



UFMT

UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

**MONITORAMENTO DE UM AMBIENTE
COMPUTACIONAL, UTILIZANDO ZABBIX EM
CONTÊINERES DOCKER**

JOÃO VICTOR BARBOSA CHIROLI

CUIABÁ - MT

2021



UNIVERSIDADE FEDERAL DE MATO GROSSO
INSTITUTO DE COMPUTAÇÃO
COORDENAÇÃO DE ENSINO DE GRADUAÇÃO EM
CIÊNCIA DA COMPUTAÇÃO

**MONITORAMENTO DE UM AMBIENTE
COMPUTACIONAL, UTILIZANDO ZABBIX EM
CONTÊINERES DOCKER**

JOÃO VICTOR BARBOSA CHIROLI

Orientador: Prof. Dr. Roberto Benedito de Oliveira Pereira

Monografia apresentada ao Curso de Ciência da
Computação, do Instituto de Computação da Uni-
versidade Federal de Mato Grosso, como requisito
para obtenção do título de Bacharel em Ciência da
Computação

CUIABÁ - MT

2021

Este trabalho é dedicado aos meus familiares e amigos por sempre acreditarem em mim e me apoiarem, aos meus professores que me ensinaram os seus conhecimentos e especialmente minha mãe, meu pai e meus avós por sempre torcerem por mim.

AGRADECIMENTOS

Gostaria de agradecer toda a minha família, especialmente aos meus avós José Barbosa e Tereza Zafaneli que me apoiaram, agradeço também aos meus pais Maria Dilecia Barbosa e João Nery que sempre torceram por mim.

Agradeço a todos os meus amigos dos "Donos da Bola": Popo, Jão, Alberto, Sebas e Heitor, por sempre estarem comigo nos momentos bons e ruins.

Agradeço pelas minhas amizades que estiveram comigo ao longo da vida Hyuri Carvalho, Gabriel Kaiano, Letícia Corrêa, Natalia Petrini, Italo Alcântara e ao meu primo Matheus Chiroli, todos mesmo que em alguns momentos distantes, a todo momento me impulsionaram a sempre melhorar.

Agradeço à todos os professores do Instituto de Computação que auxiliaram no meu conhecimento social e intelectual. Agradeço ao meu orientador Prof. Dr. Roberto Benedito de Oliveira Pereira, pelos conhecimentos transmitidos, pela paciência e incentivo na orientação deste trabalho.

Agradeço aos amigos e colegas que fiz durante essa passagem pelo Instituto de Computação em destaque para: João Gabriel, Popi, Sette, Nil, Digao, Louis, Muller, Petrucci, Confuso e o monstro sagrado Newton Miotto.

Gostaria de agradecer também o Gabriel Vilela e a Infomach, que me apresentaram a cultura DevOps que foi fundamental para elaboração deste trabalho.

*O trabalho duro ganha do talento
quando o talento não trabalha duro*

Kevin Durant

RESUMO

Este trabalho apresenta a implantação do monitoramento de um ambiente computacional, utilizando zabbix em contêineres Docker, com a finalidade de fornecer aplicações de micro serviços gerenciados e utilizar um software de monitoramento não somente monitorar e controlar serviços computacionais, mas para realizar soluções e antecipações de problemas relacionados a aplicações e equipamentos de rede, no decorrer do projeto foram feitos levantamentos bibliográficos para a fundamentação teórica sobre o conceito de virtualização, redes de computadores, monitoramento de ativos e protocolos de comunicação entre computadores. Ao final do levantamento foi criada uma tabela comparativa entre todas as soluções pesquisadas com a finalidade de organizar e classificar os critérios para a escolha da ferramenta. Deste modo, foi escolhida a ferramenta *open-source* Zabbix para supervisionar os parâmetros de rede, além disso, foi utilizado o Docker para criação e configuração dos contêineres. A realização deste projeto foi feita em três etapas, sendo a primeira: a instalação de micro serviços Zabbix através de linhas de comando Docker; a segunda para instalação de múltiplos serviços Zabbix utilizado o Docker Compose e a terceira etapa foi a implantação dos serviços Zabbix através do Docker Compose criado pelo próprio autor, ao final, foi constatado que os objetivos foram atingidos, desde a análise das principais ferramentas e soluções a serem implantados até a validação e homologação dos serviços.

Palavras-chaves: Docker, Zabbix, Monitoramento, SLA.

ABSTRACT

This work presents the implantation of a computational environment through Docker containers with Zabbix, containing a source to provide managed micro services applications and use monitoring software in order to not only monitor and control computational services, but to carry out solutions and anticipation of problems related to network applications and equipment. During the course of the project, bibliographic surveys were made for the theoretical foundation on the concept of virtualization, computer networks, asset monitoring and communication protocols between computers. At the end of the survey, a comparative table was created between all the research solutions with a method of organizing and classifying the criteria for choosing the tool. In this way, an open source Zabbix tool was chosen to supervise the network parameters, in addition , Docker was used to create and configure the containers. The realization of this project was done in three stages, the first: the installation of Zabbix micro services through Docker command lines, the second: installation of multiple Zabbix services using Docker Compose and the third step was the deployment of Zabbix services through Docker Compose created by the author himself. At the end, it was found that the objectives were achieved, from an analysis of the main tools and solutions to be implemented until the validation and approval of the services.

Key-words: Docker, Zabbix, Monitoring, SLA.

SUMÁRIO

	1 INTRODUÇÃO	1
1.1	Objetivo Geral	4
1.2	Objetivos Específicos	4
1.3	Preparação capitular geral	5
	2 FUNDAMENTAÇÃO TEÓRICA	6
2.1	Sistema de Virtualização	6
2.1.1	<i>Virtualização por Hypervisors</i>	8
2.1.2	Virtualização sobre linguagem de programação	9
2.1.3	Virtualização em nível de Sistema Operacional	10
2.2	DOCKER	11
2.2.1	Arquitetura Docker	12
2.2.1.1	<i>Daemon Docker</i>	13
2.2.1.2	Cliente Docker	13
2.2.1.3	Registros Docker	15
2.2.1.4	Objetos Docker	16
2.3	Soluções de Monitoramento	16
2.3.1	Arquitetura de gerenciamento de redes e SNMP	17
2.3.1.1	Gerenciamento de Configuração	18
2.3.1.2	Gerenciamento de Falhas	19
2.3.1.3	Gerenciamento de Contabilização	19
2.3.1.4	Gerenciamento de Desempenho	20
2.3.1.5	Gerenciamento de Segurança	20
2.3.2	Modelo de gerenciamento	20
2.3.2.1	Gerente	21
2.3.2.2	Agente	21
2.3.2.3	Protocolo SNMP	21
2.3.2.4	MIB	22
2.3.2.5	Operações utilizando as variáveis MIB	23
2.3.3	Ferramentas de monitoramento	28
2.3.4	Arquitetura de um <i>software</i> de monitoramento	28
2.3.4.1	<i>Software</i> de apresentação para o usuário (interface)	28

2.3.4.2	<i>Software</i> para gerenciamento da rede (aplicação)	29
2.3.4.3	<i>Software</i> de suporte (banco de dados e comunicação)	29
2.3.5	Cacti	29
2.3.6	The Dude	31
2.3.7	Nagios	32
2.3.8	Zabbix	35
3	MATERIAIS E MÉTODOS	38
3.1	Cenário de Teste	38
3.2	Ferramenta Escolhida	39
3.2.1	<i>Hardware</i> e <i>Software</i> utilizado na solução escolhida	40
3.2.1.1	Material escolhido	40
3.2.2	Etapas para a implantação da solução de monitoramento	41
4	RESULTADOS	44
4.1	Etapa 1: Implantação de serviços Zabbix através de linhas de co- mando Docker	44
4.2	Etapa 2: Implantação de serviços Zabbix através do Docker-compose	49
4.3	Etapa 3: Implantação e comunicação dos serviços Zabbix através do Docker-compose criado pelo próprio autor	54
5	CONCLUSÕES	67
	REFERÊNCIAS	70

LISTA DE ILUSTRAÇÕES

Figura 1 – Cada servidor executando apenas uma aplicação em um único Sistema Operacional. Fonte: Próprio Autor.	2
Figura 2 – Representação do lado esquerdo uma arquitetura tradicional sem virtualização e ao lado direito, um modelo de arquitetura virtualizada. Fonte: Livre (2015).	7
Figura 3 – Representação da arquitetura Hosted e Bare Metal. Fonte: Quora (2020).	9
Figura 4 – Estrutura de uma aplicação JAVA. MarkOneTools (2019).	10
Figura 5 – Organização de uma containerização com duas aplicações. Fonte: Próprio Autor.	10
Figura 6 – Arquitetura Docker. Fonte: Docker (2020).	12
Figura 7 – Distribuição em camadas do Docker <i>Engine</i> . Fonte: Aquasec (2020).	13
Figura 8 – Painel do Docker <i>Desktop</i> em execução no Windows 10. Fonte: Próprio Autor.	14
Figura 9 – Preços do Docker Hub. Fonte: Docker (2020).	15
Figura 10 – Comunicação em camadas do protocolo SNMP. Fonte: DPSTelecom (2001).	22
Figura 11 – Subárvore da OID 1.3.6.1.2. Fonte: Cavaleiro (2020).	23
Figura 12 – Tabela de descrições de objetos da MIB. Fonte: Castro, Carvalho e Mendes (2013).	25
Figura 13 – Grupos de RMON MIB. Fonte: Castro, Carvalho e Mendes (2013).	27
Figura 14 – Exemplo gráfico de monitoramento no Cacti . Fonte: Próprio Autor.	30
Figura 15 – Topologia do cenário de teste. Fonte: Próprio Autor.	39
Figura 16 – Comparativo das funções disponibilizadas pelas ferramentas. Fonte: Próprio Autor.	39
Figura 17 – Repositório Zabbix. Fonte: Próprio Autor.	46
Figura 18 – Pull do MYSQL 8. Fonte: Próprio Autor.	47
Figura 19 – Interface inicial do Zabbix Web. Fonte: Próprio Autor.	49
Figura 20 – Exemplo de arquivo YAML. Fonte: Próprio Autor.	50
Figura 21 – Apresentação dos comandos Linux e do arquivo YAML. Fonte: Próprio Autor.	51
Figura 22 – Apresentação do comando <i>git branch</i> . Fonte: Próprio Autor.	52
Figura 23 – Apresentação do comando <i>git tag -list '5.2.*'</i> . Fonte: Próprio Autor.	52
Figura 24 – Apresentação do comando Linux <i>cp</i> e <i>ls</i> . Fonte: Próprio Autor.	52

Figura 25 – Apresentação de substituição para um arquivo mais recente. Fonte: Próprio Autor.	53
Figura 26 – <i>Pull</i> das imagens. Fonte: Próprio Autor.	53
Figura 27 – <i>Run</i> dos contêineres. Fonte: Próprio Autor.	54
Figura 28 – Imagem dos 3 arquivos YAML . Fonte: Próprio Autor.	55
Figura 29 – Imagem dos 3 arquivos YAML . Fonte: Próprio Autor.	55
Figura 30 – Imagem da execução dos serviços. Fonte: Próprio Autor.	61
Figura 31 – Imagem dos contêineres rodando. Fonte: Próprio Autor.	61
Figura 32 – Imagem do monitoramento do host em que está no Zabbix <i>Server</i> . Fonte: Próprio Autor.	62
Figura 33 – Imagem do monitoramento do host Windows-10 e Zabbix <i>Server</i> . Fonte: Próprio Autor.	63
Figura 34 – Imagem que mostra a comunicação entre o Zabbix <i>Proxy</i> e Zabbix <i>Server</i> . Fonte: Próprio Autor.	63
Figura 35 – Imagem que o Zabbix <i>Server</i> consegue receber dados de todos estes hosts. Fonte: Próprio Autor.	64
Figura 36 – Imagem de itens monitorados do Zabbix <i>Server</i> . Fonte: Próprio Autor.	65
Figura 37 – Imagem de triggers default utilizadas no monitoramento do Zabbix <i>Server</i> . Fonte: Próprio Autor.	65
Figura 38 – Imagem da configuração de serviços no Zabbix. Fonte: Próprio Autor.	66
Figura 39 – - Imagem da utilização de serviços no Zabbix. Fonte: Próprio Autor. .	66

LISTA DE CÓDIGO

4.1	Comando para atualizar o repositório de aplicativos do Linux.	45
4.2	Comando para instalar o suporte ao HTTPS.	45
4.3	Comando para adicionar chave do repositório.	45
4.4	Comando para adicionar o repositório do Docker.	45
4.5	Comando para atualizar a lista dos aplicativos no repositório	45
4.6	Comando para adicionar chave do repositório	45
4.7	Comando para verificar o status do serviço Docker.	45
4.8	Comando para criar o diretório para o banco de dados.	46
4.9	Comandos para baixar as imagens dos contêineres.	47
4.10	Comando Docker para criar o contêiner do banco de dados	47
4.11	Comando Docker para criar o contêiner do Zabbix-Server	48
4.12	Comando Docker para criar o contêiner do Zabbix-web	48
4.13	Comandos para a execução do docker compose e verificação da versão do docker-compose	51
4.14	Comando para instalar o GIT	51
4.15	Comando para clonar o repositório do Zabbix	51
4.16	Comando do console para mudar de diretório e listar arquivos	51
4.17	Comando git para substituir os arquivos atuais	52
4.18	Comando git para listar todas as versões 5.2	52
4.19	Comando Linux <i>cp</i>	52
4.20	Comando <i>sed</i>	52
4.21	Comando de execução do arquivo YAML	53
4.22	Comando de execução do arquivo YAML	54
4.23	Arquivo YAML contendo o serviço MYSQL	56
4.24	Arquivo YAML contendo o serviço Zabbix Server	56
4.25	Arquivo YAML contendo os serviços Zabbix Front e Zabbix Agent	57
4.26	Arquivo YAML do Zabbix Proxy	58
4.27	Arquivo YAML do Zabbix Agent	59
4.28	Execução do arquivo YAML	60

LISTA DE ABREVIATURAS E SIGLAS

SLA	<i>Service Level Agreement</i>
SO	Sistema Operacional
IP	<i>Internet Protocol</i>
ID	Identificação
TI	Tecnologia da Informação
YAML	<i>YAML Ain't Markup Language</i>
VM	<i>Virtual Machine</i>
VMM	<i>Virtual Machine Manager</i>
GB	<i>Gigabyte</i>
ISO	<i>International Organization for Standardization</i>
GHZ	<i>GigaHertz</i>
HP	<i>Hewlett-Packard Company</i>
GPL	<i>General Public License</i>
CPU	<i>Central Process Unit</i>
SSH	<i>Secure Shell</i>

JMX	<i>Java Management Extensions</i>
API	<i>Application Programming Interface</i>
CLI	<i>Command-line Interface</i>
TCP	<i>Transmission Control Protocol</i>
IPMI	<i>Intelligent Platform Management Interface</i>
HTTP	<i>Hypertext Transfer Protocol</i>
SNMP	<i>Simple Network Management Protocol</i>
IBM	<i>International Business Machines Corporation</i>
LDAP	<i>Lightweight Directory Access Protocol</i>
SMS	<i>Short Message Service</i>
JPEG	<i>Joint Photographic Experts Group</i>
WEB	<i>World Wide Web</i>
US	<i>United States</i>
JVM	<i>Java Virtual Machines</i>
KVM	<i>Kernel-based Virtual Machine</i>
DNS	<i>Domain Name System</i>
VPN	<i>Virtual Private Network</i>
GNU	<i>GNU's Not Unix</i>
QoS	<i>Quality of Service</i>
OSI	<i>Open System Interconnection</i>
UDP	<i>User Datagram Protocol</i>
ICMP	<i>Internet Control Message Protocol</i>

CAPÍTULO 1

INTRODUÇÃO

Com o crescente aumento das navegações online e o avanço da tecnologia, empresas e instituições evidenciaram a necessidade de utilizarem em suas plataformas, mecanismos com alta compatibilidade de aplicações que lide com eventuais falhas e proporcione um ambiente computacional moderno, com intuito de manter os serviços disponíveis o máximo de tempo possível. Visto que, quanto maior o período que uma empresa fica desconectada da Internet, maior é o seu prejuízo, em alguns casos podendo levar a falência da corporação.

Um dado que justifica essa afirmação é um estudo realizado pela One Day Testing (SANT'ANA, 2017), empresa especializada em testes de produtos digitais, que monitorou 43 *e-commerces* durante a Black Friday, 83.7% apresentaram instabilidade em algum momento e 32% dos sites ficaram, juntos, 4 horas e 16 minutos fora do ar, resultando em uma perda de faturamento de pelo menos 6.4 milhões de reais.

Desta forma, com intuito de diminuir os gastos em relação aos prejuízos envolvendo falta de disponibilidade de recursos e compatibilidade de aplicações, as corporações têm intensificado os investimentos em tecnologias computacionais que solucionem ou diminuam esses problemas, gerando assim, mais lucro.

Nesse mesmo sentido, os métodos atuais de monitoramento podem ser implementados tanto por um *appliance* ou quanto por uma solução *open-source* em um servidor dedicado para essa finalidade, como por exemplo o Zabbix.

Logo, para manter um sistema de monitoramento era necessário realizar o armazenamento de apenas um serviço por servidor, como apresentado na Figura 1. Assim, quanto maior o número de serviços providos no sistema, maior seria o número de servidores para armazená-los.



Figura 1 – Cada servidor executando apenas uma aplicação em um único Sistema Operacional. Fonte: Próprio Autor.

Ao longo do tempo, esse método dedicado foi se tornando inviável devido a essa arquitetura tinha uma capacidade de aproveitamento do *hardware* limitada, além dos custos financeiros com energia, manutenção e compra de aparelhos. Como exemplo, um servidor com apenas uma aplicação tinha que estar configurado de tal forma que sua capacidade não atingisse todo seu poderio computacional, visto que a configuração era feita dessa maneira para evitar que em dias de alto consumo de *hardware*, ocasionados pela elevada quantidade de acessos, o dispositivo não se sobrecarregasse e por conta disso, ficasse indisponível ou instável. Logo, o mais comum para esse tipo de situação é a utilização de *hardwares* mais robustos que conseguissem operar com o máximo de sua capacidade e não sucedessem a eventuais falhas.

Assim, para suprir o desperdício computacional por ociosidade de recursos de hardware provenientes dessa arquitetura, foi criada a tecnologia de virtualização. Com o objetivo de otimizar o uso de infraestruturas de TI existentes, proporcionando a execução de um ou mais sistemas operacionais em uma única máquina física. Além disso, com a virtualização é possível otimizar o uso dos recursos de uma máquina física por meio da distribuição desses recursos entre vários usuários ou ambientes (FILHO; PEREIRA, 2018).

Não obstante, o processo de virtualização só é possível através da utilização de uma máquina virtual, um *software* que executa programas como um computador real. Assim, empresas não precisam comprar um servidor por serviço, pois passou a existir a

possibilidade de particionamento de recursos do servidor e execução de programas em vários tipos e versões de sistemas operacionais diferentes numa mesma máquina física.

Desta forma, a adoção dessa tecnologia aumentou a eficiência dos servidores (e aposentou tantos outros), resultando em uma redução dos custos associados à aquisição, configuração, refrigeração, manutenção e espaços em racks dos *data centers*.

Portanto, para prover aumento na eficiência da utilização de dispositivos de infraestrutura de redes, foram criadas arquiteturas de software e hardware baseadas em contêineres. Os contêineres provêm isolamento da aplicação, dependências e recursos de maneira similar a uma máquina virtual, mas de maneira mais leve por compartilhar o kernel com o sistema hospedeiro. Esta combinação se mostra como uma alternativa ao desenvolvimento de *software* monolítico tradicional, pois a natureza leve e efêmera dos contêineres pode ser usada para provisionar recursos e isolamento para micros serviços (FILHO, 2016).

Esta nova tecnologia consegue suprir necessidades relacionadas a falta de compatibilidade de aplicações de software, por conter todas as funcionalidades necessárias para executar uma aplicação, sendo assim, a arquitetura de contêineres permite que o desenvolvedor crie uma aplicação em um sistema operacional Linux e aplique testes em um sistema operacional Windows de maneira leve, rápida e sem erros de conciliabilidade de sistemas.

Logo, os contêineres se apresentam como uma solução que possibilita melhor controle sobre o uso de recursos, agilidade na hora de criação e remoção de aplicações, maior facilidade na hora de trabalhar com diferentes versões de linguagens e bibliotecas, além de ser mais leve que as VMs. Sendo a plataforma Docker é um exemplo de plataforma encarregada de fornecer um conjunto de serviços e aplicações para promover a criação, exclusão e edição de contêineres.

Além disso, atualmente existe uma dependência considerável por serviços computacionais para realizarem tarefas de diversas naturezas, inclusive tarefas críticas, cujas falhas causariam prejuízos materiais, financeiros ou até mesmo perda de vidas. No intuito de mitigar esses prejuízos, são utilizadas diversas técnicas que visam garantir a disponibilidade desses serviços (FILHO, 2004).

Desse modo, o objetivo principal de um sistema que pretende promover alta disponibilidade é prevenir e eliminar todos os pontos suscetíveis a falha de um ecossistema computacional. Isso deve incluir vários planos de ação que foram testados e implementados, prontos para reagir de forma independente e imediata a qualquer e todas as perturbações, interrupções e falhas de serviço, incluindo irregularidades de hardware, software e aplicativos. Sendo assim, esse recurso se tornou indispensável para um ambiente computacional.

Para mais, com a intenção de evitar grandes períodos de inatividade de um sistema, é possível aplicar técnicas com intuito de diminuir a indisponibilidade, tanto de *hardware* como de *software*. Uma solução é o uso de ferramentas de monitoramento, que são encarregados de medir a disponibilidade e desempenho de componentes de infraestrutura, aplicações e gerar indicadores estratégicos para negócios e instituições.

Logo, é de suma necessidade o uso de uma plataforma que tenha um controle unificado sobre os processos e recursos utilizados na rede para atender da melhor maneira possível às necessidades de seus usuários, possibilitando flexibilidade tanto para monitorar equipamentos de *hardware* físico, em nuvem ou serviços de *software*.

1.1 Objetivo Geral

Dessa forma, tendo em vista todas as prerrogativas abordadas, o objetivo geral deste trabalho é realizar o monitoramento de um ambiente computacional por meio de contêineres Docker com Zabbix. A tecnologia Docker responsável por fornecer aplicações relacionadas a contêineres, e a utilização do software de código aberto Zabbix, tem o intuito de não somente monitorar e controlar serviços computacionais, mas para realizar soluções e antecipações de problemas relacionados a aplicações e equipamentos de rede. Além disso, o trabalho pretende apresentar como contêineres e micros serviços Docker auxiliam na administração de uma rede de infraestrutura, permitindo a criação mais rápida de processos e extinguindo a necessidade de executar processos manuais, os quais levariam mais tempo para serem executados.

