

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
GOIANO - CAMPUS MORRINHOS  
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS PARA  
*INTERNET***

**MATEUS AUGUSTO DE SOUZA PEREIRA**

**AMBIENTE DE ALTA DISPONIBILIDADE PARA BANCO DE DADOS  
UTILIZANDO *MYSQL* EM *CONTAINERS* ORQUESTRADOS PELO  
*KUBERNETES***

**MORRINHOS - GO  
2020**

**MATEUS AUGUSTO DE SOUZA PEREIRA**

**AMBIENTE DE ALTA DISPONIBILIDADE PARA BANCO DE DADOS  
UTILIZANDO *MYSQL* EM *CONTAINERS* ORQUESTRADOS PELO  
*KUBERNETES***

Trabalho de curso apresentado ao Curso Superior de Tecnologia em Sistemas para *Internet* do Instituto Federal Goiano - Campus Morrinhos, como requisito parcial para obtenção de título de Tecnólogo em Sistemas para *Internet*.

**Área de concentração:** rede de computadores e banco de dados.

**Orientadora:** MSc. Ana Maria Martins Carvalho.

**MORRINHOS - GO**

**2020**

Sistema desenvolvido pelo ICMC/USP  
Dados Internacionais de Catalogação na Publicação (CIP)  
**Sistema Integrado de Bibliotecas - Instituto Federal Goiano**

SP436a Souza Pereira, Mateus Augusto de  
AMBIENTE DE ALTA DISPONIBILIDADE PARA BANCO DE  
DADOS UTILIZANDO MYSQL EM CONTAINERS ORQUESTRADOS  
PELO KUBERNETES / Mateus Augusto de Souza  
Pereira;orientadora Ana Maria Martins Carvalho. --  
Morrinhos, 2020.  
96 p.

Dissertação ( em CURSO SUPERIOR DE TECNOLOGIA EM  
SISTEMAS PARA INTERNET) -- Instituto Federal Goiano,  
Campus Morrinhos, 2020.

1. Alta disponibilidade. 2. Contêiner. 3. Banco  
de dados. 4. Rede de computadores. I. Martins  
Carvalho, Ana Maria , orient. II. Título.

**TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR PRODUÇÕES  
TÉCNICO-CIENTÍFICAS NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO**

Com base no disposto na Lei Federal nº 9.610/98, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano, a disponibilizar gratuitamente o documento no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

**Identificação da Produção Técnico-Científica**

- |  |   |
|--|---|
| <input type="checkbox"/> Tese                          | <input type="checkbox"/> Artigo Científico              |
| <input type="checkbox"/> Dissertação                   | <input type="checkbox"/> Capítulo de Livro              |
| <input type="checkbox"/> Monografia – Especialização   | <input type="checkbox"/> Livro                          |
| <input checked="" type="checkbox"/> TCC - Graduação    | <input type="checkbox"/> Trabalho Apresentado em Evento |
| <input type="checkbox"/> Produto Técnico e Educacional | - Tipo:   |

Nome Completo do Autor: Mateus Augusto de Souza Pereira

Matrícula: 2015104211710071

Título do Trabalho: AMBIENTE DE ALTA DISPONIBILIDADE PARA BANCO DE DADOS UTILIZANDO MYSQL EM CONTAINERS ORQUESTRADOS PELO KUBERNETES

**Restrições de Acesso ao Documento**

Documento confidencial:  Não  Sim, justifique: \_\_\_\_\_

Informe a data que poderá ser disponibilizado no RIIF Goiano: 01/05/2020

O documento está sujeito a registro de patente?  Sim  Não

O documento pode vir a ser publicado como livro?  Sim  Não

**DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA**

O/A referido/a autor/a declara que:

- o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;
- obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;
- cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

Morrinhos, 18/03/2020.

Local Data

*Mateus Augusto de S. Pereira*

Assinatura do Autor e/ou Detentor dos Direitos Autorais

Ciente e de acordo:



Assinatura do(a) orientador(a)

**MATEUS AUGUSTO DE SOUZA PEREIRA**

**AMBIENTE DE ALTA DISPONIBILIDADE PARA BANCO DE DADOS  
UTILIZANDO *MYSQL* EM *CONTAINERS* ORQUESTRADOS PELO  
*KUBERNETES***

Data da defesa: 10 de março de 2020.

Resultado: APROVADO.

