: 16 de novembro de 2019.

Mundo Docker. **O que é Docker?** Disponível em: <https://www.mundodocker.com.br/o-que-e-docker/>. Acesso em: 16 de novembro de 2019.

Kubernetes Concepts. **O que é um pod?** Disponível em: <https://kubernetes.io/docs/concepts/workloads/pods/pod/>. Acesso em: 16 de novembro de 2019.

Red Hat. **O que é Kubernetes?** Disponível em: <https://www.redhat.com/pt-br/topics/containers/what-is-kubernetes/>. Acesso em: 16 de novembro de 2019.

Kubernetes. **Site Oficial.** Disponível em: < https://kubernetes.io/pt/docs/home/>. Acesso em: 16 de novembro de 2019.

Kubernetes. **Getting Started – Installing Kubernetes with Minikube.** Disponível em: < https://kubernetes.io/docs/setup/learning-environment/minikube/>. Acesso em: 16 de novembro de 2019.

AWS. **Documentação da AWS.** Disponível em: <https://docs.aws.amazon.com/index.html?nc2=h\_ql\_doc\_do\_v />. Acesso em: 16 de novembro de 2019.

Site Oficial do AWS. **O que é AWS?** Disponível em: <https://aws.amazon.com/pt/what-is-aws/>. Acesso em: 16 de novembro de 2019.