1.2 Objetivos Específicos

Para tal, esse trabalho foi segmentado nos seguintes objetivos específicos, sendo eles:

- Realizar um levantamento bibliográfico para a fundamentação teórica sobre o conceito de monitoramento, alta disponibilidade em sistemas computacionais, virtualização, virtualização em contêiner, conceitos sobre infraestrutura em forma de código e micros serviços containerizados;
- Analisar os conceitos de ambientes monitorados e apresentar o Zabbix como solução para a realização de um sistema monitorado;
- Implantar um cenário de monitoramento, utilizando contêineres Docker, com dependência de algumas aplicações externas;

- Avaliar e homologar a implementação de um ambiente computacional gerenciado por Zabbix através de contêineres Docker em uma página modelo;
- Homologar uma solução Docker que implemente um monitorador de ativos de rede por meio do Zabbix customizável;

1.3 Preparação capitular geral

Este trabalho está dividido da seguinte maneira, no capítulo 1: foi feita a introdução acerca dos conceitos que serão abordados e discutidos intrinsecamente no decorrer do documento. Além disso, este capítulo tem o intuito de familiarizar o leitor com as definições da tecnologia apresentada, para melhor entendimento do mesmo.

No capítulo 2, foi apresentada a fundamentação teórica acerca das principais técnicas e ferramentas de virtualização, alta disponibilidade em hardware e software, descrição de contêiner e da ferramenta Docker, não obstante, foi apresentado Zabbix como software de monitoramento e como cada uma dessas tecnologias se integra.

No Capítulo 3, foram definidos os materiais e métodos utilizados no processo de investigação do problema e a preparação dos cenários para realização dos testes. Também foram feitas, descrições dos métodos utilizados nos experimentos, além de conter explicações técnicas sobre as métricas empregadas nos estudos de casos.

A seguir, no Capítulo 4, foram feitas as descrições dos resultados obtidos em cada um dos experimentos. Contendo também, elucidações das ações realizadas.

E no capítulo 5, foi exposto as considerações finais em relação ao estudo, comparando os resultados obtidos com os objetivos pretendidos. Relatou-se também as dificuldades encontradas, os potenciais pontos de melhoria e trabalhos futuros que podem ser feitos nesta área.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão abordados os principais conceitos teóricos utilizados neste trabalho, e as soluções existentes para estas finalidades. Inicialmente será feita uma abordagem geral sobre a virtualização: histórico, principais técnicas e os principais *softwares* de virtualização.

Em seguida, serão descritos os conceitos da ferramenta Docker, e como esta tecnologia auxilia na criação e virtualização de contêineres. O próximo tema a ser indagado é sobre a descrição de algumas soluções de monitoramento disponíveis no mercado.

Com os conteúdos essenciais devidamente apresentados, o capítulo é finalizado com a apresentação do instrumento de monitoramento utilizado neste trabalho, o Zabbix, passando por sua história, arquitetura e mostrando como o *softwares* pode auxiliar na prevenção e antecipação de problemas em ambientes computacionais comuns e altamente disponíveis.

2.1 Sistema de Virtualização

Como já foi apresentado, o objetivo das organizações gira em torno da construção de plataformas digitais que consigam crescer em ambientes turbulentos, através de um ecossistema computacional estável e flexível. Neste contexto, a virtualização torna-se uma peça chave, pois possibilita o crescimento de infraestrutura de *hardware* e *software* através

da existência de ambientes mistos com diferentes Sistemas Operacionais em execução. Sendo assim, a virtualização permite que em uma mesma máquina sejam executados simultaneamente dois ou mais ambientes distintos e isolados (MENEZES, 2008).

A virtualização teve seu início ainda na década de 70 quando a IBM criou o sistema VM/370, sendo composto de um monitor de máquina virtual, instalado diretamente sobre o *hardware* e implementando multiprogramação (ALVES, 2013). O sistema VM/370, um VMM (Virtual Machine Monitor) possui algumas instruções extras que permitiam a execução de vários sistemas operacionais trabalharem de forma isolada sobre um único *hardware*, sendo assim o primeiro sistema a tentar executar a virtualização (SANTOS, 2018).

No entanto com o avanço da computação, o barateamento de *hardware* e por conta da popularização de computadores pessoais, o estudo da virtualização não tinha grande importância até a década de 90, antes desse período grande parte das empresas possuíam servidores e stacks de TI de apenas um fornecedor, o que impossibilitava a execução de aplicações ligadas ao *hardware* de fornecedores diferentes. À medida que as empresas atualizavam os ambientes de TI com servidores comuns, os sistemas operacionais e aplicações mais econômicas oferecidas por uma variedade de fornecedores, elas se viam limitadas pela subutilização do *hardware* físico.

Logo, na década de 90, a virtualização, por meio da consolidação de servidores, tornou-se solução para problemas de particionamento e aplicações de vários tipos, contendo diferentes versões de Sistemas Operacionais como mostrado na Figura 2. Ademais, a inovação tecnológica permite que um computador execute o trabalho de vários computadores, com a ajuda de compartilhamento de recursos de um único *hardware* em vários ambientes, ocasionando em uma ampla aplicabilidade de setores computacionais, contribuindo para a diminuição da dependência com fornecedores de equipamentos e tornando-se a base da *cloud computing*.

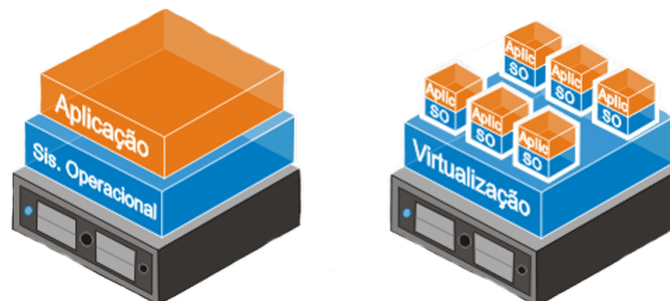


Figura 2 – Representação do lado esquerdo uma arquitetura tradicional sem virtualização e ao lado direito, um modelo de arquitetura virtualizada. Fonte: Livre (2015).

Atualmente, a virtualização é dividida em duas categorias, a paravirtualização e a virtualização completa. A virtualização completa é a primeira solução de *software* referente à virtualização de servidores e desenvolvida no ano de 1966 pela IBM, este processo utiliza o *hypervisor* para simular completamente o *hardware* da máquina física, para permitir que um SO convidado não modificado seja executado isoladamente. Já a paravirtualização, é a interação do SO convidado com o *hypervisor* para aumentar o desempenho e produtividade.

Em vista disso, um estudo realizado pela Microsoft em 2018, propôs a divisão deste tipo de virtualização em quatro grupos principais, sendo eles: a primeira é a virtualização de área de trabalho, que permite que um servidor centralizado sirva e gerencie desktops individualizados. A segunda é a virtualização de rede, desenvolvida para dividir a largura de banda da rede entre canais independentes que são posteriormente atribuídos a servidores ou dispositivos específicos. A terceira categoria é a virtualização de *software*, que separa aplicativos do *hardware* e do sistema operacional. E, por fim, a quarta é a virtualização de armazenamento, que combina múltiplos recursos de armazenamento de rede em um único dispositivo de armazenamento que pode ser acessado por diversos usuários. Portanto, para devida compreensão das terminologias abordadas no trabalho, é essencial o conhecimento da virtualização de *software* e suas três categorias básicas:

- **Virtualização por *hypervisors*;**
- **Virtualização sobre linguagem de programação;**
- **Virtualização em nível de Sistema Operacional.**

2.1.1 Virtualização por *Hypervisors*

A virtualização neste método ocorre em nível de *hardware*. Um *hypervisor* é o *software* responsável pelo gerenciamento de máquinas virtuais que são executadas sobre o Sistema Operacional (SO) hospedeiro, ou seja, da própria máquina física. Cada máquina virtual possui o seu próprio SO (conhecido como convidado) com o seu próprio núcleo, aplicações e suas dependências (binários e bibliotecas) (FILHO; PEREIRA, 2018).

Portanto, é preciso considerar que a virtualização em nível de *hardware* consiste em prover uma réplica (virtual) do *hardware* subjacente de tal forma que o SO e as aplicações podem executar como se tivessem executando diretamente sobre o *hardware* original (CARISSIMI, 2008).

Ademais, antigamente em um data center comum apenas 15% de sua capacidade era utilizada, os outros 85% restantes ficavam ociosos (VERAS, 2018). Com a virtualização o aproveitamento da capacidade dos servidores subiu para ao menos 60%.

Gerando em alguns casos redução no consumo de energia e gastos com *hardware*, proporcionando uma diminuição do impacto ambiental causado pela TI sem abrir mão da confiabilidade ou serviços.

Não menos importante, a virtualização completa possui dois tipos de *hypervisor*. O tipo 1, chamado de bare-metal e o tipo 2 chamado de hosted. O bare-metal, apresentado na Figura 3, funciona diretamente sobre o *hardware* da máquina física, ou seja, o *software* é executado em modo real, sem nenhum contato com o SO, fazendo assim uma comunicação direta entre a parte física e a parte lógica do computador. Para mais, esta ferramenta é normalmente aplicada em virtualização de servidores, possuindo o Hyper-V, KVM (*Kernel-based Virtual Machine*), VMware ESX Server e Xen Server, como as soluções mais usufruídas no mercado de TI. Já o *hypervisor* do tipo hosted, apresentado na Figura 3, funciona sobre o SO hospedeiro, tornando-se dependente do mesmo, esse método é mais aplicado em *desktop*, possuindo o VMwareplayer, VirtualBox e Virtual PC, como os principais programas.

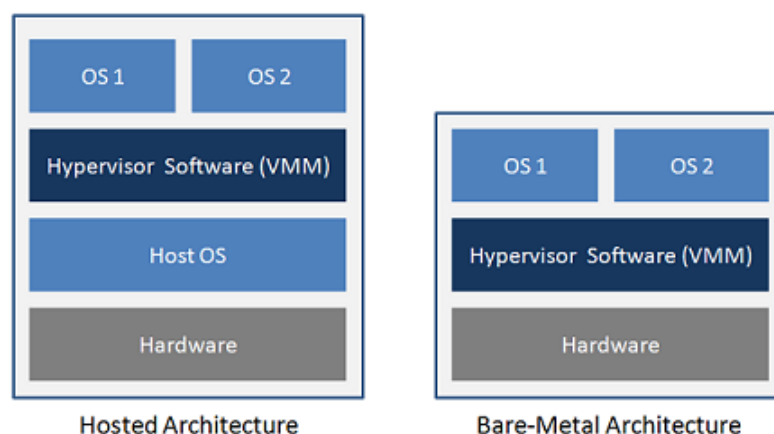


Figura 3 – Representação da arquitetura Hosted e Bare Metal. Fonte: Quora (2020).

2.1.2 Virtualização sobre linguagem de programação

Nesse processo a máquina física recebe um sistema operacional, mas há a criação de vários ambientes virtuais sobre este. Cada um destes ambientes tem acesso a determinados recursos, como espaço em disco e quotas de processamento, mas compartilham do mesmo kernel (ALECRIM, 2013). Nesse mecanismo, é atribuído na camada de virtualização, um programa de aplicação que executa sistemas e aplicações compatíveis com a sua plataforma e bibliotecas (SANTOS, 2018). O principal exemplo deste mecanismo é a máquina virtual JVM (*Java Virtual Machine*), utilizada para compilar os códigos na linguagem de programação JAVA, essa estrutura é apresentada na Figura 4.

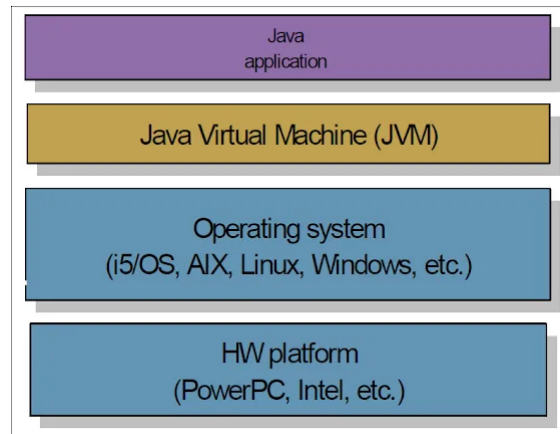


Figura 4 – Estrutura de uma aplicação JAVA. MarkOneTools (2019).

2.1.3 Virtualização em nível de Sistema Operacional

A virtualização em nível de SO também conhecida como containerização por possuir instâncias chamadas de contêineres, tem por objetivo empacotar tudo o que é necessário para executar um *software* de forma rápida e confiável em um ambiente computacional para outro, ou seja, um contêiner é uma unidade leve e isolada que executa um aplicativo no Sistema Operacional do host.

Em resumo enquanto uma virtualização tradicional fará uma emulação do *hardware*, do sistema operacional, da aplicação, de suas bibliotecas, e dos binários necessários para sua execução, a virtualização por container somente fará a emulação da aplicação, de suas bibliotecas, e de seus binários (LEITE, 2019). Portanto, essa tecnologia se caracteriza por executar instâncias de processamento isoladas umas das outras de modo que cada contêiner opera em um mesmo sistema de *software* hospedeiro e possui sua própria concentração de recursos. O esquema básico dessa arquitetura, executando 2 aplicações distintas é representada na Figura 5.

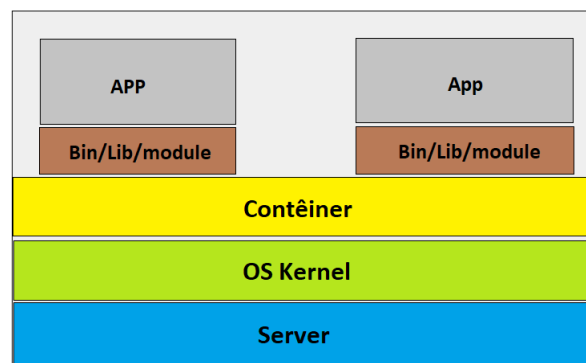


Figura 5 – Organização de uma containerização com duas aplicações. Fonte: Próprio Autor.

Desse modo, esse dispositivo lógico consegue promover isolamento, ao mesmo tempo que garante a execução consistente e portátil de aplicações, ocasionando em uma ferramenta extremamente versátil de fácil implementação e manuseio, essencial para fornecer ecossistemas computacionais de alta disponibilidade.

2.2 DOCKER

O Docker é um conjunto de produtos de plataforma como serviço que utilizam a virtualização em nível de SO para entregar *software* em pacotes chamados de contêineres. Essa tecnologia foi apresentada em 2013, durante a PyCon US, pela empresa DotCloud, companhia esta que estava prestes a falir quando disponibilizou o Docker 0.9 como *open source*. Esta iniciativa da instituição rendeu investimentos financeiros de cerca de 50 milhões de dólares, além de obter parcerias com IBM e Red Hat. Por consequência do crescimento do dispositivo, a empresa DotCloud mudou seu nome para Docker.

Desta maneira, os desenvolvedores ao utilizarem essa tecnologia buscam eliminar problemas de compatibilidade em seus programas, pois é possível lidar com contêineres como se fossem máquinas virtuais modulares e extremamente leves. Por conseguinte, pelo fato do contêiner ser executado sobre o *Kernel* da máquina *host*, não existe a necessidade de virtualização de *hardware*.

Logo, o objetivo dos contêineres é criar essa independência: a habilidade de executar diversos processos e aplicações separadamente para utilizar melhor a infraestrutura e, ao mesmo tempo, manter a segurança que você teria em sistemas separados (HAT, 2017).

Segundo o Red Hat existem algumas vantagens de se usar Docker:

- **Implementação rápida de aplicativos:** os contêineres incluem os requisitos mínimos de tempo de execução do aplicativo, reduzindo seu tamanho e permitindo que sejam implementados rapidamente;
- **Portabilidade entre máquinas:** um aplicativo e todas as suas dependências podem ser agrupados em um único contêiner que é independente da versão do *host* do *kernel* Linux, distribuição de plataforma ou modelo de implantação. Este contêiner pode ser transferido para outra máquina que executa o Docker que é executado neste dispositivo sem problemas de compatibilidade;
- **Controle de versão e reutilização de componentes:** você pode rastrear versões sucessivas de um contêiner, inspecionar diferenças ou reverter para versões anteriores. Os contêineres reutilizam componentes das camadas anteriores, o que os torna notavelmente leves;

- **Compartilhamento:** você pode usar um repositório remoto para compartilhar seu contêiner com outras pessoas. A Red Hat fornece um registro para esse propósito, e também é possível configurar seu próprio repositório privado;
- **Tamanho leve e sobrecarga mínima:** as imagens do Docker são geralmente muito pequenas, o que facilita a entrega rápida e reduz o tempo para implantar novos contêineres de aplicativos;
- **Manutenção simplificada:** as imagens do Docker são geralmente muito pequenas, o que facilita a entrega rápida e reduz o tempo para implantar novos contêineres de aplicativos.

Nos demais sub capítulos serão abordados os principais conceitos sobre a arquitetura Docker. Tal entendimento é de extrema importância para compreensão da tecnologia e como a plataforma auxilia na criação e administração de ambientes isolados.

2.2.1 Arquitetura Docker

O Docker possui uma arquitetura formada basicamente pelo Docker *Server*, *daemon*, imagens e registros. Em que o cliente Docker se comunica com o *daemon* Docker, que faz o trabalho de construção, distribuição e execução de seus contêineres, como é apresentado na Figura 6. Além disso, cada um desses itens da arquitetura Docker serão apresentados nas subseções seguintes.

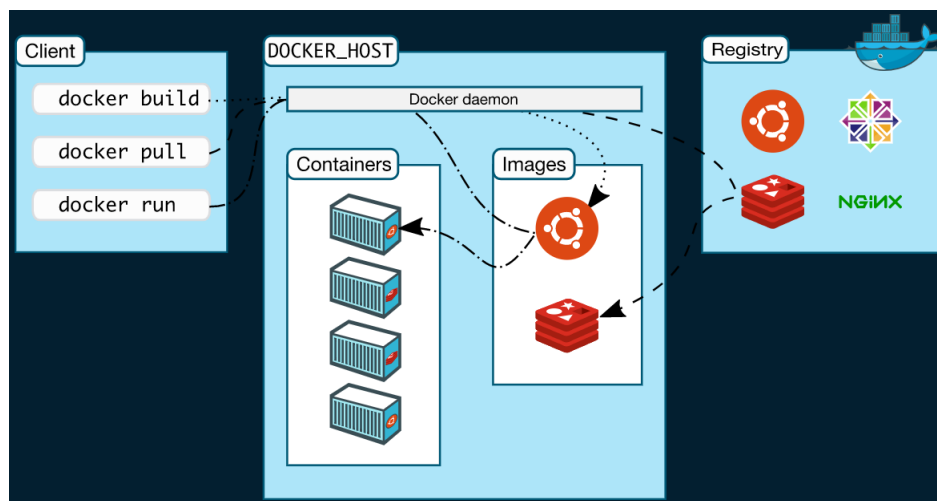


Figura 6 – Arquitetura Docker. Fonte: Docker (2020).

2.2.1.1 *Daemon Docker*

O Docker *daemon* escuta as solicitações da API Docker e gerencia objetos Docker, como imagens, contêineres, redes e volumes. Um *daemon* também pode se comunicar com outros *daemon* para gerenciar os serviços Docker (DOCKER, 2020).

2.2.1.2 *Cliente Docker*

O cliente Docker é a principal maneira pela qual muitos usuários do Docker interagem com a ferramenta. Usando comandos como *docker run*, o cliente envia esses comandos para *dockerd*, que os executa. Já o *docker* é um comando que usa a API Docker, além disso, o cliente Docker pode se comunicar com mais de um *daemon*. Na Figura 7, é mostrada de forma simplificada como a arquitetura da ferramenta é distribuída, envolvendo Cliente, *Daemon* e API.

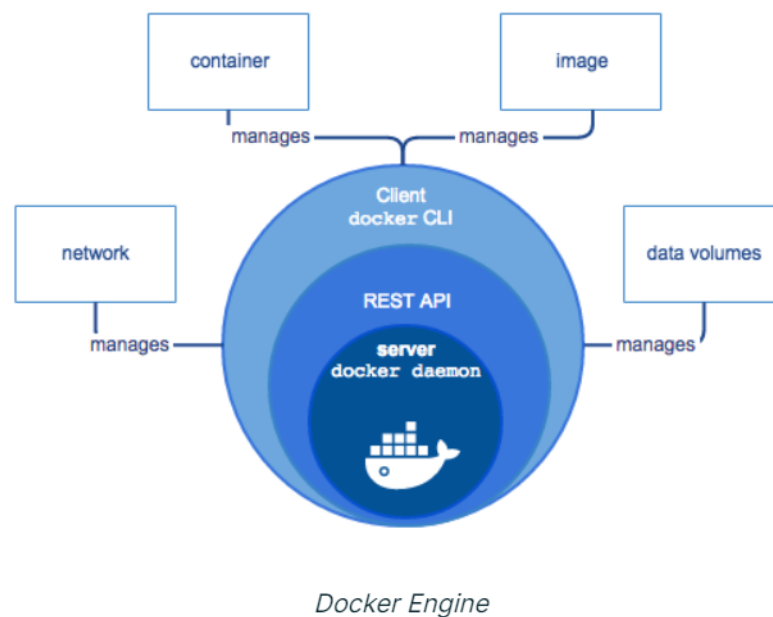


Figura 7 – Distribuição em camadas do Docker *Engine*. Fonte: Aquasec (2020).

Dessa forma, o usuário pode começar a utilização e implementação da ferramenta de várias formas, pois este dispositivo, está disponível em uma ampla variedade de plataformas, como:

- **Desktop:** Mac OS, Windows 10 e diversas distribuições Linux;
- **Servidor:** várias distribuições de Linux e Windows *Server* 2016;
- **Nuvem:** Amazon Web Services, Google Compute Platform, Microsoft Azure, IBM Cloud e muito mais.

Após realizada a instalação, é importante destacar que o Docker *Desktop* oferece uma visão rápida dos contêineres em execução em sua máquina. Sendo assim, a ferramenta fornece acesso aos logs dos contêineres, permitindo que o usuário obtenha um shell dentro do contêiner, além do gerenciamento do ciclo de vida do contêiner (interrupção, remoção, etc.). Desse modo, na Figura 8 é retratado o painel do Docker *Desktop* sendo executado em um Sistema Operacional Windows 10 e alguns contêineres de teste que foram criados, excluídos e outros que ainda estão em execução.

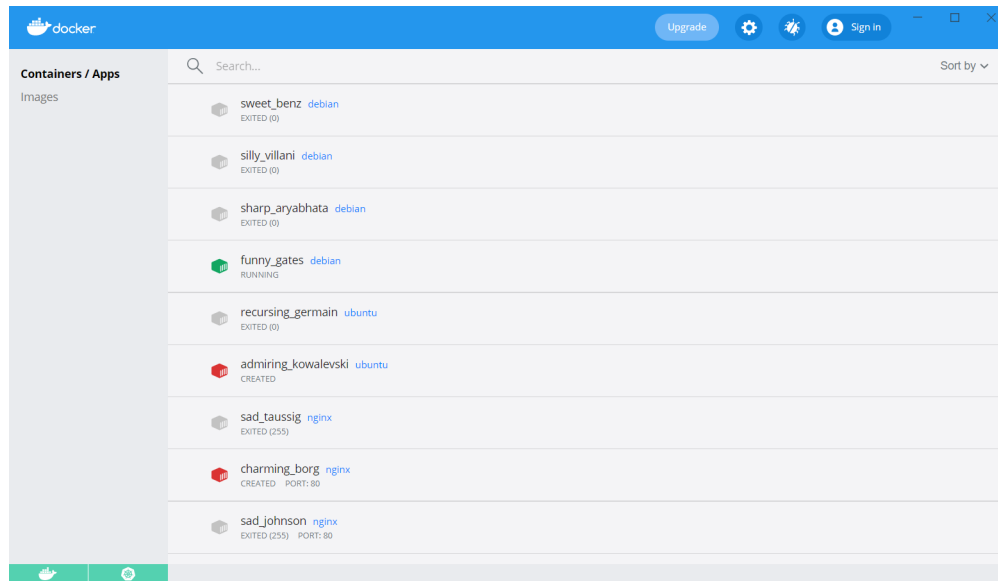


Figura 8 – Painel do Docker *Desktop* em execução no Windows 10. Fonte: Próprio Autor.