**BANCA EXAMINADORA**

**ASSINATURAS**

Ana Maria Martins Carvalho  
Instituto Federal Goiano - Campus Morrinhos

Prof<sup>a</sup> MSc. \_\_\_\_\_



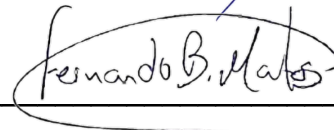
Antônio Neco de Oliveira  
Instituto Federal Goiano - Campus Morrinhos

Prof<sup>o</sup> Dr. \_\_\_\_\_



Fernando Barbosa Matos  
Instituto Federal Goiano - Campus Morrinhos

Prof<sup>o</sup> Dr. \_\_\_\_\_



**MORRINHOS - GO**

**2020**

## **DEDICATÓRIA**

Dedico esse trabalho de conclusão de curso a toda minha família e amigos, em especial minha mãe e minha orientadora as quais foram os grandes pilares para que tornasse possível a realização deste trabalho.

## **AGRADECIMENTOS**

Agradeço primeiramente a Deus por ter me guardado em toda minha trajetória, aos meus familiares, por sempre ter acreditado no meu potencial e à eSolution Tecnologia por ter me transformado no profissional que sou hoje, deixo aqui um agradecimento especial a todo Instituto Federal Goiano - Campus Morrinhos por toda a oportunidade que aqui me deram, e principalmente a meus amigos que tornaram esse caminho um capítulo inesquecível em minha vida.

## RESUMO

Em um mundo 24 horas conectado, uma falha em um sistema pode levar a falência de uma empresa, por essa razão a alta disponibilidade é um fator de extrema importância para uma empresa, entretanto muitas empresas aplicam a alta disponibilidade nas ferramentas e deixam de lado o banco de dados, o que pode resultar em uma grande catástrofe em caso de falhas. Este trabalho tem como objetivo elaborar um ambiente de alta disponibilidade em banco de dados utilizando contêineres orquestrados pela ferramenta *Kubernetes* em uma nuvem computacional. Para o trabalho, realizou-se a criação de uma arquitetura utilizando o banco de dados *MySQL* e ferramentas *open-source* que auxiliam na implementação de um grupo de replicação de banco de dados *master-slave* no *cluster Kubernetes*, a fim de obter alta disponibilidade, tolerância a falhas e alta escalabilidade, toda a arquitetura foi exposta na nuvem *Google Cloud*. Com a arquitetura montada, realizou-se testes de falha para verificar quanto a estrutura conseguiria se manter altamente disponível e consistente, mesmo em casos de falha, também foi analisado a escalabilidade da aplicação, alterando o tamanho da arquitetura de forma a atender as novas requisições sem a construção de um novo *cluster*. Como resultados, espera-se que através do *Kubernetes* seja possível alcançar uma arquitetura de alta disponibilidade totalmente escalável e tolerante a falhas no banco de dados *MySQL*.

**Palavras chave:** Alta disponibilidade. Contêiner. Banco de dados. Rede de computadores.



## **ABSTRACT**

*In a 24-hour connected world, a system failure can lead to the bankruptcy of a company, which is why high availability is an extremely important factor for a company, however many companies apply high availability in tools and leave aside databases, which can result in a major catastrophe in the event of failures. This work aims to develop a high availability database environment using containers orchestrated by the Kubernetes tool in a computational cloud. For the work, an architecture was created using the MySQL database and open-source tools that assist in the implementation of a master-slave database replication group in the Kubernetes cluster, in order to obtain high availability, fault tolerance and high scalability, the entire architecture was exposed in the Google Cloud. With the assembled architecture, failure tests were performed to verify how much the structure would be able to remain highly available and consistent, even in cases of failure, the scalability of the application was also analyzed, changing the size of the architecture in order to meet the new requests without construction of a new cluster. As a result, it is expected that through Kubernetes it will be possible to achieve a fully scalable and fault tolerant high availability architecture in the MySQL database.*