Já instalado, os comandos comuns emitidos por um cliente são:

- ***docker build***: de uma forma simplificada, o comando cria imagens Docker a partir de um Dockerfile e um “contexto”;
- ***docker pull***: extrai uma imagem ou repositório de um registro;
- ***docker run***: executa processos em contêineres isolados.

Para prover uma melhor compreensão do documento, é necessário dissertar previamente sobre alguns temas que já foram abordados e que serão explicados de maneira mais aprofundada no decorrer deste trabalho. Sendo o primeiro deles: o termo *Dockerfile*, que se trata de um documento de texto que contém todas as instruções que um usuário pode chamar na linha de comando para montar uma imagem. E o segundo: é o termo imagem, de maneira geral a instrução é a *template* que permite a execução de um contêiner, ela é responsável por vários sistemas de arquivos de camadas que ficam sobrepostas uma sobre

a outra. Essa ferramenta é primordial para construção de várias aplicações, dentre elas: CentOS, Apache, Nginx, entre outros.

A despeito disso, a execução dos comandos mais comuns da ferramenta e instruções que possibilitam a confirmação que estes contêineres foram devidamente criados está contido no capítulo de materiais e métodos, que exemplifica todos esses conceitos de uma maneira mais profunda.

2.2.1.3 Registros Docker

Um registro Docker armazena imagens Docker. O Docker Hub é um registro público que qualquer pessoa pode usar, em que o Docker está configurado para procurar imagens no Docker Hub por padrão. Qualquer pessoa pode até executar seu próprio registro privado (DOCKER, 2020). Não obstante, quando usado os comandos *docker pull* ou *docker run*, as imagens necessárias são obtidas do registro configurado. Quando é usado o comando *docker push*, a imagem utilizada é enviada para o registro configurado. Dessa forma, a utilização de registros Docker é importante para controle rigoroso de imagens armazenadas, possibilidade do usuário possuir acesso total ao canal de distribuição de imagens, além, da funcionalidade integrar o armazenamento e distribuição de imagens em seu fluxo de trabalho de desenvolvimento interno.

Porém, existem algumas limitações entre o plano gratuito e os demais planos disponíveis, como apresentado na Figura 9.



Figura 9 – Preços do Docker Hub. Fonte: Docker (2020).

2.2.1.4 Objetos Docker

Quando se usa objetos, o desenvolvedor está criando e usando imagens, contêineres, redes, volumes, *plug-ins* e entre outros objetos. As imagens são um modelo somente leitura, com instruções para criar um contêiner Docker. Frequentemente, uma imagem é baseada em outra imagem, com alguma personalização adicional.

Já um contêiner é uma instância executável de uma imagem. Qualquer pessoa pode criar, iniciar, parar, mover ou excluir um contêiner usando a API Docker ou CLI. Você pode conectar um contêiner a uma ou mais redes, anexar armazenamento a ele ou até mesmo criar uma nova imagem com base em seu estado atual.

E os serviços permitem que você dimensione contêineres em vários *daemons* do Docker, que funcionam juntos como um enxame com vários gerentes e trabalhadores. Cada membro de um *swarm* é um *daemons* do Docker e todos os daemons se comunicam usando a API Docker.

Um serviço permite que você defina o estado desejado, como o número de réplicas do serviço que devem estar disponíveis a qualquer momento. Por padrão, o serviço tem balanceamento de carga em todos os nós de trabalho. Para o consumidor, o serviço Docker parece ser um único aplicativo.

Contudo, apesar da tecnologia Docker resolver vários problemas, aplicações que abrangem múltiplos contêineres devem ser implantados em vários hosts do servidor, outra dificuldade é com a segurança dos contêineres que possui várias camadas e pode ser complexa. Dessa maneira nasce a necessidade da utilização de um orquestrador de contêiner, sendo o Kubernetes o principal sistema usado, assunto esse que será tratado em um possível trabalho futuro.

2.3 Soluções de Monitoramento

Além disso, atualmente, a confiabilidade e disponibilidade são cada vez mais desejáveis e necessários em sistemas de computação. Isto devido ao aumento da dependência das organizações de sistemas automatizados e informatizados (PEREIRA, 2004). Tolerância a falhas e alta disponibilidade são algumas vezes citadas como sinônimos. Sendo assim, um sistema é tolerante a falhas se ele pode mascarar a presença de defeitos no sistema utilizando mecanismos de redundância e monitoramento em nível de *hardware* e/ou *software*.

Não obstante, para garantir que exista um plano de ação realmente eficiente, são necessários técnicos com conhecimentos avançados sobre o assunto, pois no mercado de tecnologia existem diversos produtos e serviços, com arquiteturas e custos diferentes

que têm o mesmo objetivo de auxiliar na criação de um ambiente disponível. Logo, é fundamental que o profissional consiga criar uma solução preparada para se manter *online* após falhas de diversos tipos, como: mau funcionamento de algum componente, erros de manutenção e operação do sistema, desastres naturais, má elaboração no projeto, dentre tantos outros problemas que podem vir a acontecer.

Sendo assim, o monitoramento de *softwares* e *hardwares* se tornou imprescindível para manter o sistema computacional altamente disponível, sem qualquer *downtime* nos serviços críticos e de negócios da empresa ou instituição. Logo, as ferramentas de monitoramento tem por objetivo realizar avaliações de itens computacionais de maneiras automáticas, com o intuito de prevenir a indisponibilidade de serviços, possibilitando em alguns casos não apenas alertar sobre algum problema que veio a surgir, mas também baseado no histórico do item monitorado, alertar para um possível desastre futuro.

Além do mais, após a incrementação de um sistema desse tipo, a ferramenta possibilita ações proativas de serviços, um exemplo seria: uma empresa contrata uma operadora de internet para garantir que fique sem internet o mínimo de tempo possível. Com o ambiente de TI devidamente monitorando, é viável realizar a verificação mensal do status do link, ou seja, no final do mês a empresa pode conferir através de um relatório quanto tempo o link de internet da operadora se manteve disponível e se o preço cobrado é justo pelo serviço entregue.

Não obstante, além deste cenário citado acima, o monitoramento é importante para realização de investimentos corretos no ambiente de TI, pois a equipe consegue verificar tendências dentro do ambiente computacional. Obtendo dados suficientes para projetar uma demanda de recursos alguns meses à frente, garantindo que o orçamento seja devidamente aplicado no que realmente é necessário no ambiente.

Ademais, possuindo uma ferramenta de monitoramento em um ambiente de TI, é possível realizar o controle de *Services Level Agreement* (SLA), traduzido do inglês trata-se de acordos de níveis de serviços, isto é, em termos mensuráveis, são todos os serviços que precisam apresentar uma visão de alto nível da infraestrutura, com o objetivo de exibir detalhes de um serviço provido por um setor de TI. Logo, este mecanismo permite as empresas e instituições verificarem quanto tempo algum serviço deveria ficar à disposição e quanto tempo este mesmo serviço realmente ficou disponível.

2.3.1 Arquitetura de gerenciamento de redes e SNMP

Dessa forma, como foi dissertado previamente, é de suma importância realizar gerenciamento de equipamentos e elementos de TI, com o propósito de diminuir prejuízos econômicos. Sendo assim, para manter uma rede funcionando de maneira satisfatória, é

necessário possuir o conhecimento sobre a divisão das atividades de gerenciamento de redes e suas categorias. Passando por um planejamento sobre quais *softwares* e *hardwares* serão monitorados e a forma como estes dispositivos irão se comunicar com a entidade responsável por receber e processar esses dados.

Logo, o gerenciamento de redes pode ser dividido em duas categorias de atividades:

- **Monitoramento:** é uma função destinada à observação e análise do estado e comportamento dos dispositivos gerenciados;
- **Controle:** é uma função destinada a alteração de parâmetros de gerenciamento que acarretam ações junto aos dispositivos gerenciados.

Sendo assim, um ecossistema computacional deve ser composto pelas entidades que se deseja monitorar (agente) e por uma entidade responsável de adquirir as informações (gerente), para que dessa forma, seja centralizada a fonte de informações.

Assim sendo, de acordo com a ISO 7498-4 (TELECO, 2012), a gerência de redes pode ser classificada em cinco áreas funcionais: gerência de falhas, gerência de contabilização, gerência de configuração, gerência de desempenho e gerência de segurança. Embora esta classificação, geralmente referenciada como FCAPS (*Fault, Configuration, Accounting, Performance and Security*), tenha sido desenvolvida para o modelo OSI, houve uma grande aceitação desta por parte dos fabricantes de *hardware* e *software* de rede, tanto em tecnologias padronizadas como em proprietárias. Nas próximas seções serão abordados individualmente todos os itens acima citados.

2.3.1.1 Gerenciamento de Configuração

Para que uma rede funcione de maneira adequada, esta deve estar configurada. Esta configuração consiste em descobrir a rede, os dispositivos que fazem parte dela, manutenção e o que deverá ser monitorado tanto logicamente quanto fisicamente (CASTRO; CARVALHO; MENDES, 2013). Esta categoria sugere que haja uma sincronização entre o sistema monitorado e a rede, para que não seja necessário ficar constantemente verificando a rede para saber se é necessário realizar alguma configuração.

Logo, como foi descrito pelo grupo TELECO (2012), o gerente da rede deve ser capaz de identificar os componentes da rede e definir a conectividade entre eles, além de modificar a configuração em resposta às avaliações de desempenho, recuperação de falhas, problemas de segurança, atualização da rede ou para atender às necessidades dos usuários.

2.3.1.2 Gerenciamento de Falhas

O objetivo do gerenciamento de falhas é registrar, detectar e reagir às condições de falha da rede (KUROSE; ROSS, 2010), e se divide em três fases: monitorar, diagnosticar e tratar a falha. Quando ocorre uma falha, é importante que o administrador da rede possa reagir rapidamente analisando os seguintes requisitos (CASTRO; CARVALHO; MENDES, 2013):

- Detectar onde está o problema;
- Isolar a falha do resto da rede para que o problema não afete sua totalidade;
- Reajustar as configurações para reduzir impactos;
- Reparar ou trocar componentes que geraram a falha.

Portanto, é importante a compreensão do conceito de falha, porque falha não é a mesma coisa que erro. Uma falha é uma condição anormal que exige uma ação de gerenciamento, enquanto que um erro é um evento único. Em geral, uma falha é ocasionada pela incapacidade de operar normalmente ou pela excessiva quantidade de erros (STALLINGS, 2005).

2.3.1.3 Gerenciamento de Contabilização

Segundo o grupo TELECO (2012), mesmo que nenhuma cobrança interna seja feita pela utilização dos recursos da rede, o administrador da rede deve estar habilitado para controlar o uso dos recursos por usuário ou grupo de usuários, com o objetivo de:

- Evitar que um usuário ou grupo abuse de seus privilégios de acesso e monopolize a rede, em detrimento de outros usuários;
- Evitar que usuários façam uso ineficiente da rede, assistindo-os na troca de procedimentos e garantindo a desempenho da rede;
- Conhecer as atividades dos usuários com detalhes suficientes para planejar o crescimento da rede;

Desta forma, o gerenciamento de contabilização também pode em algumas situações, dar suporte a auditorias e comunicações de fraudes a partir de análises suspeitas ou pouco comuns.

2.3.1.4 Gerenciamento de Desempenho

De acordo com Kurose e Ross (2010), a meta do gerenciamento de desempenho é quantificar, medir, informar, analisar e controlar o desempenho (por exemplo, utilização e vazão) de diferentes componentes da rede. Para este controle, métricas e valores apropriados devem, ser adotados aos recursos de rede para que possam fornecer parâmetros indicadores de diferentes níveis de desempenho. Estatísticas de desempenho podem ajudar no planejamento, administração e manutenção das redes, onde através de análises é possível descobrir gargalos na rede e ações preventivas possam ser tomadas com a finalidade de garantir o bom desempenho da rede (CASTRO; CARVALHO; MENDES, 2013).

De acordo com KOCH e WESTPHALL (2001) em algumas questões que envolvem desempenho, o administrador da rede deve se preocupar com:

- Qual a capacidade da rede quando se fala em desempenho?
- O tráfego atual é excessivo?
- A vazão tem diminuído para níveis inaceitáveis?
- Existe algum gargalo?

Mediante a análise e possuindo critérios de métricas para comparação, o administrador da rede pode detectar variações no comportamento da rede e tomar providências caso seja necessário.

2.3.1.5 Gerenciamento de Segurança

O gerenciamento de segurança visa prover facilidades para proteger os recursos e as informações da rede e dos usuários. Tem como fundamento garantir que somente pessoas autorizadas possam ter acesso aos recursos disponíveis. Deve monitorar e controlar acessos a computadores da rede, coletar e examinar logs de acordo com alguma política definida (CASTRO; CARVALHO; MENDES, 2013).

2.3.2 Modelo de gerenciamento

Assim sendo, o modelo de gerenciamento de redes pode ser entendido como o processo de controle de uma rede computacional de tal modo que seja possível maximizar sua eficiência e produtividade, compreendendo um conjunto de funções integradas que podem estar em uma máquina ou espalhados por milhares de quilômetros, em diferentes organizações e residindo em máquinas distintas. Logo, é importante observar que com

estas funções, pode-se controlar uma rede de computadores e seus serviços, provendo mecanismos de monitoração, análise e controle dos dispositivos e recursos da rede.

Portanto, um sistema de gerenciamento nada mais é que uma coleção de ferramentas integradas, possuindo uma interface em que o usuário pode visualizar e realizar comandos de controle. De forma simplificada, este ambiente é formado por dois itens: gerente e agente.

2.3.2.1 Gerente

Também conhecido como NMS (*Network Management Stations*), o gerente consiste em um *software* responsável por recuperar dados dos agentes de monitoramento. Com a finalidade de monitorar e controlar vários dispositivos na rede, através de protocolos das camadas de transporte, de rede e de aplicação, para a comunicação com a entidade de rede gerenciada.

Sendo assim, as informações de gerenciamento podem ser obtidas com o uso de requisições efetuadas pelo gerente ao agente, como também, mediante envio automático disparado pelo agente a um determinado gerente. Tipicamente um gerente está presente em uma estação de gerenciamento de rede, podendo em alguns casos estar alocado a algum serviço em nuvem. Não obstante, a aplicação gerente é considerada o coração do sistema de gerenciamento de rede e como tal, deve-se fornecer atenção redobrada ao mesmo.

2.3.2.2 Agente

O agente por sua vez consiste na materialização da entidade gerenciada, compreendendo um tipo de *software* presente junto aos dispositivos gerenciados. A função principal de um agente é gerar requisições enviadas por um *software* gerente e o envio automático de informações de gerenciamento, indicando a ocorrência de um evento previamente programado. O agente também é responsável por efetuar a criação da interface entre os diferentes mecanismos usados na instrumentação das funcionalidades de gerenciamento inseridas em um determinado dispositivo gerenciado. Para os agentes, não existe passado nem futuro, pois as suas ações são baseadas nas informações colhidas pelo sensor naquele instante e atua no meio através das regras contidas na sua base de conhecimentos. Por isso, quando cessa a percepção do ambiente cessa a ação.

2.3.2.3 Protocolo SNMP

Com o objetivo de maximizar a quantidade de produtos e serviços monitorados de diversos fabricantes, utilizam-se o protocolo SNMP (*Simple Network Management Protocol*), em português Protocolo Simples de Gerência de Rede. É um protocolo padrão

da Internet para gerenciamento de dispositivos em redes IP. Na prática, o SNMP é mais usado para saber o que acontece dentro de ativos de redes e serviços, como: roteadores, computadores, servidores, estações de trabalho, impressoras, racks modernos, entre outros dispositivos.

Além do mais, o protocolo foi criado para facilitar o monitoramento e gerenciamento de redes, permitindo que uma ferramenta de gerenciamento possa trabalhar com produtos e serviços de diversos fabricantes. A Figura 10, apresenta através de camadas, como o protocolo SNMP realiza a comunicação entre dispositivos.

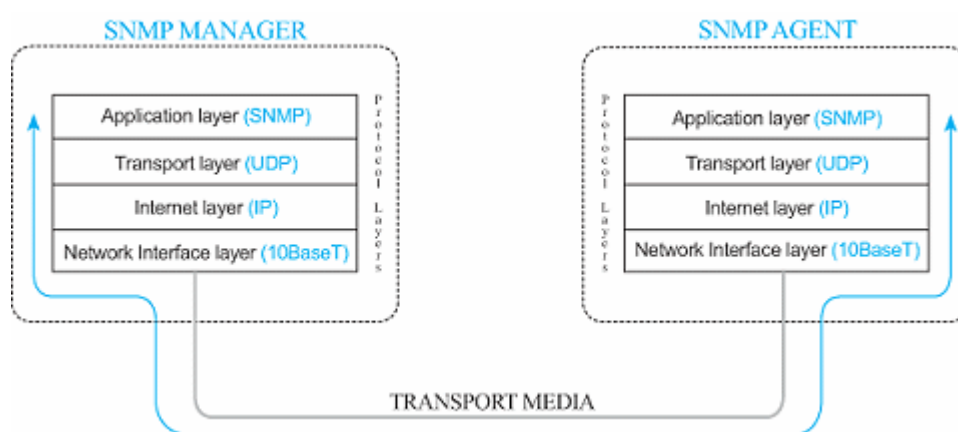


Figura 10 – Comunicação em camadas do protocolo SNMP. Fonte: DPSTelecom (2001).

Todavia, as portas de comunicação utilizadas para o transporte de dados são:

- UDP 161 para consultar os agentes;
- UDP 162 para traps e mensagens automáticas, são enviadas aos gerentes quando os agentes se comportam de maneira anormal.

Além do mais, é importante ressaltar que o protocolo SNMP não utiliza o controle de transmissão (TCP), um serviço de rede que tem por objetivo estabelecer uma conexão segura com dispositivos de rede antes de começar a realizar a transmissão de dados, ou seja, o protocolo de gerência utiliza o *User Datagram Protocol* (UDP), serviço que não estabelece comunicação com dispositivos de rede antes de transmitir os dados. Isso gera um sistema que envia dados de um modo mais veloz, pois exige menos recursos do emissor e do receptor, todavia, o UDP pode não ser confiável e a entrega da mensagem não é garantida.

2.3.2.4 MIB

Ademais, para que o protocolo SNMP seja eficiente quando utilizado em um sistema de gerenciamento de rede é necessário que este, utilize as MIBs que de acordo com

(MAURO; SCHMIDT, 2001) a MIB (*Management Information Base*) pode ser considerada um banco de dados de objetos gerenciados que o agente rastreia.

Logo, grande parte das informações coletadas relacionadas a status e estatística é definida pela MIB. Portanto, a SMI, estrutura de informações de gerenciamento, é um método para definir objetos gerenciados e os seus respectivos comportamentos, já a MIB, pode ser considerada de forma mais grosseira, como um banco de dados de objetos gerenciados que contém informações organizadas hierarquicamente, que por sua vez, são consultados/manipulados pelos agentes SNMP.

Dessa maneira, a MIB permite coletar dados estatísticos de interface, octetos recebidos e enviados. A ferramenta tem como principal objetivo informar sobre o gerenciamento TCP/IP e não engloba todos os equipamentos gerenciáveis disponíveis em um dispositivo computacional. A Figura 11 exibe a subárvore que define 10 grupos utilizados no gerenciamento definidos pela OID 1.3.6.1.2.

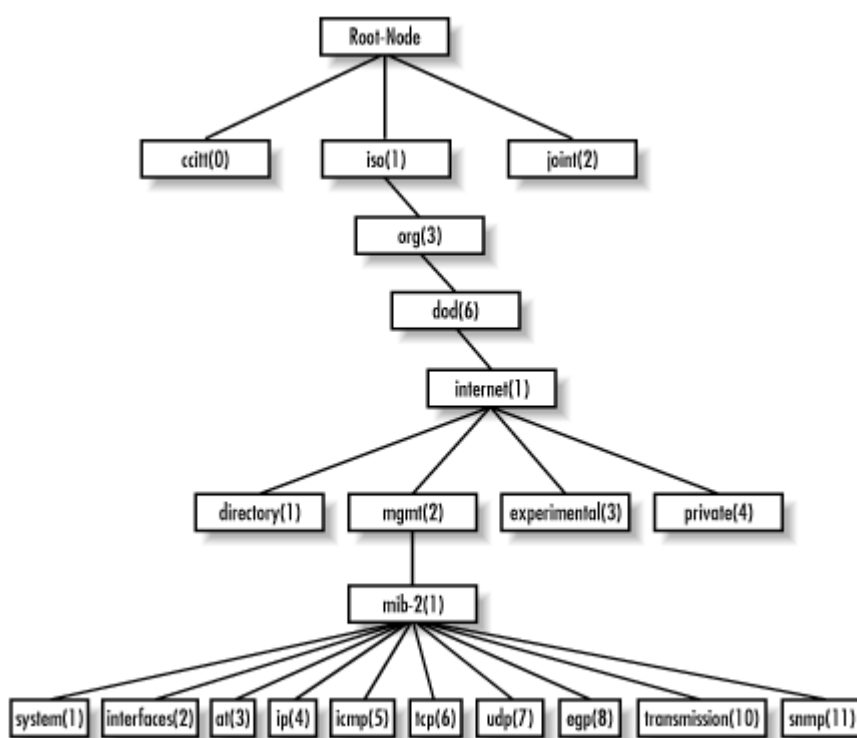


Figura 11 – Subárvore da OID 1.3.6.1.2. Fonte: Cavalheiro (2020).

2.3.2.5 Operações utilizando as variáveis MIB

Resumidamente o número de comandos se remete em dois, *GET* e *SET*, sendo que a partir deles surgiram outros comandos como (*GET-NEXT* e *TRAP*). Abaixo segue uma descrição resumida dos comandos, segundo (CASTRO; CARVALHO; MENDES, 2013):

- **Operação SET:** é quando um gerente solicita que o agente faça uma alteração no valor da variável;
- **Operação GET:** é quando gerente solicita que o agente obtenha valor da variável;
- **Operação GET-NEXT:** é quando o gerente fornece o nome de uma variável e o cliente obtém o valor e o nome da próxima variável. Também é utilizado para obter valores e nomes de variáveis de uma tabela de tamanho desconhecido;
- **Operação TRAP:** é utilizada para comunicar um evento; o agente comunica ao gerente o acontecimento de um evento, previamente determinado. São sete tipos básicos de trap determinados:
 - **coldStart:** a entidade que a envia foi reinicializada, indicando que a configuração do agente ou a implementação pode ter sido alterada;
 - **warmStart:** a entidade que a envia foi reinicializada, porém a configuração do agente e a implementação não foram alteradas;
 - **linkDown:** o enlace de comunicação foi interrompido;
 - **linkUp:** o enlace de comunicação foi estabelecido;
 - **authenticationFailure:** o agente recebeu uma mensagem SNMP do gerente que não foi autenticada;
 - **egpNeighborLoss:** um par EGP parou;
 - **egpNeighborLoss:** indica a ocorrência de uma operação TRAP não básica.

Para conseguir efetuar o monitoramento através das MIBS, é necessário entender os tipos de objetos e como cada item se comporta, logo, a Figura 12 expõe em forma de tabela, os nomes dos grupos das subárvores, suas OIDS e a descrição de cada item.

Além disso, alguns exemplos dos objetos pertencentes aos grupos da MIB 2 são:

- **Grupo System (1.3.6.1.2.1.1)**
 - **sysDescr (1.3.6.1.2.1.1.1):** descrição textual da unidade. Pode incluir o nome e a versão do *hardware*, sistema operacional e o programa de rede;
 - **sysUpTime (1.3.6.1.2.1.1.3):** tempo decorrido (em milhares de segundos) desde a última reinicialização do gerenciamento do sistema na rede;
 - **sysContact (1.3.6.1.2.1.1.4):** texto de identificação do gerente da máquina gerenciada e como contatá-lo.

- **Grupo Interfaces (1.3.6.1.2.1.2)**

- **ifNumber (1.3.6.1.2.1.2.1):** número de interfaces de rede (não importando seu atual estado) presentes neste sistema;
- **ifOperStatus (1.3.6.1.2.1.2.2.1.8):** estado atual da interface;
- **ifInOctets (1.3.6.1.2.1.2.2.1.10):** número total de octetos recebidos pela interface.

- **Grupo IP (1.3.6.1.2.1.4)**

- **ipForwarding (1.3.6.1.2.1.4.1):** indica se esta entidade é um gateway;
- **ipInReceives (1.3.6.1.2.1.4.3):** número total de datagramas recebidos pelas interfaces, incluindo os recebidos com erro;
- **ipInHdrErrors (1.3.6.1.2.1.4.4):** número de datagramas que foram recebidos e descartados devido a erros no cabeçalho IP.

- **Grupo ICMP (1.3.6.1.2.1.5)**

- **icmpInMsgs (1.3.6.1.2.1.5.1):** número total de mensagens ICMP recebidas por esta entidade, incluindo aquelas com erros;
- **ipInReceives (1.3.6.1.2.1.4.3):** número total de datagramas recebidos pelas interfaces, incluindo os recebidos com erro;

Nome de Sub-árvore	OID	Descrição
<i>system</i>	1.3.6.1.2.1.1	Define uma lista de objetos que pertencem ao sistema operacional, como “uptime” do sistema, contato do sistema, nome do sistema.
<i>interfaces</i>	1.3.6.1.2.1.2	Mantém históricos de status de cada interface da entidade gerenciada, como status de <i>up</i> ou <i>down</i> , octetos enviados e recebidos, erros, descartes, etc.
<i>at</i>	1.3.6.1.2.1.3	O grupo <i>at</i> (<i>address translation</i>) está obsoleto e existe para manter compatibilidade.
<i>ip</i>	1.3.6.1.2.1.4	Mantém históricos de diversos aspectos referentes do protocolo IP, inclusive roteamento.
<i>icmp</i>	1.3.6.1.2.1.5	Trata aspectos como pacotes de erro <i>ICMP</i> , descartes, etc.
<i>tcp</i>	1.3.6.1.2.1.6	Trata entre outras coisas dos estados do <i>TCP</i> : <i>closed</i> , <i>listen</i> , <i>synSent</i> , etc.
<i>udp</i>	1.3.6.1.2.1.7	Trata de estatísticas do <i>UDP</i> , datagramas recebidos e enviados, etc.
<i>egp</i>	1.3.6.1.2.1.8	Trata de várias estatísticas sobre <i>EGP</i> e mantém um tabela de vizinhança <i>EGP</i> .
<i>transmission</i>	1.3.6.1.2.1.10	Atualmente não há nenhuma definição de objeto neste grupo, mas outras especificações de mídia são definidas usando esta subárvore.
<i>snmp</i>	1.3.6.1.2.1.11	Medição da performance <i>SNMP</i> na entidade gerenciada, tratando coisas como número de pacotes <i>SNMP</i> recebidos e enviados.

Figura 12 – Tabela de descrições de objetos da MIB. Fonte: Castro, Carvalho e Mendes (2013).

- **icmpOutMsgs (1.3.6.1.2.1.5.14)**: número total de mensagens ICMP enviadas por esta entidade, incluindo aquelas com erros.
- **Grupo TCP (1.3.6.1.2.1.6)**
 - **tcpMaxConn(1.3.6.2.1.6.4)**: número máximo de conexões TCP que esta entidade pode suportar;
 - **tcpCurrentEstab (1.3.6.2.1.6.9)**: número de conexões TCP que estão como estabelecidas ou a espera de fechamento;
 - **tcpRetransSegs (1.3.6.2.1.6.12)**: número total de segmentos retransmitidos.
- **Grupo UDP (1.3.6.1.2.1.7)**
 - **udpInDatagrams (1.3.6.1.2.1.7.1)**: número total de datagramas UDP entregues aos usuários UDP;
 - **udpNoPorts (1.3.6.1.2.1.7.2)**: número total de datagramas UDP recebidos para os quais não existiam aplicações na referida porta;
 - **udpLocalPort (1.3.6.1.2.1.7.5.1.2)**: número da porta do usuário UDP local.
- **Grupo SNMP (1.3.6.1.2.1.11)**
 - **snmpInPkts (1.3.6.1.2.1.11.1)**: número total de mensagens recebidas pela entidade SNMP;
 - **snmpOutPkts (1.3.6.1.2.1.11.2)**: número total de mensagens enviadas pela entidade SNMP;
 - **snmpInTotalReqVars (1.3.6.1.2.1.11.13)**: número total de objetos da MIB que foram resgatados pela entidade SNMP.

Contudo, no cenário de monitoramento, existe também o protocolo RMON (Figura 13). Dentre os protocolos de gerenciamento, o RMON é certamente, um dos primeiros a permitir o gerenciamento proativo. Talvez seja esta a grande vantagem do mesmo em relação às outras arquiteturas de gerenciamento. O trabalho de gerenciamento é simplificado e a resolução dos problemas facilitada. Assim, aumenta a disponibilidade da rede e caem os custos de manutenção de forma significativa (LESSA, 1999).

Portanto, com a evolução dos ambientes de rede e o aumento de sua complexidade, ficou cada vez mais difícil monitorar e gerenciar equipamentos de tecnologia. Para resolver esse problema, vários pesquisadores buscaram criar soluções que pudessem auxiliar no processo de gerenciamento de grandes ambientes de rede. Dentre essas soluções destacou-se o SNMP, que possui 3 diferentes versões descritas abaixo:

- **SNMPv1:** é a primeira versão do SNMP que possui um esquema de autenticação extremamente frágil, sendo seu único mecanismo de segurança os "nomes de comunidade". Em que uma comunidade representa um grupo de gerenciamento com permissões específicas, ou seja, atribuição dos direitos de utilização das instruções *SET* e *GET* sobre um certo parâmetro a membros desta comunidade. Não obstante, a principal falha deste modelo de segurança reside no fato de que qualquer um que conheça o nome de comunidade com os privilégios adequados pode enviar um comando do SNMP pela rede.
- **SNMPv2:** a segunda versão do SNMP tinha como principal objetivo reformular o modelo de segurança do SNMPv1, todavia muitos usuários vieram a considerar essa nova versão muito complexa, pois existia um conjunto de versões incompatíveis e não existia um padrão prévio a ser seguido. Além disso, o segundo modelo não teve êxito ao tentar melhorar o modelo de segurança e esta versão continuou seguindo o sistema antigo. Ademais, o SNMPv2 conseguiu integrar algumas novas funcionalidades como o *Getbulkrequest*, instrução responsável por retornar uma grande quantidade de informações de gerenciamento de uma só vez, eliminando uma das maiores limitações do SNMP, que era a sua incapacidade de recuperar grandes blocos de dados.

Nome de Sub-árvore	OID	Descrição
<i>rmon</i>	1.3.6.1.2.1.16	Fornecer os dados estáticos no nível de pacotes sobre uma LAN ou WAN.
<i>statistics</i>	1.3.6.1.2.1.16.1	Contém dados estatísticos sobre todas as interfaces Ethernet monitorados pela prova.
<i>history</i>	1.3.6.1.2.1.16.2	Registra amostras periódicas de dados estatísticos do grupo <i>statistics</i>
<i>alarm</i>	1.3.6.1.2.1.16.3	Permite que um usuário configure um intervalo de <i>polling</i> e um limiar para qualquer objeto registrado pela prova do RMON.
<i>hosts</i>	1.3.6.1.2.1.16.4	Registra dados estatísticos sobre o tráfego de cada host na rede.
<i>hostTopN</i>	1.3.6.1.2.1.16.5	Contém dados estatísticos do host utilizado para gerenciar relatórios sobre os hosts que encabeçam uma lista ordenada por um parâmetro existente na tabela de hosts.
<i>matrix</i>	1.3.6.1.2.1.16.6	Armazena informações de erro e utilização para os conjuntos de dois endereços
<i>filter</i>	1.3.6.1.2.1.16.7	Faz a correspondência de pacotes com base em uma equação de filtro; quando um pacote encontra um filtro correspondente, esse filtro pode ser obtido ou um evento pode ser gerado
<i>capture</i>	1.3.6.1.2.1.16.8	Permite a captação dos pacotes se corresponde a um filtro do grupo de filtros.
<i>event</i>	1.3.6.1.2.1.16.9	Controla a definição dos eventos RMON.

Figura 13 – Grupos de RMON MIB. Fonte: Castro, Carvalho e Mendes (2013).

- **SNMPv3:** esta versão do protocolo teve como principal foco a melhoria da segurança oferecida pelas versões anteriores do protocolo SNMP. Foram desenvolvidos mecanismos para lidar com cada uma das falhas de segurança discutidas até então. Desta forma, se tornou possível utilizar o potencial completo do protocolo, incluindo as instruções *SET*, sem comprometer a segurança da rede. O novo modelo de segurança garante confidencialidade, integridade, autenticação e controle de acesso.

2.3.3 Ferramentas de monitoramento

As ferramentas de monitoramento possibilitam o acompanhamento prévio de empresas e instituições em relação ao ambiente de tecnologia. Contudo, existem diversas opções de *softwares* disponíveis no mercado, cada uma com sua característica, instrumentos apropriados para monitorar servidores, equipamentos de rede e aplicações. Há, também, soluções que rastreiam desempenho de sistemas e dispositivos, oferecendo tendências e análises. Algumas dessas tecnologias disparam alarmes e notificações quando detectam problemas, enquanto outras inclusive já começam a trabalhar antes mesmo das coisas ficarem efetivamente críticas.

Dessa maneira, esse subcapítulo, tem por objetivo apresentar diversas soluções de monitoramento que foram estudadas e expor os motivos pelos quais a ferramenta Zabbix foi a escolhida. Além de evidenciar como é genericamente a arquitetura de um *software* de monitoramento.

2.3.4 Arquitetura de um *software* de monitoramento

Um *software* de gerenciamento de acordo com o grupo TELECO (2012) é composto basicamente por elementos gerenciados, agentes, gerentes, banco de dados, protocolo para troca de informações de gerenciamento, interfaces para programas aplicativos e interfaces com o usuário.

Podendo ser dividido três grandes categorias:

- *Software* de apresentação para o usuário (interface);
- *Software* para gerenciamento da rede (aplicação);
- *Software* de suporte (banco de dados e comunicação).

2.3.4.1 *Software* de apresentação para o usuário (interface)

Com o objetivo de integrar o administrador de rede aos recursos de rede é necessário que exista uma interface gráfica, para que dessa forma o usuário possa gerenciar

todo o ambiente monitorado através de uma plataforma unificada que contenha diversos ambientes heterogêneos, gerando desta forma, um sistema simples para visualização de dados.

2.3.4.2 Software para gerenciamento da rede (aplicação)

É nesse nível que situa-se o *software* para a implementação de protocolos padrão como SNMP. Normalmente está dividido em três níveis: no primeiro nível, têm-se as aplicações de gerenciamento de rede que fornecem as funcionalidades necessárias para o gerenciamento das áreas funcionais: desempenho, falhas, configuração, segurança e contabilização; no segundo nível, estão as aplicações elementares que tratam das rotinas mais triviais como providenciar alarmes ou resumir dados; no terceiro nível, ficam os serviços de transporte de dados do protocolo que fornecem comandos básicos como a busca de informações, o estabelecimento de parâmetros e outros.

2.3.4.3 Software de suporte (banco de dados e comunicação)

Desse modo, um banco de dados tem por objetivo armazenar, organizar, proteger, atualizar, acrescentar, acessar e excluir dados sempre que necessário, devendo corresponder à demanda que a aplicação que o utiliza exige. Já a comunicação é feita através de diversos protocolos de rede, presentes no modelo *TCP/IP* e *OSI*.

2.3.5 Cacti

Como foi apresentado neste documento é necessário receber constantemente informações sobre a situação da infraestrutura, para que medidas possam ser tomadas, visando melhorar o desempenho dos recursos computacionais.

Partindo dessa premissa, o projeto Cacti (Figura 14) foi iniciado por Ian Berry em setembro de 2001. Berry se inspirou para iniciar o projeto enquanto trabalhava para um pequeno provedor de serviços de internet. Seu objetivo central ao criar Cacti "era oferecer mais facilidade de uso do que o RRDtool e mais flexibilidade do que o MRTG ", sendo estes, dois *softwares* de monitoramento e gerência de dados.

Dessa maneira, o projeto foi feito utilizando a linguagem de programação PHP para desenvolver o *software* e usou também o sistema de gerenciamento de banco de dados MySQL.

Ademais, o Cacti utiliza a GNU (*General Public License*), uma licença desenvolvida por Richard Matthew Stallman em 1989, e é historicamente utilizada por projetos de software e de código aberto.

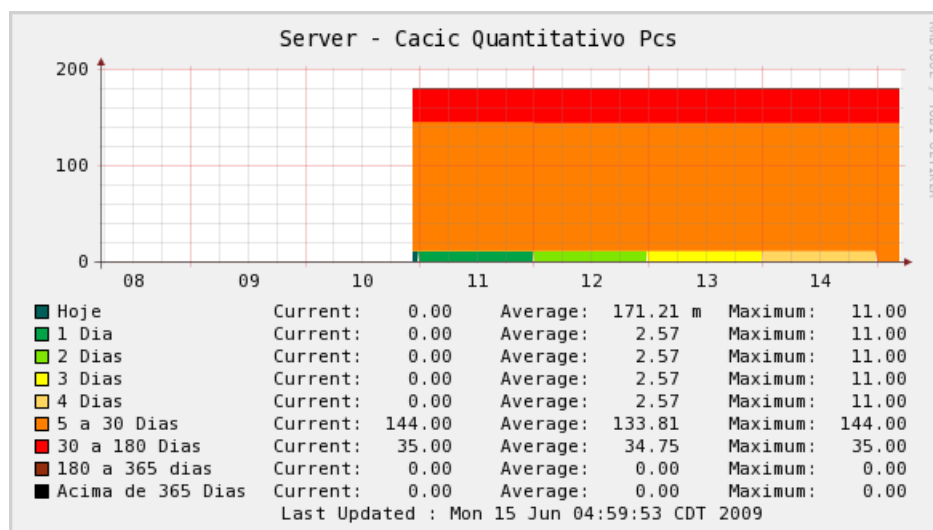


Figura 14 – Exemplo gráfico de monitoramento no Cacti . Fonte: Próprio Autor.

Outrossim, para que a ferramenta possa operar, é necessário que o SNMP esteja instalado no servidor de monitoramento e também nos equipamentos que serão monitorados. Logo, o *software* trabalha diretamente com SNMP e com qualquer informação que este possa recuperar, pode também ser reproduzida no seu *front end*.

Para mais, a plataforma possui uma limitação em relação a sua *database* já que sua base de dados gerada possui um tamanho máximo o qual uma vez atingido não é ultrapassado. Os dados numéricos armazenados são consolidados conforme a configuração fornecida, de modo que a resolução deles seja reduzida de acordo com o tempo que eles estão armazenados. Neste processo, apenas as médias dos valores antigos são armazenados.

Todavia, o Cacti é uma ferramenta que recolhe informações utilizando SNMP e exibe informações gráficas sobre o estado da rede, logo, a tecnologia foi desenvolvida com o intuito de ser flexível e se adaptar facilmente a diversas necessidades, bem como sua arquitetura prevê a possibilidade de expansão através de plugins que adicionam novas funcionalidades. Além disso, a plataforma utiliza o RRDtool, ferramenta de banco de dados round-robin, que visa lidar com dados de série temporal, como largura de banda da rede, temperaturas ou carga da CPU.

Suas vantagens são:

- Facilmente instalável não necessitando de conhecimentos avançados;
- Ele reage às condições e envia notificações se as condições não estiverem dentro de um intervalo especificado;
- Ajuda a aproveitar ao máximo o rrdgraph para automatizar sua exibição;

- Sistema de árvores, possibilitando diversas visões detalhadas.

As principais desvantagens são:

- A configuração é um processo elaborado e é difícil corrigir os erros de configuração;
- Não realiza o resumo de múltiplos recursos;
- A configuração do Cacti deve estar sempre atualizada. É melhor usar scripts para esse propósito;
- Necessidade de download de plugins de terceiros;
- O Cacti não envia alerta em caso de paradas de serviço ou algum outro evento anormal.

2.3.6 The Dude

The Dude é um aplicativo gratuito da MikroTik de gerenciamento de rede. Algumas de suas funções é escanear automaticamente todos os dispositivos dentro de sub-redes especificadas, desenhar e fazer o layout de um mapa de suas redes, monitorar serviços de seus dispositivos e executar ações com base nas mudanças de estado do dispositivo.

A ferramenta foi desenvolvida por uma empresa da Letônia chamada MikroTik. Esta organização fábrica equipamentos para redes de computadores além de vender produtos *wireless* e roteadores. Ademais, a companhia foi fundada em 1995 com a intenção de possuir como o principal consumidor final países emergentes. Seus equipamentos ainda são consumidos no mercado global por sua conhecida estabilidade e versatilidade.

Não obstante, apesar da empresa possuir o The Dude como um das principais ferramentas de gerenciamento de rede do mercado, o principal produto da empresa é um SO baseado em Linux chamado de MikroTik RouterOS que tem por objetivo permite que qualquer plataforma x86 torne-se um poderoso roteador, com funções como VPNs, *Proxy*, *Hotspots*, Controle de Banda, QoS, *Firewall*, dentre outras, que variam de acordo com o nível de licença do sistema adquirido.

Portanto, como dispositivo de gerência de rede este, *software* possui algumas características importantes, como: fornece informações acerca de quedas e restabelecimentos de redes, serviços, assim como uso de recursos de equipamentos; permite o mapeamento da rede com gráficos da topologia da rede e relacionamentos lógicos entre os dispositivos; notifica via áudio/vídeo/e-mail acerca de eventos; possui gráfico de serviços mostrando latência, tempos de resposta de DNS, utilização de banda, informações físicas de *links*, além de monitorar dispositivos não RouterOS com SNMP.

Todavia, apesar da ferramenta ser de fácil instalação e configuração sua instalação se limita a plataforma Windows e a plataformas Mikrotik, tornando seu uso restrito já que o aplicativo ao ser instalado em ambiente Linux perde muito desempenho. Com isso, o instrumento de gerenciamento apresentou várias desvantagens como: documentação e fóruns com poucas informações detalhadas; interface web com funcionalidades básicas e código fonte não disponibilizado, que impede customizações conforme a real necessidade do administrador de rede, além de destinar ao usuário um ambiente limitado de recursos para gerenciamento; toda arquitetura é mais voltada para gerenciamento dos equipamentos da própria empresa.

2.3.7 Nagios

Nagios é uma popular aplicação de monitoramento de rede de código aberto distribuída sob a licença GPL, já citada neste trabalho. Esse dispositivo tem como principais objetivos possibilitar o monitoramento tanto de hosts quanto serviços, alertando quando ocorrerem problemas e também quando os problemas são resolvidos. Esta tecnologia foi originalmente criado sob o nome de Netsaint.

Todavia, em 2002, devido a mudanças de estratégia da empresa, o nome foi alterado para NAGIOS, que é na verdade um acrônimo recursivo para "Nagios Ain't Gonna Insist On Sainthood"(Nagios não vai insistir em santidades), fazendo assim, uma referência para o antigo novo. Além do mais, a ferramenta foi escrita e é atualmente mantida por Ethan Galstad, junto com uma equipe de desenvolvedores que ativamente mantém plugins oficiais e não-oficiais.

Dentre os principais recursos estão:

- **Interface da Web personalizável:** inclusão de um painel visualizado em um navegador da Web, que pode ser usado para monitorar e gerenciar dispositivos de rede. As opções e configurações do Nagios também podem ser configuradas no navegador da web e os painéis podem ser personalizados para exibir uma visão geral para facilitar as atualizações de status;
- **Diversos recursos de monitoramento:** a ferramenta é capaz de realizar verificações ativas / passivas de host, verificações de serviço, coletar vários tipos de informações do sistema, monitorar aplicativos, protocolos de rede e muito mais. Complementos de terceiros estão disponíveis, aumentando ainda mais o escopo monitorável do Nagios XI;
- **Reporting Services:** Nagios XI oferece suporte a relatórios personalizados detalhando todos os tipos de eventos com base em filtros definidos pelo usuário. Algumas

das opções de filtragem incluem intervalos de tempo, hosts, grupos e muito mais. Os dados são exibidos em formato de gráfico e tabela e podem ser adicionados ao painel ou exportados para JPEG, se necessário, com dois cliques;

- **Recursos de notificação:** pode enviar alertas por e-mail ou SMS. As opções de filtragem estão disponíveis apenas para permitir determinados tipos de alertas. As opções de status listadas são confirmação, recuperação, inatividade, inacessível, oscilação ou tempo de inatividade para hosts. As mesmas opções estão disponíveis para o serviço, com a adição de avisos e status críticos, e a remoção de status inacessível. As próprias mensagens de notificação podem ser personalizadas, e o envio de notificações de teste também é suportado;
- **Suporte a LDAP e usuários personalizados:** o Nagios XI é capaz de se integrar ao LDAP e definir diferentes níveis de acesso e visualizações para diferentes usuários, permitindo o uso departamental em vez de apenas uma conta de administrador.
- **Monitoramento de largura de banda via SNMP:** o Nagios fornece uma análise aprofundada de sua largura de banda de rede pelo monitoramento via SNMP v1, 2c ou 3. Dê uma olhada em sua utilização de interface de rede e a quantidade de interfaces de rede de entrada e saída de tráfego. Identifique facilmente as portas trafegadas ou utilizadas, descubra os excessos de tráfego / largura de banda da rede e identifique rapidamente a latência ou interrupções.

Porém, pelo fato da ferramenta Nagios não ser nativa para Windows, existe a possibilidade de realizar o monitoramento desse tipo de SO, através da instalação de agentes na máquina alvo. Dessa maneira, a comunicação toda é feita entre servidor e agente. O agente fica na máquina a ser monitorada esperando requisições feitas pelo servidor, ao receber uma requisição, ele descobre a informação requisitada e retorna ao servidor. A coleta de dados também pode ser realizada utilizando o protocolo SNMP, todavia, nesse caso não é necessária a utilização de agente.

Ademais, o Nagios é uma ferramenta com grande capacidade de modificação, logo, qualquer comunicação realizada por plugins pode ser efetuada com sucesso. Sendo assim, a detentora do dispositivo, criou um plug in padrão para a plataforma o NRPE, já traduzido a sigla significa Executor de Plugins Remoto Nagios. Este *plugin* é instalado na máquina que está sendo monitorada e na máquina que está fazendo o monitoramento. Dessa maneira, as checagens do Nagios são feitas localmente, pois o NRPE se encarrega de realizar a comunicação com o servidor remoto.

Contudo, para conseguir ter acesso a plataforma e a todos os recursos é necessário desembolsar uma quantia em dinheiro, ou então utilizar a versão gratuita por 30 dias.

Abaixo estão os preços do Nagios XI, contendo duas edições distintas com três opções cada uma em termos de preço e licenciamento. Eles são como tais:

- **Standard Edition:** 100 Nós - \$ 1.995, 200 Nós - \$ 3.495, 300 Nós - \$ 4.995, 400 Nós - \$ 5.995, 500 Nós - \$ 6.995, 1.000 Nós - \$ 11.995, Node ilimitado - \$ 19.995. A edição Standard inclui configuração GUI, relatórios, visualizações, painéis/ visualizações personalizados, notificações e muito mais.
- **Enterprise Edition:** A atualização da *Enterprise Edition* custa US \$ 1.500 além do custo da versão Standard Edition. A edição *Enterprise* inclui todos os recursos listados acima, bem como ferramentas em massa, registro de auditoria, suporte e relatórios de SLA, descomissionamento automatizado de host, relatórios de planejamento de capacidade, acesso ao console do servidor, páginas agendadas, interface SNMP Trap e muitas outras opções.

Ademais, após todos os levantamentos realizados sobre a tecnologia, a seguir estão alguns pontos positivos e negativos sobre o Nagios. Pontos positivos:

- Manutenção preventiva;
- Monitora toda sua infraestrutura de rede;
- Aumento da produtividade dos administradores e analistas da rede;
- Integração com *plug-ins*;
- Age prontamente e executa ações de resolução de problemas.

Pontos negativos:

- Falta de recursos gráficos de configuração;
- Atraso de configuração do sistema;
- Indisponibilidade de ferramentas de configuração na interface web;
- Apresenta certa limitação para se adaptar à rápida evolução das redes atuais.

2.3.8 Zabbix

O Zabbix começou como um projeto de *software* interno em 1998, por Alexei Vladishev. Após três anos, em 2001, foi lançado ao público sob a GPL e três anos depois, foi publicada a primeira versão estável 1.0, em 2004. Ademais, grande parte das aplicações do *software* foram escritos em C, como o servidor, o *proxy* e os agentes. Além disso, o *frontend* foi desenvolvido em PHP, e a funcionalidade *gateway* Java foi desenvolvida em Java.

Assim sendo, o *software* é ferramenta de monitoramento de código aberto para diversos componentes de TI, incluindo redes, servidores, máquinas virtuais e serviços em nuvem. Além disso, o dispositivo fornece métricas de monitoramento, como utilização da rede, carga da CPU e consumo de espaço em disco.

Outrossim, o Zabbix utiliza o tipo servidor-agente de funcionamento, inclusive possibilitando que mais de um servidor esteja rodando ao mesmo tempo, o que possibilita uma melhor performance e maior consistência de dados.

Portanto, para criar um ambiente que proporcione uma melhor performance, o servidor se comunica com o agente (instalado em cada uma das máquinas que devem ser monitoradas) para requisitar informações sobre a máquina alvo. Sendo essas informações armazenadas em um banco de dados relacional, podendo ser ele: MySQL, PostgreSQL, Oracle, MariaDb.

Porém, se o host que necessita de monitoramento não possui a disponibilidade de instalação de um agente, é necessário utilizar os seguintes protocolos:

- **SNMP:** protocolo Simples de Gerência de Rede, descrito com mais detalhes no subcapítulo 2.3.2.3;
- **HTTP:** O *Hypertext Transfer Protocol*, traduzido significa Protocolo de Transferência de Hipertexto, é um protocolo de comunicação utilizado para sistemas de informação de hipermídia, distribuídos e colaborativos. Ele é a base para a comunicação de dados da internet;
- **IPMI:** *Intelligent Platform Management Interface*, traduzido significa Interface Inteligente de Gerenciamento de Plataforma consiste num conjunto de especificações padronizadas desenvolvidas para sistemas de gerenciamento de plataforma, baseados em *hardware*, permitindo gerenciar e monitorar servidores de maneira centralizada;
- **TCP:** Protocolo de Controle de Transmissão, é orientado à conexão e uma conexão entre o cliente e o servidor é estabelecida antes que os dados possam ser enviados;

- **Telnet:** é um protocolo de aplicativo usado na Internet ou rede local para fornecer um recurso de comunicação orientado a texto interativo bidirecional usando uma conexão de terminal virtual;
- **JMX:** *Java Management Extensions* (JMX), traduzido significa Extensões de gerenciamento Java é uma tecnologia Java que fornece ferramentas para gerenciar e monitorar aplicativos, objetos de sistema, dispositivos (como impressoras) e redes orientadas a serviços. Esses recursos são representados por objetos chamados *MBeans*;
- **SSH:** SSH ou *Secure Shell* é um protocolo de rede criptográfico para operar serviços de rede com segurança em uma rede não segura.

Ademais, através dos recursos descritos acima, consegue realizar: o monitoramento centenas de milhares de dispositivos, detectar automaticamente servidores e dispositivos e interfaces de rede, realizar o monitoramento distribuído com administração web centralizada, além de elaborar uma visão de alto nível dos recursos monitorados por meio de telas, painéis visuais do console definidos pelo usuário e Métricas de SLA.

A funcionalidade SLA, é um serviço interno do Zabbix, que possibilita o controle e gerenciamento de um compromisso assumido por um prestador de serviços de TI perante um cliente, ou seja, essa funcionalidade do Zabbix, descreve o serviço de TI, os níveis de qualidade que devem ser garantidos, as responsabilidades das partes e eventuais compensações quando os níveis de qualidade não forem atingidos. Logo, quanto maior o tempo de inatividade de algum produto, serviço ou funcionalidade da empresa, as perdas são mais significativas para uma empresa que se utiliza de uma infraestrutura de rede.

Além do mais, a arquitetura da plataforma possui vários módulos separados, sendo eles:

- *Zabbix Server*, não suportado em Windows, realiza o *polling* e *trapping* de dados, calcula *triggers* envia notificações aos usuários e precisa de um banco de dados para armazenar configurações e dados;
- Agentes Zabbix, instalados no sistema que deve ser monitorado para atingir valores como o uso de CPU / memória, que só podem ser acessados de dentro do sistema operacional;
- *Zabbix Frontend*, front-end da web usado para instalação, configuração e navegação de dados;
- *Zabbix Proxy*, não suportado no Windows, permite o acesso a sistemas que não podem ser acessados diretamente e reduz a carga no *Zabbix Server*.

Contudo, o Zabbix possui suas eventuais vantagens e desvantagens que serão descritas abaixo. Vantagens:

- Facilidade na criação de objetos;
- Relatórios gráficos altamente detalhados em diferentes formatos, recebendo todas as informações sobre o estado da rede e dos hosts que são criados em tempo hábil;
- Suporta um grande número de plataformas;
- Tem um sistema proativo para solucionar automaticamente os problemas. Um sistema de monitoramento, alertas e exibição de gráficos como nenhum outro sistema de monitoramento;
- É de código aberto e licenciado sob a GPL versão 2.

Desvantagens:

- Configuração inicial pode não ser trivial, e aprender os conceitos pode levar algum tempo, já que os manuais do usuário são muito grandes e complexos.

CAPÍTULO 3

MATERIAIS E MÉTODOS

Este capítulo aborda a metodologia, as ferramentas e os recursos computacionais a serem utilizados para realizar o monitoramento de um ambiente computacional por meio de contêineres Docker. Ademais, para realizar este capítulo, foi necessário preparar e definir o cenário de teste para o ambiente de execução do experimento, e as etapas pré estabelecidas para implantação dos serviços.

Além disso, foi fundamental para o trabalho abordar as comparações de uma série de conceitos para a escolha da ferramenta, com o objetivo de identificar pontos fortes e fracos de cada tecnologia.

3.1 Cenário de Teste

O cenário escolhido para coleta e comparação de dados é comum por apresentar uma estrutura e ativos de redes encontrados em muitas empresas. É constituído de:

- Um servidor Windows;
- Um notebook contendo Windows *Desktop*;
- Três computadores contendo Linux.

Tem-se ainda o servidor Zabbix responsável pela coleta de dados e monitoramento de estados. Neste mesmo servidor está instalado também o servidor *web* (*front end*), encarregado de disponibilizar páginas e todos os recursos que podem ser acessados via web pelo usuário, além do banco de dados, que tem por objetivo armazenar os dados coletados. Todo o conjunto está representado na topologia da Figura 15.

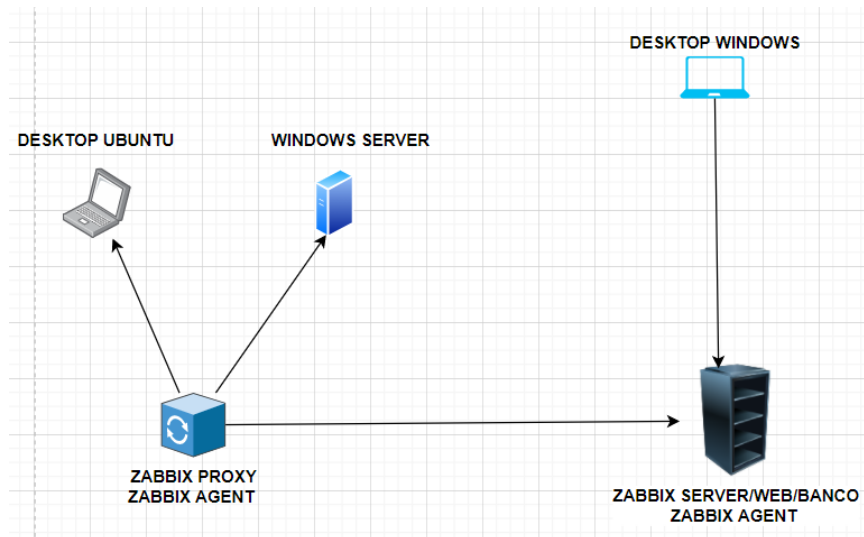


Figura 15 – Topologia do cenário de teste. Fonte: Próprio Autor.

3.2 Ferramenta Escolhida

As ferramentas estudadas apresentam muitas semelhanças entre si e, em geral, fornecem soluções para a maioria das necessidades que o gerenciamento de redes exige. Ainda assim, observa-se características únicas entre elas que sobressaem-se na hora da escolha. A Figura 16 apresenta de maneira geral um comparativo entre as funcionalidades de cada *software*.

Função/Ferramenta	Cacti	The Dude	Nagios	Zabbix
SLA Reports	Não	Não	Através de plugins	Sim
Auto Discovery	Através de Plugins	Sim	Através de plugins	Sim
Agente	Não	Não	Sim	Sim
SNMP	Sim	Sim	Através de plugins	Sim
Syslog	Não	Não	Através de plugins	Sim
Permite Scripts Externos	Sim	Não	Sim	Sim
Plugins	Sim	Não	Sim	Sim
Linguagem que foi escrito	PHP	C	Perl	C e PHP
Gatilhos/Alertas	Sim	Sim	Sim	Sim
Front end Web	Controle Completo	Controle Restrito	Controle Parcial	Controle Completo
Monitoramento Distribuído	Sim	Sim	Sim	Sim
Inventário	Através de Plugins	Sim	Através de plugins	Sim
Método de Armazenamento de Dados	RDDTool, MySQL, PostgreSQL	Arquivos	MySQL, MSSQL	Oracle, MySQL, PostgreSQL e SQLite
Licenciamento	GPL	Livre	GPL	GPL
Geração de Gráficos/Mapas	Sim/Através de plugins	Sim/Sim	Sim/Sim	Sim/Sim
Eventos	Através de Plugins	Sim	Sim	Sim

Figura 16 – Comparativo das funções disponibilizadas pelas ferramentas. Fonte: Próprio Autor.

A partir desta tabela pode-se analisar e verificar o que cada ferramenta fornece no que diz respeito às suas funcionalidades. Assim sendo, a partir desta lista, é possível representar particularidades de cada ferramenta e podendo deste modo concluir que o Zabbix possui praticamente todas as funções as quais são nativas da ferramenta, enquanto o Cacti e o The Dude faz-se necessário a instalação de plugins que possam habilitar ou facilitar determinada tarefa de monitoramento, além do Nagios que se mostrou um *software* limitado em sua versão gratuita, dependendo também de muitos plugins para efetuar algumas funcionalidades.

Ainda, deve-se levar em conta a questão do licenciamento de *software*, se ele é comercial ou não. Os produtos comerciais possuem um apelo maior no que diz respeito a suporte e ao conceito em geral de que há uma empresa que se responsabiliza pelo produto, sua possíveis falhas e futuras melhoras. O *software* GPL, por sua vez, possui uma comunidade maior e mais ativa e destacam-se principalmente, por sua flexibilidade e expansibilidade através de plugins. Ademais, após esta análise, verifica-se que o Zabbix consegue suprir de forma satisfatória, a maioria dos recursos existentes no âmbito de redes.

3.2.1 *Hardware e Software utilizado na solução escolhida*

Esta subseção aborda a metodologia, as ferramentas e os recursos computacionais a serem utilizados para efetuar o monitoramento através de um contêineres Docker com Zabbix.

Para realizar este trabalho, foi necessária a seleção do local, a definição dos requisitos com base no cenário de teste para o ambiente de execução do experimento, e as etapas pré estabelecidas para implantação dos serviços.

3.2.1.1 *Material escolhido*

O ambiente utilizado para construção do sistema computacional, se iniciou pela escolha do *software* de virtualização que seria utilizado como base projeto, e que estaria encarregada de alocar a maioria dos hosts utilizados, desde o Zabbix *Server* até uma simples aplicação Linux.

Assim sendo, foi utilizado o *hypervisor* VMWARE, este virtualizador numa máquina física com as seguintes características: 4 CPUs x Intel (R) Xeon (R) CPU E5606 a 2,13 GHz, 35,99 GB de memória, com dois datastore um com 828GB e o outro de 931 GB, além disso, trata-se de um modelo HP ProLiant DL160 G6.

Após esta etapa, foi feita a elaboração dos requisitos computacionais dos hosts utilizados. Segue abaixo requisitos e características de cada ativo:

- **Windows Server:** 2 vCPUs, 2 GB de memória, 20 GB de Disco, ISO Windows Server 2016;
- **Windows Desktop:** Processador Intel Core i5-7300U CPU @ 2.60GHz 2.70 GHz, 2 vCPUs, 2 GB de memória, 20 GB de Disco, ISO Windows 10 PRO (este item neste caso é um notebook físico que não está instalado na VMWARE);
- **Linux Desktop:** 2 vCPUs, 2 GB de memória, 20 GB de Disco, Ubuntu 20.04;
- **Zabbix Proxy:** 2 vCPUs, 2 GB de memória, 20 GB de Disco, Ubuntu 20.04;
- **Zabbix Server:** 3 vCPUs, 4 GB de memória, 30 GB de Disco, Ubuntu 20.04.

3.2.2 Etapas para a implantação da solução de monitoramento

Para a implantação desse projeto foi usado o Docker 20.10.5, uma tecnologia *open source*, que favorece a execução de *deploys*, além do escalonamento de aplicações com uma maior facilidade, possibilitando um ambiente leve e isolado para rodar o programa.

Sendo assim, a escolha do Docker foi feita para verificar se realmente a utilização de contêineres torna a instalação do Zabbix Server, Zabbix Proxy, Zabbix Agent e o Banco de Dados mais eficiente e flexível, já que o objetivo do Docker é tornar a instalação de aplicações mais rápida e leve.

Além disso, já existem imagens configuradas no repositório do Zabbix para instalação destes serviços, porém o intuito desse projeto é além de realizar a instalação desses serviços via contêiner, mas também efetuar modificações nos scripts para mostrar alguns conceitos como o Docker Hub, Dockerfile e Docker-compose.

Portanto, o intuito do trabalho é elaborar um ambiente de micros serviços Docker, implantar um cenário de monitoramento, utilizando contêineres Docker, com dependência de algumas aplicações externa; avaliar e homologar a implementação de um ambiente computacional gerenciado por Zabbix através de contêineres Docker em uma página modelo e homologar uma solução Docker que implemente um monitorador de ativos de rede por meio do Zabbix customizável.

Dessa maneira, foi realizada a instalação do Zabbix 5.2, contendo um front end Apache com banco de dados MYSQL 8.0 ambos em sua última versão, e os binários do Zabbix Server 5.2. Já pro Zabbix Proxy foi utilizado os binários do Zabbix Proxy 5.2 junto ao SQLITE3, responsável por fazer o armazenamento dos dados no Proxy, e para os Agents Windows foi feita a instalação tradicional dos Agents Zabbix 5.2 já que a instalação via contêiner ainda não é suportada. Para os Agents Linux foi utilizado os binários e informações dos Agents 5.2 utilizando contêineres para instalação.

Após essa etapa de instalação dos serviços Zabbix em todos os hosts listados no ambiente computacional, é necessário destacar quais os componentes de cada ativo de rede serão monitorados de uma maneira geral, e os principais itens a serem monitorados.

Com isso, obtendo o monitoramento de todos os ativos, é possível criar gráficos que possibilitam a melhor visualização dos dados pelo usuário da plataforma. Além disso, após a configuração de todos os itens é apresentando como é feita a criação de triggers, que nada mais são do que gatilhos que tem por objetivo avisar que algum item está apresentando um tipo de comportamento incomum, ou seja, a trigger permite definir um limite aceitável de dados, quando o dado recebido ultrapassar o limite aceitável a trigger será acionada, mudando seu estado para “incidente”.

Desta maneira, passando por etapas comuns para implementação de um *software* de monitoramento, como: instalação, escolha dos ativos de redes a serem monitorados, especificação de triggers, apresentação gráfica de dados, a próxima etapa do documento visa estabelecer formas de elaborar SLAs dentro do projeto.

Logo, além de monitorar itens comuns de rede é de suma importância para um estabelecimento comercial ou uma organização computacional conseguir estabelecer que não só a empresa contratante, como a própria prestadora de serviço estão prestando as ações estipuladas no contrato. Assim sendo, esse tipo de monitoramento constitui todos os serviços que precisam apresentar uma visão de alto nível da infraestrutura, com o objetivo de exibir detalhes de um serviço provido por um setor de TI.

Dessa forma, foi criado itens de serviço no Zabbix com os principais indicadores de desempenho que estão intimamente relacionados, como indisponibilidade de host.

Para facilitar a execução deste trabalho, as ações de teste foram divididas em três etapas. A primeira etapa será a implantação do Zabbix *Server*, Zabbix *Agent*, Zabbix-Front e Banco de Dados MYSQL através de linhas de comando Docker. O intuito com isso é apresentar ao leitor funcionalidades básicas do Docker, mostrar onde ficam armazenadas as imagens das aplicações e como essas imagens são montadas.

A segunda etapa será a implantação de serviços Zabbix utilizando o Docker-compose, com o objetivo de definir e executar microsserviço Docker simultaneamente e integrados. E por último, a terceira etapa serão configurados o Zabbix *Server*, Front-end, Banco de Dados, Zabbix *Proxy* e Zabbix *Agent*, além de aplicar essas funcionalidades nos hosts que terão seus ativos monitorados, por conseguinte no final deste capítulo o intuito é possuir um ambiente computacional igual ao da Figura 15.

Ademais, antes de chegar no estágio final, todo subcapítulo contará com uma preparação o ambiente Docker e Docker-compose, em que são descritos procedimentos

realizados para definir a estrutura da aplicação para a efetivação do experimento, que irá conter uma convergência de tecnologias e conhecimentos adquiridos ao longo do projeto, o intuito foi o de criar um arquivo YAML de própria autoria, que faça o resgate de imagens de serviços no Docker Hub e execute todas aplicações necessárias para criar um sistema completo de monitoramento.

CAPÍTULO 4

RESULTADOS

Neste capítulo, serão apresentados os resultados obtidos com base nas etapas definidas no capítulo anterior.

4.1 Etapa 1: Implantação de serviços Zabbix através de linhas de comando Docker

Nesta etapa, foram instanciados quatro contêineres, sendo o primeiro container para o banco de dados MYSQL, o segundo container é para a aplicação do Zabbix *Server*, o terceiro container é para criar o Zabbix-Web e o quarto container será para criar o Zabbix *Agent*. Para cada container foi utilizado imagens armazenadas no Docker Hub, criadas por engenheiros Docker, logo, este fato garante a integridade, eficiência e confiabilidade da imagem utilizada.

O primeiro passo foi realizar a instalação do Docker, como o pacote de instalação do Docker disponível no repositório oficial do Ubuntu pode não ser a versão mais recente, para garantir que seja utilizada a versão mais recente, foi instalado o Docker através do repositório oficial do *software*. Para que isso seja feito é necessário adicionar uma nova fonte de pacote, adicionar a chave GPG do Docker para garantir que os downloads sejam válidos, e então instalar o pacote. A seguir estão listados os procedimentos usados para instalar o Docker no Ubuntu 20.04.

Primeiro, é necessário atualizar a lista existente de pacotes (Lista 4.1):

```
1 # sudo apt-get update
```

Lista 4.1 – Comando para atualizar o repositório de aplicativos do Linux.

Em seguida, foi feita a instalação de alguns pacotes que deixam o apt usar pacotes pelo HTTPS (Lista 4.2):

```
1 # sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Lista 4.2 – Comando para instalar o suporte ao HTTPS.

Então, foi adicionada a chave GPG para o repositório oficial do Docker no sistema (Lista 4.3):

```
1 # sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Lista 4.3 – Comando para adicionar chave do repositório.

Foi acrescentado o repositório do Docker as fontes do APT (Lista 4.4):

```
1 # sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
```

Lista 4.4 – Comando para adicionar o repositório do Docker.

Em seguida, o banco de dados foi atualizado com os pacotes do Docker do recém adicionado repositório (Lista 4.5):

```
1 # sudo apt-get update
```

Lista 4.5 – Comando para atualizar a lista dos aplicativos no repositório

Certificar que irá prestes a instalar do repositório do Docker ao invés do repositório padrão do Ubuntu (Lista 4.6):

```
1 # sudo apt-cache policy docker-ce
```

Lista 4.6 – Comando para adicionar chave do repositório

Verificar se o serviço estava sendo executado (Lista 4.7):

```
1 # sudo systemctl status docker
```

Lista 4.7 – Comando para verificar o status do serviço Docker.

Após concluir esses passos iniciais para instalação do *software*, foi dado andamento na configuração dos serviços Zabbix, Apache e MYSQL, os comandos utilizados estão descritos abaixo:

Foi feita a criação de um diretório para que os dados do banco persistam (Lista 4.8):

```
1 # sudo mkdir -p /docker/mysql/zabbix/data
```

Lista 4.8 – Comando para criar o diretório para o banco de dados.

Depois foi necessário baixar a última versão das imagens Docker do MYSQL, Zabbix-Server e Zabbix-Web. Como já foi mencionado previamente o Docker possui um local público chamado de Docker Hub, em que várias pessoas e empresas publicam imagens pré-compiladas de soluções computacionais, desta maneira, este local nada mais é do que um repositório que ficam armazenados soluções em forma de contêineres.

Por conta disso, existem dois tipos de imagens, as criadas por usuários, estas contam muitas vezes com configurações particulares de um sistema e que na maioria das vezes não se tem procedência do código, por outro lado existem também as imagens desenvolvidas por engenheiros de aplicações de cada empresa, assim, estas imagens ao contrário das elaboradoras por usuários possuem procedência. A Figura 17, apresenta o repositório padrão do Zabbix no Docker Hub.

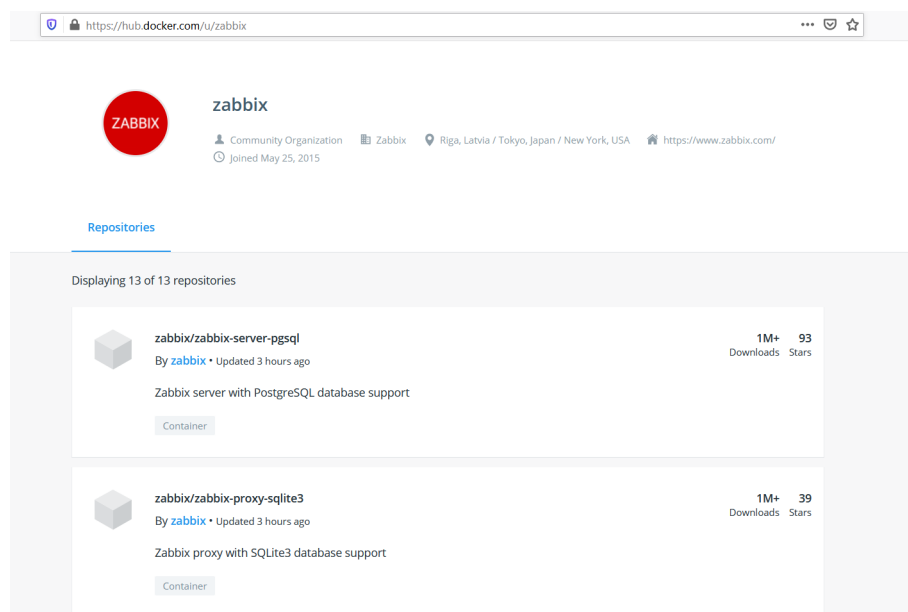


Figura 17 – Repositório Zabbix. Fonte: Próprio Autor.

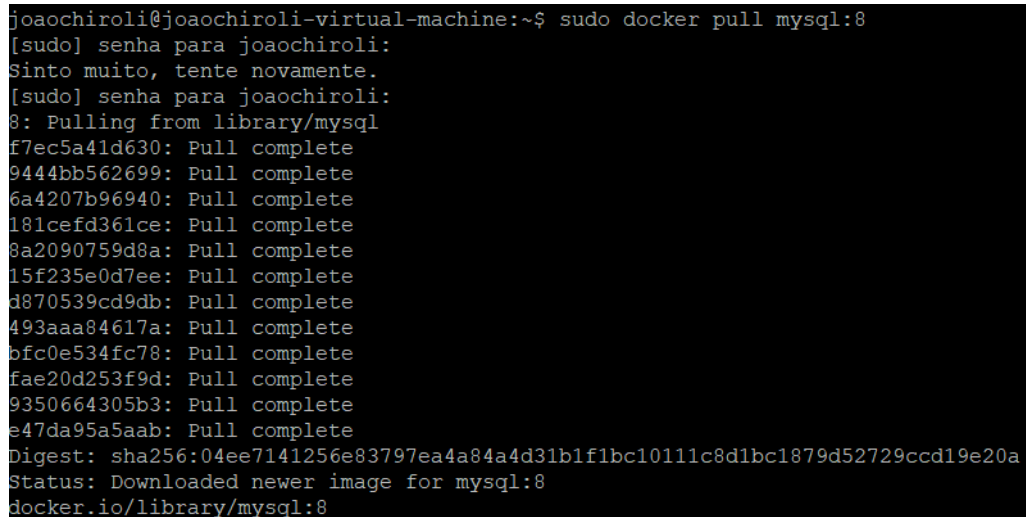
Assim sendo, foi necessário realizar a explicação prévia do conceito de Docker Hub para que o leitor saiba que o processo seguinte tenha sentido. Como já descrito no subcapítulo 2.3.1.1, foi necessário efetuar o pull das imagens, considerando que seria

utilizado a versão 5.2 do Zabbix e MYSQL, esta etapa foi feita da seguinte maneira (Lista 4.9):

```
1 # sudo docker pull mysql:8
2 # sudo docker pull zabbix/zabbix-server-mysql:ubuntu-5.2-latest
3 # sudo docker pull zabbix/zabbix-web-apache-mysql:ubuntu-5.2-latest
```

Lista 4.9 – Comandos para baixar as imagens dos contêineres.

Como apresentada na Figura 18, a utilização de contêineres é vantajosa porque no momento em que é realizado o download da imagem é feito também a instalação de seus binários um após o outro, seguindo o formato de pilha (estruturas de dados que armazenam os elementos em um formato sequencial). Contudo a vantagem não se encontra na forma como é realizada a instalação da imagem, mas sim no reaproveitamento dos binários já instalados.



```
joaochirolis@joaochirolis-virtual-machine:~$ sudo docker pull mysql:8
[sudo] senha para joaochirolis:
Sinto muito, tente novamente.
[sudo] senha para joaochirolis:
8: Pulling from library/mysql
f7ec5a41d630: Pull complete
9444bb562699: Pull complete
6a4207b96940: Pull complete
181cefd361ce: Pull complete
8a2090759d8a: Pull complete
15f235e0d7ee: Pull complete
d870539cd9db: Pull complete
493aaa84617a: Pull complete
bfc0e534fc78: Pull complete
fae20d253f9d: Pull complete
9350664305b3: Pull complete
e47da95a5aab: Pull complete
Digest: sha256:04ee7141256e83797ea4a84a4d31b1f1bc10111c8d1bc1879d52729ccd19e20a
Status: Downloaded newer image for mysql:8
docker.io/library/mysql:8
```

Figura 18 – Pull do MYSQL 8. Fonte: Próprio Autor.

Logo, se porventura um usuário já tiver um contêiner executando uma aplicação Ubuntu 18.04 e queira instalar o Ubuntu 20.04 a instalação deste último Sistema Operacional será mais rápida, pelo fato do Docker conseguir reaproveitar binários já instalados.

Após essas etapas foi necessário iniciar o contêiner Docker do MYSQL, criando o banco de dados para o Zabbix (Quadro 4.10):

```
1 # sudo docker run -d --name mysql-zabbix \
2 --restart always \
3 -p 3306:3306 \
4 -v /docker/mysql/zabbix/data:/var/lib/mysql \
5 -e MYSQL_HOST=172.17.0.1 \
6 -e MYSQL_ROOT_PASSWORD=secret \
```

```
7 -e MYSQL_DATABASE=zabbix \
8 -e MYSQL_USER=zabbix \
9 -e MYSQL_PASSWORD=zabbix \
10 mysql:8 \
11 --default-authentication-plugin=mysql_native_password \
12 --character-set-server=utf8 \
13 --collation-server=utf8_bin
```

Lista 4.10 – Comando Docker para criar o contêiner do banco de dados

Através do comando abaixo, é iniciado o contêiner Docker do Zabbix-Server acessando o banco de dados criado na etapa anterior (Lista 4.11):

```
1 # sudo docker run -d --name zabbix-server \
2 --restart always \
3 -p 10051:10051 \
4 -e MYSQL_ROOT_PASSWORD="secret" \
5 -e DB_SERVER_HOST="172.17.0.1" \
6 -e DB_SERVER_PORT="3306" \
7 -e MYSQL_USER="zabbix" \
8 -e MYSQL_PASSWORD="zabbix" \
9 -e MYSQL_DATABASE="zabbix" \
10 zabbix/zabbix-server-mysql:ubuntu-5.2-latest
```

Lista 4.11 – Comando Docker para criar o contêiner do Zabbix-Server

Seguindo as linhas de comando para instalação, foi iniciado o contêiner Zabbix-Web sem HTTPS (Lista 4.12):

```
1 # sudo docker run -d --name zabbix-web \
2 --restart always \
3 -p 80:8080 \
4 -e DB_SERVER_HOST="172.17.0.1" \
5 -e DB_SERVER_PORT="3306" \
6 -e MYSQL_USER="zabbix" \
7 -e MYSQL_PASSWORD="zabbix" \
8 -e MYSQL_DATABASE="zabbix" \
9 -e ZBX_SERVER_HOST="172.17.0.1" \
10 -e PHP_TZ="America/Sao_Paulo" \
11 zabbix/zabbix-web-apache-mysql:ubuntu-5.2-latest
```

Lista 4.12 – Comando Docker para criar o contêiner do Zabbix-web

Após todas essas etapas, a instalação foi feita com sucesso como representada na Figura 19. Além disso, é importante salientar que pode demorar cerca de 5 a 8 minutos para que todos os serviços consigam se comunicar e a interface do Zabbix apareça no browser.

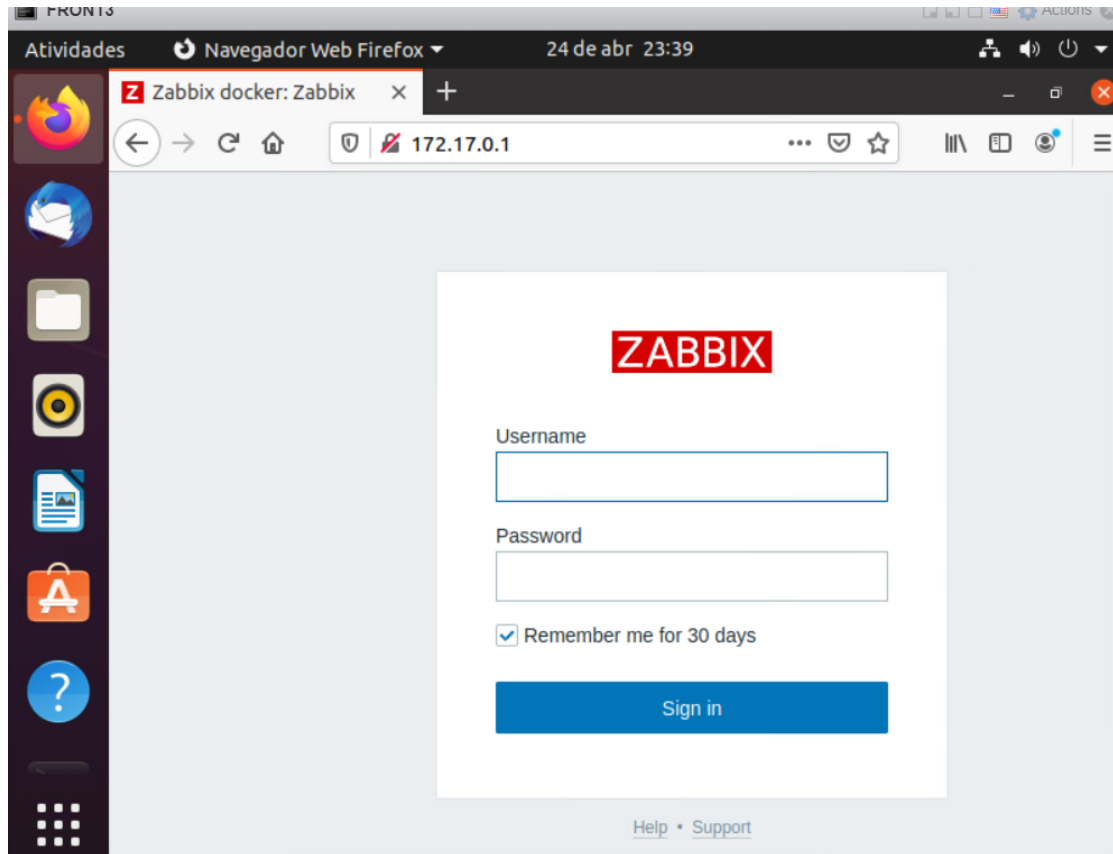


Figura 19 – Interface inicial do Zabbix Web. Fonte: Próprio Autor.

Ademais, com o intuito de informar o leitor sobre o que significa as siglas e os comandos utilizados nas linhas de comandos executadas acima, abaixo estão descritos o que cada elemento faz:

- -d: usado para executar um contêiner no modo desanexado;
- --name: o nome que será dado ao contêiner;
- --restart: utilizado para reiniciar o contêiner;
- -p: indica a porta utilizada no serviço;
- -v: tem por objetivo de atuar com configuração volume de armazenamento de dados;
- -e: indica uma variável de ambiente.

4.2 Etapa 2: Implantação de serviços Zabbix através do Docker-compose

Assim sendo, como a intenção do trabalho é procurar uma forma mais rápida e eficiente para instalação da solução de monitoramento, foi levado em consideração o

Docker-compose. Esta ferramenta serve para definir e executar aplicativos Docker de vários contêineres.

O Docker-compose é usado por meio de um arquivo YAML para configurar serviços do aplicativo, então com um único comando, é possível criar e iniciar todos os serviços de configuração. A tecnologia funciona em todos os ambientes: produção, teste, desenvolvimento, bem como fluxo de trabalho de integração contínua. Utilizar esta ferramenta é basicamente um processo de três etapas:

- Definir o ambiente do aplicativo com um Dockerfile para que ele possa ser reproduzido em qualquer lugar;
- Definir os serviços que compõem o aplicativo docker-compose.yml para que possam ser executados juntos em um ambiente isolado;
- Execute *docker compose up* e o comando Docker compose inicia e executa todo o aplicativo. Podendo também alternativamente executar *docker-compose up -d* usando o binário Docker-compose

Além disso, o arquivo docker-compose.yml é apresentado na Figura 20.

```
version: "3.9" # optional since v1.27.0
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Figura 20 – Exemplo de arquivo YAML. Fonte: Próprio Autor.

Dessa forma, neste subcapítulo foram registrados os passos para iniciar uma aplicação usando o Docker-compose, além disso, essa etapa foi empregada através de um arquivo YAML do repositório oficial do Zabbix no Git Hub.

Portanto, a primeira etapa para o começo do processo é realizar a instalação do Docker Compose, para que os comandos sejam executados, abaixo estão listados respectivamente: o método de download, a forma para tornar o arquivo executável e como verificar a versão da ferramenta instalada (Lista 4.13):

```
1 # sudo curl -L "https://github.com/docker/compose/releases/download
    /1.29.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin
    /docker-compose
2 # sudo chmod +x /usr/local/bin/docker-compose
3 # sudo docker-compose --version
```

Lista 4.13 – Comandos para a execução do docker compose e verificação da versão do docker-compose

Após esta etapa é necessário instalar o git *client* para que se possa efetuar o download dos arquivos YAML do Zabbix armazenados no Git Hub. A etapa foi realizada da seguinte forma (Lista 4.14):

```
1 # sudo apt-get install git
```

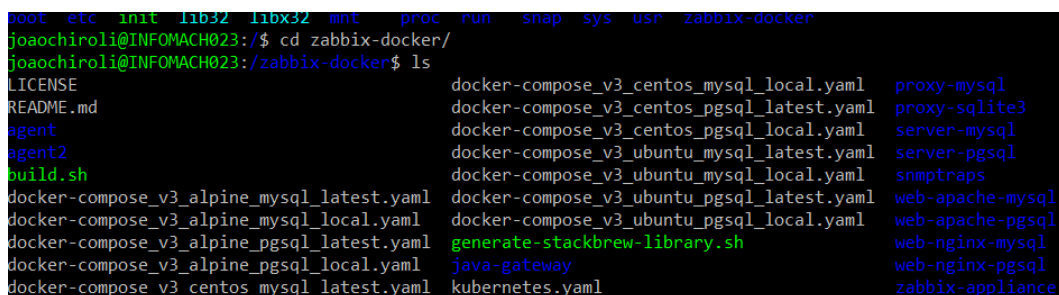
Lista 4.14 – Comando para instalar o GIT

Com o git já instalado é necessário efetuar o download do arquivo YAML e foi utilizado os seguintes comandos (Lista 4.15):

```
1 # sudo git clone https://github.com/zabbix/zabbix-docker.git
```

Lista 4.15 – Comando para clonar o repositório do Zabbix

E para saber quais arquivos estão contidos na pasta zabbix-docker é necessário entrar na pasta e depois listar quais os arquivos foram baixados. Os comandos estão listados na Lista 4.16 e a Figura 21 apresenta este contexto.



```
root etc init lib32 libx32 mnt proc run snap sys usr zabbix-docker
joaochirol@INFOMACH023:/$ cd zabbix-docker/
joaochirol@INFOMACH023:/zabbix-docker$ ls
LICENSE                                docker-compose_v3_centos_mysql_local.yaml    proxy-mysql
README.md                             docker-compose_v3_centos_pgsql_latest.yaml    proxy-sqlite3
agent                                 docker-compose_v3_centos_pgsql_local.yaml      server-mysql
agent2                               docker-compose_v3_ubuntu_mysql_latest.yaml     server-pgsql
build.sh                             docker-compose_v3_ubuntu_mysql_local.yaml      snmptraps
docker-compose_v3_alpine_mysql_latest.yaml  docker-compose_v3_ubuntu_pgsql_latest.yaml     web-apache-mysql
docker-compose_v3_alpine_mysql_local.yaml   docker-compose_v3_ubuntu_pgsql_local.yaml       web-apache-pgsql
docker-compose_v3_alpine_pgsql_latest.yaml  generate-stackbrew-library.sh                  web-nginx-mysql
docker-compose_v3_alpine_pgsql_local.yaml   java-gateway                                    web-nginx-pgsql
docker-compose_v3_centos_mysql_latest.yaml  kubernetes.yaml                                zabbix-appliance
```

Figura 21 – Apresentação dos comandos Linux e do arquivo YAML. Fonte: Próprio Autor.

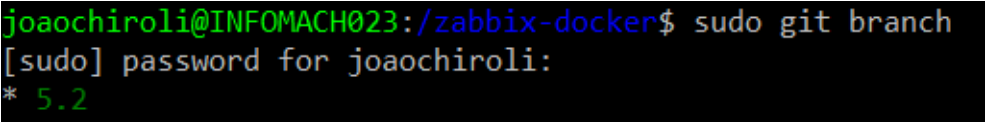
```
1 # cd zabbix-docker
2 # ls
```

Lista 4.16 – Comando do console para mudar de diretório e listar arquivos

Com todos os arquivos já presentes no *host*, é necessário mostrar o ramo atual, mostrar *tags* do ramo 5.2, copiar o arquivo que substitui a referência à imagem docker mais recente com 5.2 e depois substituir o arquivo mais recente por 5.2. Abaixo estão listados os comandos (Lista 4.17, 4.18 4.19 e 4.20) e suas respectivas imagens (Figura 22, 23, 24, 25):

```
1 # sudo git branch
```

Lista 4.17 – Comando git para substituir os arquivos atuais

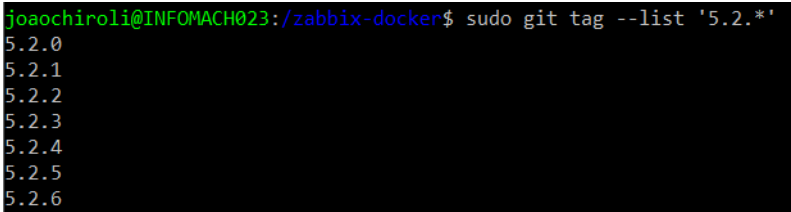


```
joaochirolis@INFOMACH023:/zabbix-docker$ sudo git branch
[sudo] password for joaochirolis:
* 5.2
```

Figura 22 – Apresentação do comando *git branch*. Fonte: Próprio Autor.

```
1 # sudo git tag --list '5.2.*'
```

Lista 4.18 – Comando git para listar todas as versões 5.2

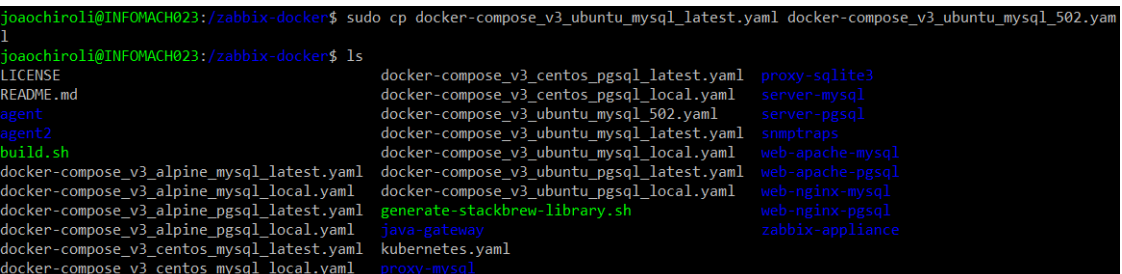


```
joaochirolis@INFOMACH023:/zabbix-docker$ sudo git tag --list '5.2.*'
5.2.0
5.2.1
5.2.2
5.2.3
5.2.4
5.2.5
5.2.6
```

Figura 23 – Apresentação do comando *git tag --list '5.2.*'*. Fonte: Próprio Autor.

```
1 # sudo cp docker-compose_v3_ubuntu_mysql_latest.yaml docker-
  compose_v3_ubuntu_mysql_502.yaml
```

Lista 4.19 – Comando Linux *cp*



```
joaochirolis@INFOMACH023:/zabbix-docker$ sudo cp docker-compose_v3_ubuntu_mysql_latest.yaml docker-compose_v3_ubuntu_mysql_502.yaml
joaochirolis@INFOMACH023:/zabbix-docker$ ls
LICENSE                                docker-compose_v3_centos_pgsql_latest.yaml  proxy-sqlite3
README.md                             docker-compose_v3_centos_pgsql_local.yaml    server-mysql
agent                                docker-compose_v3_ubuntu_mysql_502.yaml      server-pgsql
agent2                               docker-compose_v3_ubuntu_mysql_latest.yaml    snmptraps
build.sh                             docker-compose_v3_ubuntu_mysql_local.yaml     web-apache-mysql
docker-compose_v3_alpine_mysql_latest.yaml  docker-compose_v3_ubuntu_pgsql_latest.yaml    web-apache-pgsql
docker-compose_v3_alpine_mysql_local.yaml  docker-compose_v3_ubuntu_pgsql_local.yaml     web-nginx-mysql
docker-compose_v3_alpine_pgsql_latest.yaml  generate-stackbrew-library.sh                 web-nginx-pgsql
docker-compose_v3_alpine_pgsql_local.yaml  java-gateway                                   zabbix-appliance
docker-compose_v3_centos_mysql_latest.yaml  kubernetes.yaml
docker-compose_v3_centos_mysql_local.yaml   proxy-mysql
```

Figura 24 – Apresentação do comando Linux *cp* e *ls*. Fonte: Próprio Autor.

```
1 # sudo sed -i "s/-5.2-latest/-5.2.0/" docker-
  compose_v3_ubuntu_mysql_502.yaml
```

Lista 4.20 – Comando *sed*

```

docker-compose_v3_centos_mysql_local.yaml proxy-mysql
joaochirol1@INFOMACH023:/zabbix-docker$ sudo sed -i "s/-5.2-latest/-5.2.0/" docker-compose_v3_ubuntu_mysql_502.yaml
joaochirol1@INFOMACH023:/zabbix-docker$

```

Figura 25 – Apresentação de substituição para um arquivo mais recente. Fonte: Próprio Autor.

Após estas etapas, é necessário executar o arquivo YAML, através do comando:

```

1 # sudo docker-compose -f ./docker-compose_v3_ubuntu_mysql_502.yaml up -
  d

```

Lista 4.21 – Comando de execução do arquivo YAML

Logo depois dessa última execução, o Docker-compose realiza o *pull* das imagens de modo automática e faz o *run* dos contêineres para que os serviços funcionem, ou seja, todos os passos que no subcapítulo anterior foi feita parte por parte, com o Docker-compose é tudo feita de maneira automática como apresentado nas Figuras 26 e 27.

```

Creating network "zabbix-docker_zbx_net_backend" with driver "bridge"
Creating network "zabbix-docker_zbx_net_frontend" with driver "bridge"
Creating network "zabbix-docker_default" with the default driver
Creating volume "zabbix-docker_snmptraps" with default driver
Pulling mysql-server (mysql:8.0)...
8.0: Pulling from library/mysql
f7ec5a41d630: Pull complete
9444bb562699: Pull complete
6a4207b96940: Pull complete
181cefd361ce: Pull complete
8a2090759d8a: Pull complete
15f235e0d7ee: Pull complete
d870539cd9db: Pull complete
493aaa84617a: Pull complete
bfc0e534fc78: Pull complete
fae20d253f9d: Pull complete
9350664305b3: Pull complete
e47da95a5aab: Pull complete
Digest: sha256:04ee7141256e83797ea4a84a4d31b1f1bc10111c8d1bc1879d52729ccd19e20a
Status: Downloaded newer image for mysql:8.0
Pulling zabbix-server (zabbix/zabbix-server-mysql:ubuntu-5.2.0)...
ubuntu-5.2.0: Pulling from zabbix/zabbix-server-mysql
6a5697faee43: Pull complete
ba13d3bc422b: Pull complete
a254829d9e55: Pull complete
9e28a4f86fc8: Pull complete
df66e2d67a5e: Pull complete
b5d586deb7ee: Pull complete
0523c4161df0: Pull complete
Digest: sha256:5c17c342fa80555ec5871f1dc5ad49427cbc57c8625d1a4f6381d6e0f59e8749
Status: Downloaded newer image for zabbix/zabbix-server-mysql:ubuntu-5.2.0
Pulling zabbix-web-nginx-mysql (zabbix/zabbix-web-nginx-mysql:ubuntu-5.2.0)...
ubuntu-5.2.0: Pulling from zabbix/zabbix-web-nginx-mysql
6a5697faee43: Already exists
ba13d3bc422b: Already exists
a254829d9e55: Already exists
2f6063acccc6: Pull complete
4591318d9bb9: Pull complete
b927a2d2ad69: Pull complete
b57cadda8c8f: Pull complete
90e64d4b8ece: Pull complete
Digest: sha256:023619422ea197669dd14cf39cb9f9841e71fc831ad2fa749de2c39ba85dee52
Status: Downloaded newer image for zabbix/zabbix-web-nginx-mysql:ubuntu-5.2.0
Pulling db_data_mysql (busybox:...)...
latest: Pulling from library/busybox
f531cdc67389: Pull complete
Digest: sha256:ae39a6f5c07297d7ab64dbd4f82c77c874cc6a94cea29fdec309d0992574b4f7

```

Figura 26 – *Pull* das imagens. Fonte: Próprio Autor.

Dessa maneira, após todas etapas o resultado foi semelhante ao do subcapítulo anterior, todavia, é importante salientar que não necessariamente é preciso fazer uma cópia

```
Status: Downloaded newer image for busybox:latest
Creating zabbix-docker_db_data_mysql_1 ... done
Creating zabbix-docker_mysql-server_1 ... done
Creating zabbix-docker_zabbix-server_1 ... done
Creating zabbix-docker_zabbix-web-nginx-mysql_1 ... done
```

Figura 27 – Run dos contêineres. Fonte: Próprio Autor.

do arquivo YAML do Zabbix para depois executá-lo, se for o caso de baixar os arquivos e após isso já querer subir os contêineres, basta usar o comando: .

```
1 # sudo docker-compose -f ./<nome_do_arquivo> up -d
```

Lista 4.22 – Comando de execução do arquivo YAML

Neste caso foi feita uma cópia, porque a intenção era preservar o arquivo original e além disso, testar algumas linhas de comando Linux, como permissão e cópia.

4.3 Etapa 3: Implantação e comunicação dos serviços Zabbix através do Docker-compose criado pelo próprio autor

Após as duas primeiras etapas iniciais, foi realizada a junção de todos conhecimentos adquiridos neste trabalho. Portanto esta etapa exigiu além de conhecimentos sobre Docker, Docker-compose e Zabbix, um grande conhecimento de redes e Linux, já que grande parte dos contêineres foi criado em um ambiente Ubuntu, e para que os contêineres se comunicassem foi necessário ter uma compreensão sólida sobre os conceitos de rede e sub-rede.

A princípio, para dar início a este subcapítulo, foi necessário realizar a instalação do Docker e Docker-compose, método que já foi apresentado nos dois subcapítulos anteriores, vale ressaltar que para executar algum arquivo YAML utilizando o Docker-compose é necessário fazer a instalação do Docker já que esta ferramenta é responsável pela construção dos contêineres.

Sendo assim, a primeira etapa foi realizar a construção de 3 arquivos YAML, sendo eles:

- ZabbixAgentCompose.yaml: responsável por conter informações do Zabbix *Agent*;
- ZabbixProxyCompose.yaml: responsável por conter informações do Zabbix *Agent* e Zabbix *Proxy*;
- ZabbixServerComplete.yaml: responsável por conter informações do Zabbix *Agent*, MySQL, Zabbix *Server*, Zabbix-Web.

Os arquivos foram escritos e colocados no GitHub, com o intuito de realizar o download dos itens em todas as máquinas de maneira ágil como apresentado na Figura 28.

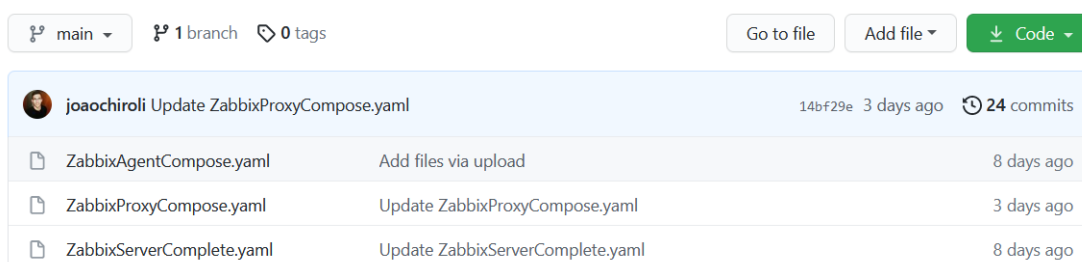


Figura 28 – Imagem dos 3 arquivos YAML . Fonte: Próprio Autor.

Após ter realizado a separação de cada processo em um arquivo, foi feita a criação do script de serviços no arquivo YAML, especificando: imagem utilizada, nome do container, nome do serviço, portas, variáveis de ambiente, entre outros.

O primeiro script criado foi o ZabbixServerComplete.yaml, este arquivo como já descrito previamente contém ao todo 4 serviços, primeiro foi necessário especificar a versão do Docker-compose e criar uma network em modo bridge para comunicação entre os serviços contidos no arquivo YAML, como apresentado na Figura 29

```

1  version: "3.5"
2
3  networks:
4    network-zabbix:
5      driver: bridge
6

```

Figura 29 – Imagem dos 3 arquivos YAML . Fonte: Próprio Autor.

Após esta etapa, foi feita a especificação do banco de dados MYSQL contendo uma imagem do serviço na versão 8.0, especificando o nome do contêiner, realizando uma ação de *restart* do container caso seja feita alguma alteração, declarando quais portas serão utilizadas, detalhando alguns comandos utilizados pelo banco de dados, definindo onde será alocado informações no host e no próprio contêiner através do volume, além de adicionar variáveis de ambiente utilizadas pelo banco de dados e empregar a network criada anteriormente.

É de suma importância salientar que no momento da utilização de um grupo de contêineres em um host, o Docker automaticamente cria uma sub-rede interna para que estes contêineres se comuniquem uns com os outros, normalmente esta sub-rede possui a faixa 172.17.0.1. Desta maneira, no momento da criação do arquivo YAML, foi passado

para o banco de dados esta faixa de rede nas variáveis de ambiente como mostrado abaixo (Lista 4.23):

```

1 # services:
2   mysql-server:
3     image: mysql:8.0
4     container_name: mysql-zabbix
5     restart: always
6     ports:
7       - '3306:3306'
8     command:
9       - mysqld
10      - --character-set-server=utf8
11      - --collation-server=utf8_bin
12      - --default-authentication-plugin=mysql_native_password
13     volumes:
14       - ./zbx_env/var/lib/mysql:/var/lib/mysql:rw
15     environment:
16       - MYSQL_HOST=172.17.0.1
17       - MYSQL_ROOT_PASSWORD=secret
18       - MYSQL_DATABASE=zabbix
19       - MYSQL_USER=zabbix
20       - MYSQL_PASSWORD=zabbix
21     networks:
22       - network-zabbix

```

Lista 4.23 – Arquivo YAML contendo o serviço MYSQL

Após esta fase, foi feita a criação do script para o serviço do Zabbix-server. Praticamente este serviço usa quase todos os tópicos utilizados pelo MYSQL, porém, existe um tópico a mais, o método links, este recurso tem por objetivo realizar ligações automáticas dentro da rede entre um serviço e outro, permitindo assim que diferentes serviços consigam se comunicar como mostrado abaixo (Lista 4.24):

```

1 # zabbix-server:
2   container_name: zabbix-server
3   image: zabbix/zabbix-server-mysql:ubuntu-5.2-latest
4   networks:
5     - network-zabbix
6   links:
7     - mysql-server
8   restart: always
9   ports:
10     - '10051:10051'
11   volumes:
12     - /etc/localtime:/etc/localtime:ro

```

```

13     - ./zbx_env/usr/lib/zabbix/alertscripts:/usr/lib/zabbix/
      alertscripts:ro
14     - ./zbx_env/usr/lib/zabbix/externalscripts:/usr/lib/zabbix/
      externalscripts:ro
15     - ./zbx_env/var/lib/zabbix/export:/var/lib/zabbix/export:rw
16     - ./zbx_env/var/lib/zabbix/modules:/var/lib/zabbix/modules:ro
17     - ./zbx_env/var/lib/zabbix/enc:/var/lib/zabbix/enc:ro
18     - ./zbx_env/var/lib/zabbix/ssh_keys:/var/lib/zabbix/ssh_keys:ro
19     - ./zbx_env/var/lib/zabbix/mibs:/var/lib/zabbix/mibs:ro
20   environment:
21     - MYSQL_ROOT_PASSWORD=secret
22     - DB_SERVER_HOST=172.17.0.1
23     - DB_SERVER_PORT=3306
24     - MYSQL_USER=zabbix
25     - MYSQL_PASSWORD=zabbix
26     - MYSQL_DATABASE=zabbix
27   depends_on:
28     - mysql-server

```

Lista 4.24 – Arquivo YAML contendo o serviço Zabbix Server

Assim sendo, foram criados os demais serviços do arquivo ZabbixServerComplete.yaml, seguindo as mesmas métricas apresentadas previamente, como mostrado a seguir (Lista 4.25):

```

1  # zabbix-frontend:
2    container_name: zabbix-frontend
3    image: zabbix/zabbix-web-apache-mysql:ubuntu-5.2-latest
4    networks:
5      - network-zabbix
6    links:
7      - mysql-server
8    restart: always
9    volumes:
10     - /etc/localtime:/etc/localtime:ro
11     - ./zbx_env/etc/ssl/apache2:/etc/ssl/apache2:ro
12     - ./zbx_env/usr/share/zabbix/modules:/usr/share/zabbix/modules/:
      ro
13    ports:
14     - '80:8080'
15     - '443:8443'
16    environment:
17     - MYSQL_ROOT_PASSWORD_FILE=secret
18     - DB_SERVER_HOST=172.17.0.1
19     - DB_SERVER_PORT=3306
20     - MYSQL_USER=zabbix
21     - MYSQL_PASSWORD=zabbix

```

```

22     - MYSQL_DATABASE=zabbix
23     - ZBX_SERVER_HOST=172.17.0.1
24     - ZBX_SERVER_NAME=Composed installation
25     - PHP_TZ=America/Sao_Paulo
26     depends_on:
27       - mysql-server
28
29     zabbix-agent:
30       container_name: zabbix-agent
31       image: zabbix/zabbix-agent:ubuntu-5.2-latest
32       user: root
33       networks:
34         - network-zabbix
35       links:
36         - zabbix-server
37       restart: always
38       privileged: true
39       volumes:
40         - /etc/localtime:/etc/localtime:ro
41         - ./zbx_env/etc/zabbix/zabbix_agentd.d:/etc/zabbix/zabbix_agentd.d:ro
42         - ./zbx_env/var/lib/zabbix/modules:/var/lib/zabbix/modules:ro
43         - ./zbx_env/var/lib/zabbix/enc:/var/lib/zabbix/enc:ro
44         - ./zbx_env/var/lib/zabbix/ssh_keys:/var/lib/zabbix/ssh_keys:ro
45       ports:
46         - '10050:10050'
47       environment:
48         - ZBX_SERVER_HOST=172.18.0.1
49         - ZBX_HOSTNAME=Zabbix server

```

Lista 4.25 – Arquivo YAML contendo os serviços Zabbix Front e Zabbix Agent

Desta maneira, depois de concluído o script anterior foi necessário realizar a criação do ZabbixProxyCompose.yaml e ZabbixAgentCompose.yaml. Mostrados a seguir consecutivamente (Lista 4.26 e 4.27):

```

1 # version: "3.5"
2
3 networks:
4   network-zabbix:
5     driver: bridge
6
7 services:
8   zabbix-proxy:
9     container_name: zabbix-proxy
10    image: zabbix/zabbix-proxy-sqlite3:ubuntu-5.2-latest
11    restart: always

```



```

12     ports :
13         - '10051:10051'
14     environment :
15         - ZBX_HOSTNAME=zabbix-proxy
16         - ZBX_SERVER_HOST=zabbix-server
17         - ZBX_SERVER_PORT=10051
18     volumes :
19         - /etc/localtime:/etc/localtime:ro
20         - ./zbx_env/usr/lib/zabbix/externalscripts:/usr/lib/zabbix/
           externalscripts:ro
21         - ./zbx_env/var/lib/zabbix/modules:/var/lib/zabbix/modules:ro
22         - ./zbx_env/var/lib/zabbix/enc:/var/lib/zabbix/enc:ro
23         - ./zbx_env/var/lib/zabbix/ssh_keys:/var/lib/zabbix/ssh_keys:ro
24         - ./zbx_env/var/lib/zabbix/mibs:/var/lib/zabbix/mibs:ro
25     networks :
26         - network-zabbix
27
28     zabbix-agent :
29         container_name: zabbix-agent
30         image: zabbix/zabbix-agent:ubuntu-5.2-latest
31         user: root
32         networks :
33             - network-zabbix
34         links :
35             - zabbix-proxy
36         restart: always
37         privileged: true
38         volumes :
39             - /etc/localtime:/etc/localtime:ro
40             - ./zbx_env/etc/zabbix/zabbix_agentd.d:/etc/zabbix/
               zabbix_agentd.d:ro
41             - ./zbx_env/var/lib/zabbix/modules:/var/lib/zabbix/modules:ro
42             - ./zbx_env/var/lib/zabbix/enc:/var/lib/zabbix/enc:ro
43             - ./zbx_env/var/lib/zabbix/ssh_keys:/var/lib/zabbix/ssh_keys:ro
44         ports :
45             - '10050:10050'
46         environment :
47             - ZBX_SERVER_HOST=172.18.0.1
48             - ZBX_HOSTNAME=Zabbix_server

```

Lista 4.26 – Arquivo YAML do Zabbix Proxy

```

1 # version: "3.5"
2
3 networks :
4     network-zabbix :
5         driver: bridge

```

```

6
7 services:
8   zabbix-agent:
9     container_name: zabbix-agent
10    image: zabbix/zabbix-agent:ubuntu-5.2-latest
11    user: root
12    networks:
13      - network-zabbix
14    #links:
15      #- zabbix-server
16    restart: always
17    privileged: true
18    volumes:
19      - /etc/localtime:/etc/localtime:ro
20      - ./zbx_env/etc/zabbix/zabbix_agentd.d:/etc/zabbix/
21        zabbix_agentd.d:ro
22      - ./zbx_env/var/lib/zabbix/modules:/var/lib/zabbix/modules:ro
23      - ./zbx_env/var/lib/zabbix/enc:/var/lib/zabbix/enc:ro
24      - ./zbx_env/var/lib/zabbix/ssh_keys:/var/lib/zabbix/ssh_keys:ro
25    ports:
26      - '10050:10050'
27    environment:
28      - ZBX_SERVER_HOST=172.18.0.1
29      - ZBX_HOSTNAME=Zabbix_server

```

Lista 4.27 – Arquivo YAML do Zabbix Agent

Desta maneira, após o download da pasta no GitHub, foi feita a execução dos serviços através do comando (Lista 4.28):

```

1 # sudo docker-compose -f ./ZabbixServerComplete.yaml up -d

```

Lista 4.28 – Execução do arquivo YAML

Como mostra a Figura 30 e após esta etapa foi feita a verificação dos contêineres que estavam rodando, com o comando *sudo docker container ls* como apresentado na Figura 31.

Logo, depois de verificado qual serviço está rodando e em qual porta, basta esperar alguns minutos para acessar o link: <http://127.17.0.1>, este tempo de espera é necessário pois o banco de dados precisa ser criado, as tabelas precisam ser geradas e os itens destas tabelas precisam ser populados, por isso se faz necessário este tempo de espera para poder acessar o ambiente web do Zabbix neste caso.

Após concluído, já é possível acessar o ambiente web do Zabbix, contudo, mesmo executando um arquivo YAML que contenha todos os serviços, realizando a criação

```
joaochirolli@joaochirolli-virtual-machine:~/dockercompose$ sudo docker-compose -f ZabbixServerComplete.yaml up -d
Creating network "dockercompose_network-zabbix" with driver "bridge"
Pulling mysql-server (mysql:8.0)...
8.0: Pulling from library/mysql
f7ec5a41d630: Pull complete
9444bb562699: Pull complete
6a4207b96940: Pull complete
181cefd361ce: Pull complete
8a2090759d8a: Pull complete
15f235e0d7ee: Pull complete
d870539cd9db: Pull complete
493aaa84617a: Pull complete
bfc0e534fc78: Pull complete
fae20d253f9d: Pull complete
9350664305b3: Pull complete
e47da95a5aab: Pull complete
Digest: sha256:04ee7141256e83797ea4a84a4d31b1fbc10111c8d1bc1879d52729ccd19e20a
Status: Downloaded newer image for mysql:8.0
Pulling zabbix-server (zabbix/zabbix-server-mysql:ubuntu-5.2-latest)...
ubuntu-5.2-latest: Pulling from zabbix/zabbix-server-mysql
345e3491a907: Pull complete
57671312ef6f: Pull complete
5e9250ddb7d0: Pull complete
6f73ec37a41e: Pull complete
67db661e36be: Pull complete
b5403f80039c: Pull complete
Digest: sha256:eb2df21ff5e661c5482290f01b19b93719330bd16a83ac8441be6cc3eeb2baa2
Status: Downloaded newer image for zabbix/zabbix-server-mysql:ubuntu-5.2-latest
```

Figura 30 – Imagem da execução dos serviços. Fonte: Próprio Autor.

```
joaochirolli@joaochirolli-virtual-machine:~/dockercompose$ sudo docker container ls
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
07f16a4919e   zabbix/zabbix-agent:ubuntu-5.2-latest  "/usr/bin/tini -- /u..."  4 minutes ago  Up 4 minutes  0.0.0.0:10050->10050/tcp, :::10050->10050/tcp
b3c2221ca118   zabbix/zabbix-web-apache-mysql:ubuntu-5.2-latest  "docker-entrypoint.s..."  4 minutes ago  Up 4 minutes  0.0.0.0:80->8080/tcp, :::80->8080/tcp, 0.0.0.0:443->443/tcp
c2f2dc3f600d   zabbix/zabbix-server-mysql:ubuntu-5.2-latest  "/usr/bin/tini -- /u..."  4 minutes ago  Restarting (1) 23 seconds ago
4bec36c8bc6   mysql:8.0                               "docker-entrypoint.s..."  5 minutes ago  Restarting (1) 18 seconds ago
```

Figura 31 – Imagem dos contêineres rodando. Fonte: Próprio Autor.

do método de links para esses serviços se comunicarem o Zabbix *Agent* vai se apresentar um alerta no Zabbix Web de acessibilidade.

Este alerta é disparado porque no momento da criação dos contêineres o Zabbix *Agent* não é criado com sua variável de ambiente contendo o ip do contêiner do Zabbix *Server*, desta forma, a comunicação não acontece, sendo assim, para resolver esse problema é necessário utilizar o comando `sudo docker container inspect <id_container_zabbix_server>` para saber qual o ip do container do Zabbix *Server* na aplicação e depois disso é necessário mudar a variável de ambiente “`ZBX_SERVER_HOST=id_container_zabbix_server`”, acrescentando a ela o ip do Zabbix *Server* após isso, é necessário dar um restart no contêiner do Zabbix *Agent* para que a aplicação atualize seus dados.

Sendo assim, depois de feito todos esses procedimentos, precisa colocar no host criado do Zabbix Web o ip do container que possui a aplicação do Zabbix *Agent* como apresentada na Figura 32, após realizada todas as alterações necessárias algumas etapas se apresentam concluídas sendo elas:

- Zabbix *Server* rodando com todos serviços necessários para que esta aplicação seja e se comunique com outros componentes do ambiente, por consequência, trocando informações;

- Monitoramento dos ativos do host em que está o Zabbix Server, sendo efetuado através do Zabbix Agent.

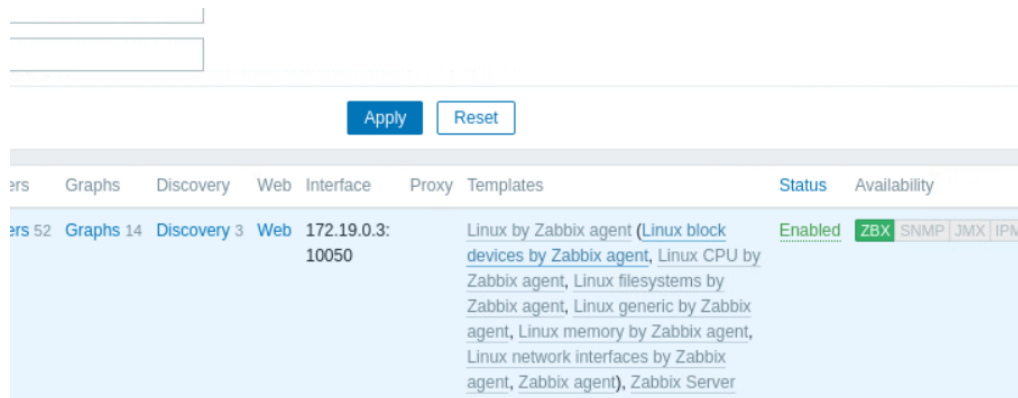


Figura 32 – Imagem do monitoramento do host em que está no Zabbix Server. Fonte: Próprio Autor.

Portanto, foi preciso configurar o host Windows-10 para enviar informações do monitoramento para o Zabbix Server. Primeiro foi feita a configuração do Zabbix Agent para sistemas Windows, após este procedimento foi feito a alteração da variável de ambiente *ServerActive* e *Hostname*, ambos presentes no arquivo *zabbix_agentd.conf*. Na variável *ServerActive* é colocado o ip do host que está executando o serviço do Zabbix Server, além de ser necessário explicitar para qual porta esses dados vão ser destinados, neste caso os dados vão ser enviados para a porta 10051. Já o *Hostname* é responsável por validar com o Zabbix Server que o host que está enviando dados é o host correto, já o ip que deve estar configurado no Zabbix Web é o ip do host que contém o Zabbix Agent. Além disso, o monitoramento do host Windows se apresenta acontecendo pela Figura 33.

Ademais, o Zabbix Agent contido no host Windows-10 é um Zabbix Agent Ativo, ou seja, ele é quem vai enviar as informações dos ativos de rede para o servidor Zabbix, por isso a necessidade de configurar o *ServerActive* e *Hostname*.

Dessa maneira, após confirmar que dois hosts já estão sendo monitorados, foi feita a execução do Zabbix Proxy e Zabbix Agent, e após este processo foi feita a verificação de quais contêineres estão rodando neste host.

Assim sendo, o Proxy utilizado neste cenário é um Proxy Ativo ou seja é ele quem vai buscar informações nos hosts e depois irá enviar os dados aos Zabbix Server, desta forma, após executar os serviços, bastou atualizar as configurações do contêiner que está executando o serviço do Zabbix Proxy para enviar os dados para o ip do host que contém o Zabbix Server. Depois desta etapa, foi necessário fazer a validação da comunicação entre os serviços como mostra a Figura 34.

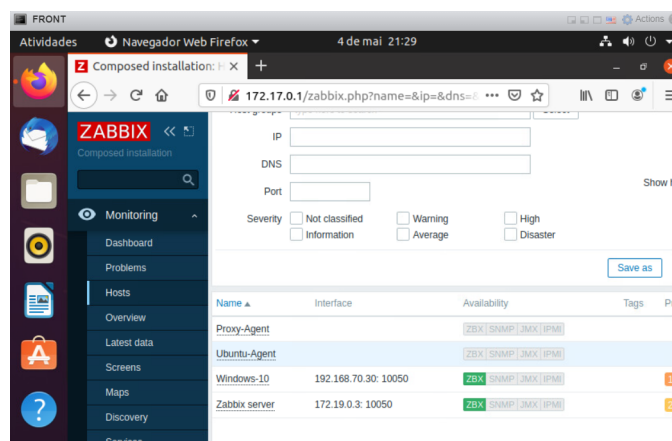


Figura 33 – Imagem do monitoramento do host Windows-10 e Zabbix Server. Fonte: Próprio Autor.

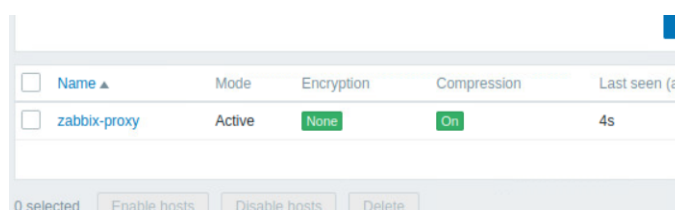


Figura 34 – Imagem que mostra a comunicação entre o Zabbix Proxy e Zabbix Server. Fonte: Próprio Autor.

Além disso, foi necessário realizar a configuração do Zabbix Agent no *host* em que está o Zabbix Proxy, pois o Zabbix Agent é o responsável por coletar os dados do *host* que se tem a intenção de monitorar. A configuração deste item foi feita utilizando a mesma linha de raciocínio em que foi configurado o Agent do Zabbix Server, porém ao invés de colocar na variável de ambiente do Zabbix Agent o ip do Zabbix Server, como os dados serão enviados pelo Zabbix Proxy é necessário colocar o ip do contêiner do Zabbix Proxy, já na parte web será colocado o ip do contêiner em que está o Zabbix Agent.

Além disso, foi configurado em um servidor Windows o Zabbix Agent que irá se comunicar com o Zabbix Proxy. Desta forma, o *host* não será responsável por encaminhar as informações, já que o Zabbix Proxy possui este objetivo. A configuração neste Zabbix Agent é realizada modificando apenas a variável de ambiente “Server” com o ip do *host* em que está rodando o Zabbix Proxy, além de ser necessário apagar as variáveis de ambiente “ServerActive” e “Hostname”.

Contudo, já no ambiente web do Zabbix é preciso colocar o ip do *host* em que está sendo executado este serviço do Zabbix Agent para que haja comunicação e por consequência o *host* seja monitorado.

Sendo assim, o último host monitorado utilizou o arquivo `ZabbixAgentCompose.yaml` para executar o serviço do *Zabbix Agent* e assim, monitorar um host que possui o SO Ubuntu. Além disso, foram realizadas modificações em suas variáveis de ambiente iguais às efetuadas anteriormente no servidor Windows.

Ademais, após ter realizado todas as etapas anteriores, foi feita a validação no *Zabbix Web*, para confirmar que todos os hosts estavam conseguindo se comunicar e enviar dados ao *Zabbix Server*. Evidenciou-se na Figura 35 já que todos os itens criados apresentam um sinal verde, ou seja, o *Zabbix Server* está conseguindo receber dados de todos estes hosts.

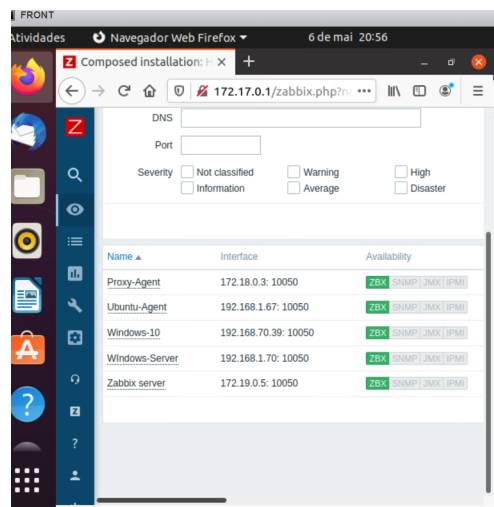


Figura 35 – Imagem que o *Zabbix Server* consegue receber dados de todos estes hosts. Fonte: Próprio Autor.

Dessa maneira, após confirmar que todos os *hosts* conseguem se comunicar e que os dados coletados estão chegando até o *Zabbix Server*, a próxima etapa é configurar quais dados deverão ser monitorados pelo *Zabbix*.

O ambiente de monitoramento *Zabbix*, já possui por *default* vários *templates* prontos com itens a serem monitorados, sendo estes *templates* especificados por sua tecnologia, ou seja, existem *templates* para monitorar itens específicos de sistemas Windows, Linux, SonicWALL e assim por diante, cada um contendo alguma especificação.

Logo, com o objetivo de realizar o monitoramento dos ativos de rede dos hosts utilizados de uma maneira rápida, foi feita a utilização dos *templates* já configurados no ambiente do *Zabbix*. Contudo, o *Zabbix* por ser um *software open source*, ele apresenta infinitas possibilidades de monitoramento de informações dos *hosts*. Na Figura 36 é apresentado alguns itens monitorados do *Zabbix Server*.

Assim sendo, com os devidos itens monitorados, é preciso criar gatilhos que irão alertar o responsável pelo monitoramento sobre algum evento não esperado, porém,

<input type="checkbox"/>	CPU system time system.cpu.util[system]	1m	7d	365d	Zabbix agent	2021-05-01 15:38:12	7.7131 %	+3.2079 %	Graph
<input type="checkbox"/>	CPU user time system.cpu.util[user]	1m	7d	365d	Zabbix agent	2021-05-01 15:38:11	29.08 %	+5.3858 %	Graph
<input type="checkbox"/>	CPU utilization system.cpu.util		7d	365d	Dependent it...	2021-05-01 15:38:13	59.7801 %	+14.3724 %	Graph
<input type="checkbox"/>	Interrupts per second system.cpu.intr	1m	7d	365d	Zabbix agent	2021-05-01 15:38:16	2302.4021	+890.5763	Graph
<input type="checkbox"/>	Load average (1m avg) system.cpu.load[all,avg1]	1m	7d	365d	Zabbix agent	2021-05-01 15:38:10	2.36	+0.64	Graph
<input type="checkbox"/>	Load average (5m avg) system.cpu.load[all,avg5]	1m	7d	365d	Zabbix agent	2021-05-01 15:38:15	1.37	+0.21	Graph
<input type="checkbox"/>	Load average (15m avg) system.cpu.load[all,avg15]	1m	7d	365d	Zabbix agent	2021-05-01 15:38:14	0.99	+0.08	Graph
<input type="checkbox"/>	Number of CPUs system.cpu.num	1m	7d	365d	Zabbix agent	2021-05-01 15:35:51	2		Graph
Zabbix server									
Disk sda (6 items)									
<input type="checkbox"/>	sda: Disk average queue size (avg... vfs.dev.queue_size[sda]		7d	365d	Dependent it...	2021-05-01 15:37:42	0.857		Graph
<input type="checkbox"/>	sda: Disk read rate vfs.dev.read.rate[sda]		7d	365d	Dependent it...	2021-05-01 15:37:42	5.6194 r/s		Graph

Figura 36 – Imagem de itens monitorados do Zabbix Server. Fonte: Próprio Autor.

como está sendo usado *templates default* do Zabbix, estes templates já possuem gatilhos previamente configurados como apresentado na Figura 37.

Severity	Name	Operational data	Expression	Status
Information	Linux generic: by Zabbix agent: /etc/passwd has been changed Depends on: Linux by Zabbix agent: Operating system description has changed Linux by Zabbix agent: System name has changed (new name: {ITEM.VALUE})		{#linux by Zabbix agent: vfs: file: /etc/passwd} < {#CPU} > 0	Enabled
Information	Linux generic: by Zabbix agent: Configured max number of open file descriptors is too low (< {#KERNEL.MAXFILES.MIN})		{#linux by Zabbix agent: kernel: maxfiles: last()} < {#KERNEL.MAXFILES.MIN}	Enabled
Information	Linux generic: by Zabbix agent: Configured max number of processes is too low (< {#KERNEL.MAXPROC.MIN}) Depends on: Linux by Zabbix agent: Getting closer to process limit (over 80% used)		{#linux by Zabbix agent: kernel: maxproc: last()} < {#KERNEL.MAXPROC.MIN}	Enabled
Warning	Linux generic: by Zabbix agent: Getting closer to process limit (over 80% used)	{ITEM.LASTVALUE1} active, {ITEM.LASTVALUE2} limit	{#linux by Zabbix agent: proc: num: last()} < {#linux by Zabbix agent: kernel: maxproc: last()} * 100 > 80	Enabled
Warning	Linux CPU by Zabbix agent: High CPU utilization (over {#CPU.UTIL.CRIT} % for 5m) Depends on: Linux by Zabbix agent: Load average is too high (per CPU load over {#LOAD.AVG.PER_CPU.MAX.WARN} for 5m)	Current utilization: {ITEM.LASTVALUE1}	{#linux by Zabbix agent: system: cpu: util: min(5m)} > {#CPU.UTIL.CRIT}	Enabled

Figura 37 – Imagem de triggers default utilizadas no monitoramento do Zabbix Server. Fonte: Próprio Autor.

Como já foi discutido neste documento que é de suma importância para uma empresa ou instituição realizar o monitoramento de acordo de serviço, este tipo de monitoramento ajuda a saber se um determinado serviço contratado está sendo feito e entregue com a qualidade.

O Zabbix já possui uma funcionalidade que possibilita este tipo de monitoração, chamada de serviços. Nesta aba o usuário pode usar alguma trigger utilizada por algum host monitorado, como apresentado na Figura 38. E a partir desta trigger estabelecer quantos por cento um item ficou ativo, caso ultrapasse a porcentagem desejada o serviço fica na cor vermelha, caso fique dentro da margem permitida o serviço fica verde, como mostra a Figura 39 em que se é monitorado um serviço de inoperância de *Proxy*.

Além do mais, após a incrementação de um sistema de monitoramento, a ferramenta possibilita ações proativas de serviços. Logo, um ambiente monitorado é importante para realização de investimentos corretos no ambiente de TI e imprescindível

The screenshot shows the 'Services' configuration page in Zabbix. It has three tabs: 'Service', 'Dependencies', and 'Time'. The 'Service' tab is active. The configuration fields are as follows:

- Name:** Inatividade do Proxy
- Parent service:** root (with a 'Change' button)
- Status calculation algorithm:** Problem, if at least one child has a problem (dropdown menu)
- Calculate SLA, acceptable SLA (in %):** 99.9 (with a checkbox and input field)
- Trigger:** Proxy-Agent: Zabbix agent is not available (for 3m) (with a 'Select' button)
- Sort order (0->999):** 0

At the bottom, there are three buttons: 'Update', 'Delete', and 'Cancel'.

Figura 38 – Imagem da configuração de serviços no Zabbix. Fonte: Próprio Autor.

The screenshot shows the 'Services' monitoring page in Zabbix. It has a 'Period' dropdown set to 'Last 7 days'. The table below shows the status of the 'root' service.

Service	Status	Reason	Problem time	SLA / Acceptable SLA
root				
Inatividade do Proxy - Zabbix agent is not available (for 3m)	OK			0.0000 100.0000 / 99.9000

Figura 39 – - Imagem da utilização de serviços no Zabbix. Fonte: Próprio Autor.

para manter *softwares* e *hardwares* altamente disponíveis, sem qualquer downtime nos serviços críticos e de negócios da empresa ou instituição.

CAPÍTULO 5

CONCLUSÕES

O principal objetivo deste trabalho resumiu-se em implantar e realizar o monitoramento de um ambiente computacional, utilizando Zabbix em contêineres Docker. Sendo a tecnologia Docker responsável por fornecer aplicações relacionadas a contêineres, e a utilização do *software* de código aberto Zabbix tem o intuito de não somente monitorar e controlar serviços computacionais, mas também para realizar soluções e antecipações de problemas relacionados a aplicações e equipamentos de rede. Após toda abordagem do relatório e através dos estudos e pesquisas realizadas, pode-se concluir que os resultados alcançados foram satisfatórios e favoráveis.

Foi possível alcançar todos os objetivos específicos para este Trabalho, destacando a ferramenta Docker que efetivamente demonstrou sua eficácia ao facilitar a criação e o gerenciamento de imagens de aplicações Zabbix, além de prover flexibilidade e velocidade para instalação de microsserviços em contêineres executados no servidor.

Para se alcançar o objetivo final, se fez necessário um estudo de caso com simulação de um ambiente computacional, contendo variados Sistemas Operacionais e ativos de redes que poderam ser monitorados, com o propósito de validar as soluções adotadas de realizar instalação de serviços do Zabbix por meio do Docker, e a partir disso analisar suas principais propriedades.

Já na tabela comparativa apresentada no capítulo de Materiais e Métodos, foi comprovado que o Zabbix é a solução mais completa do mercado para realizar o monitoramento de ativos de rede em um ambiente computacional. Desta forma, no resultado da primeira etapa do estudo de caso a tecnologia Docker se apresentou rápida e eficaz quando se trata de executar contêineres, além de possuir outras funcionalidades como o Docker Hub para alocar imagens de serviços.

No resultado da segunda etapa do estudo de caso, a tecnologia Docker-compose, se confirmou uma ferramenta para definir e executar aplicativos Docker de vários contêineres. Com o Compose, é possível utilizar um arquivo YAML para configurar os serviços de algum aplicativo.

No resultado da terceira etapa do estudo de caso, foi feita a convergência de tecnologias e conhecimentos adquiridos ao longo do projeto, tendo como objetivo criar um arquivo YAML de própria autoria, que faça o resgate de imagens de serviços no Docker Hub e execute todas aplicações necessárias para criar um sistema completo de monitoramento. Ainda, pontuando ações realizadas neste subcapítulo, foi feita a apresentação de *templates* que proporcionam o monitoramento de ativos de rede no Zabbix, além de ter evidenciado que a solução de monitoração possibilita a criação de níveis de SLAs.

Logo concluiu-se que o Docker é eficiente para subir diversos serviços de maneira rápida e flexível, além de não exigir tanto de requisitos de *hardware* como memória e processador, tornando-se assim uma alternativa para instalações de aplicações não só de serviços de monitoramento mas de diferentes tipos de *softwares*. Ademais, a tecnologia Zabbix se mostrou capaz de realizar o monitoramento de diversos ativos de rede em diferentes *hosts*, possibilitando gerar para empresas e instituições uma visão da infraestrutura monitorada de baixo e alto nível.

Tal qual, as duas tecnologias funcionaram como previsto possibilitando: implantar um cenário de monitoramento, utilizando contêineres Docker, com dependência de algumas aplicações externas; avaliar e homologar a implementação de um ambiente computacional gerenciado por Zabbix sobre contêineres Docker em uma página modelo; e homologar uma estrutura em Docker que permita implementar a ferramenta de monitoramento de rede Zabbix, de forma customizada.

Além disso, por se tratar de uma área ampla no setor de infraestrutura de redes, a instalação do ambiente junto a outras tecnologias exigiu o domínio de algumas áreas específicas, entre elas: redes de computadores, virtualização, contêineres, monitoramento de ativos de rede e protocolos de comunicação entre redes de computadores.

Dessa maneira, foi necessário fazer um aprofundamento destas tecnologias citadas para que se pudesse ter um embasamento teórico mínimo ao começar a implantação

nos casos de uso. Durante a implementação houve um tempo para que se pudesse entender o funcionamento de contêineres Docker e como a comunicação entre os contêineres acontecia na rede para que os serviços do Zabbix se integrassem e comunicassem entre si, possibilitando a troca de informações computacionais e posteriormente conseguindo realizar o monitoramento de todos os *hosts* citados.

Portanto, sugere-se como trabalhos futuros:

- Implementar a tecnologia Zabbix em um sistema Cloud AWS, utilizando alguma ferramenta de orquestração de contêineres para executar micro serviços em diversos hosts;
- Realizar testes de alta disponibilidade, principalmente no banco de dados com ambiente de produção;
- Realizar uma análise comparativa de tecnologias responsáveis por fazer a orquestração de contêineres, como: Docker Swarm e Kubernetes;
- Fazer uma análise comparativa de desempenho entre banco de dados MYSQL e Postgresql, ambos utilizados pela tecnologia Zabbix;
- Fazer uma análise comparativa de desempenho entre serviços web como Nginx e Apache, ambos utilizados pela tecnologia Zabbix;
- Testes de escalabilidade com o cluster Kubernetes, testando a escalabilidade manual e a escalabilidade automática por meio da propriedade do *autoscaling*, onde as empresas que não tenham previsibilidade da quantidade de acessos a seus serviços podem se beneficiar, alocando e desalocando horizontalmente recursos de acordo com a demanda;
- Apresentar de maneira enfática os benefícios e malefícios de serviços Cloud principalmente da AWS, pontuando de forma prática e dissertativa partes centrais de serviços da plataforma como: EC2, S3, AWS RDS, entre outros.

REFERÊNCIAS

- ALECRIM, E. O que é virtualização e para que serve? *Infoweste*, 2013. 9
- ALVES, L. C. D. *O impacto da virtualização no desempenho de aplicações distribuídas baseadas em SOA e a sua influência nos modelos de desempenho*. Tese (Doutorado) — USP, São Paulo-SP, 2013. 7
- AQUASEC. Docker container. *Aqua Cloud Native Academy*, 2020. viii, 13
- CARISSIMI, A. *Virtualização: da teoria a soluções*. Tese (Doutorado) — UFRJ, Rio de Janeiro-RJ, 2008. 8
- CASTRO, D. O. de S.; CARVALHO, F. C. de; MENDES, L. N. *Ambiente de Rede Monitorado com CACTI, ZABBIX e THE DUDE*. Tese (Doutorado) — UFJF, Juiz de Fora-MG, 2013. viii, 18, 19, 20, 23, 25, 27
- CAVALHEIRO, A. Monitoramento – smi, snmp e mib. *Blog do Alexandre Cavalheiro*, 2020. viii, 23
- DOCKER. Docker overview. *Docker*, 2020. viii, 12, 13, 15
- DPSTELECOM. Snmp tutorial - what is it and how does it work? *DPSTelecom*, 2001. viii, 22
- FILHO, A. C. de A. *Estudo comparativo entre DockerSwarm e Kubernetes para orquestração de contêineres em arquiteturas de software com microserviços*. Tese (Doutorado) — UFPE, Recife-PB, 2016. 3
- FILHO, L. C. da S.; PEREIRA, R. B. de O. Alta disponibilidade em containers docker por meio do docker swarm. *Profiscientia*, 2018. 2, 8
- FILHO, N. A. P. *Serviços de Pertinência para Clusters de Alta Disponibilidade*. Tese (Doutorado) — USP, São Paulo-SP, 2004. 3

- HAT, R. Advantages of using docker. *RED HAT*, 2017. 11
- KOCH, F. L.; WESTPHALL, C. B. *Decentralized Network Management using Distributed Artificial Intelligence*. Dissertação (Mestrado) — Plenum Publishing Corporation, USA, 2001. 20
- KUROSE, J. F.; ROSS, K. W. *Redes de computadores e a Internet - uma abordagem top-down*. Quinta edição. [S.l.: s.n.], 2010. 19, 20
- LEITE, P. A. D. S. *RELATÓRIO DE ESTÁGIO SUPERVISIONADO ALTA DISPONIBILIDADE PARA MICRO-SERVIÇOS EM CONTAINERS DOCKER, ORQUESTRADOS POR KUBERNETES*. Tese (Doutorado) — UFMT, Cuiabá-MT, 2019. 10
- LESSA, D. O protocolo de gerenciamento rmon. *Boletim bimestral sobre tecnologia de redes produzido e publicado pela RNP*, 1999. 26
- LIVRE, E. Virtualização. *Escotilha Livre*, 2015. viii, 7
- MARKONETOOLS. Java virtual machine. *MarkOneTools*, 2019. viii, 10
- MAURO, D.; SCHMIDT, K. *Redes e Sistemas de Comunicação de Dados*. Segunda edição. [S.l.: s.n.], 2001. 23
- MENEZES, D. *Intel Virtualization Technology e Intel Trusted Execution Technology*. Tese (Doutorado) — UFRJ, Rio de Janeiro-SP, 2008. 7
- PEREIRA, S. Virtualização de sistemas operacionais. In: . Paranaíba-PR: [s.n.], 2004. 16
- QUORA. Em virtualização, o que quer dizer o termo "bare metal"? *Quora*, 2020. viii, 9
- SANTOS, J. B. M. D. Relatório de estágio supervisionado serviço web de armazenamento e compartilhamento de arquivos com autenticação centralizada, em containers, para gmrl/ufmt. In: . Cuiabá-MT: [s.n.], 2018. 7, 9
- SANT'ANA, J. Consumidor compra mais e gasta menos na black friday. *Gazeta do Povo*, 2017. 1
- STALLINGS, W. *Redes e Sistemas de Comunicação de Dados*. Quinta edição. [S.l.: s.n.], 2005. 19
- TELECO. Gerenciamento e monitoramento de rede i: Teoria de gestão de redes. *TELECO*, 2012. 18, 19, 28
- VERAS, M. Componente central do datacenter. *Microsoft Corporation*, 2018. 8