**Keywords:** *High availability. Containers. Database. Computer network.*

## Lista de abreviações

ACID - *Atomicity - Consistency - Isolation - Durability*

API - *Application Programming Interface*

ASA - *Api Server Aggregation*

CLI - *Command Line Interface*

COS - *Container-Optimized OS*

CPU - *Central Process Unit*

CRD - *Custom Resources Definition*

DDC - *Docker Datacenter*

GCS - *Group Communication System*

GPL - *General Public License*

HA - *High Availability*

IaaS - *Infrastructure as a Service*

IP - *Internet Protocol*

ISO - *International Organization for Standardization*

LVS - *Linux Virtual Server*

OS - *Operating System*

PaaS - *Plataform as a Service*

REST - *Representational State Transfer*

SaaS - *Software as a Service*

SGBD - *Sistema de gerenciamento de banco de dados*

SPOF - *Single Point of Failure*

SQL - *Structured Query Language*

UID - *User identifier*

## Lista de Ilustrações

Figura 1: Relação entre falha, erro e defeito.....	10.
Figura 2: Balanceamento de Carga dentro de uma rede de servidores.....	11.
Figura 3: Nuvem pública.....	14.
Figura 4: Nuvem privada.....	14.
Figura 5: Replicação <i>MySQL master-slave</i> .....	17.
Figura 6: Replicação de Grupo.....	18.
Figura 7: Eleição de novo primário após <i>failover</i> .....	19.
Figura 8: Ocorrência de <i>failover</i> em estrutura multiprimário.....	20.
Figura 9: Arquitetura de um <i>container</i> de <i>software</i> .....	23.
Figura 10: Imagem ilustrativa da estrutura geral do <i>Docker</i> .....	24.
Figura 11: Arquitetura <i>Docker Engine</i> .....	26.
Figura 12: <i>Cluster Kubernetes</i> .....	29.
Figura 13: Arquitetura do grupo (Três Servidores - Estrutura sem dados).....	39.
Figura 14: Servidor em recuperação após falha (Três Servidores - Estrutura sem dados).....	39.
Figura 15: Tempo gasto para eleição de novo mestre e execução de operação (Três Servidores - Estrutura sem dados).....	40.
Figura 16: Grupo após recuperação de <i>failover</i> (Três Servidores - Estrutura sem dados).....	40.
Figura 17: Servidores antes da ocorrência de <i>failover</i> (Três Servidores - Dados com 47.2MB).....	40.
Figura 18: <i>Status</i> do grupo no momento do <i>failover</i> (Três Servidores - Dados com 47.2MB).....	41.
Figura 19: Ocorrência do <i>failover</i> (Três Servidores - Dados com 47.2MB).....	41.
Figura 20: <i>Status</i> do grupo após recuperação de falha (Três Servidores - Dados com 47.2MB).....	42.
Figura 21: Consulta no momento do <i>failover</i> (Três servidores - Dados com 110.2MB).....	42.

Figura 22: Ocorrência de falha no grupo (Três servidores - Dados com 110.2MB).....	43.
Figura 23: Consistência em servidor recuperado após <i>failover</i> (Três servidores - Dados com 110.2MB).....	43.
Figura 24: Grupo de replicação (Cinco servidores - Estrutura sem dados).....	44.
Figura 25: Derrubando os servidores primários (Cinco servidores - Estrutura sem dados).....	45.
Figura 26: <i>Status</i> do grupo na primeira falha (Cinco servidores - Estrutura sem dados).....	45.
Figura 27: <i>Status</i> do grupo na segunda falha (Cinco servidores - Estrutura sem dados).....	45.
Figura 28: Recuperação após <i>failover</i> (Cinco servidores - Estrutura sem dados)....	46.
Figura 29: <i>Status</i> do grupo de replicação (Cinco servidores - Dados com 47.2MB).....	47.
Figura 30: Primeiro <i>failover</i> (Cinco servidores - Dados com 47.2MB).....	47.
Figura 31: Segundo <i>failover</i> (Cinco servidores -Dados com 47.2MB).....	47.
Figura 32: Status do grupo após recuperação de falha (Cinco servidores - Dados com 47.2MB).....	48.
Figura 33: <i>Status</i> da arquitetura antes da ocorrência das falhas (Cinco servidores - Dados com 110.2MB).....	48.
Figura 34: Ocorrência da falha e <i>failover</i> bem sucedido (Cinco servidores - Dados com 110.2MB).....	48.
Figura 35: <i>Status</i> do grupo no momento da primeira falha (Cinco servidores - Dados com 110.2MB).....	49.
Figura 36: Ocorrência de falha e <i>failover</i> bem sucedido(Cinco servidores - Dados com 110.2MB).....	49.
Figura 37: <i>Status</i> do servidor no momento da segunda falha (Cinco servidores - Dados com 110.2MB).....	50.
Figura 38: Grupo recuperado após testes de <i>failover</i> (Cinco servidores - Dados com 110.2MB).....	50.
Figura 39: Estrutura inicial com sete servidores (Sete servidores - Estrutura sem dados).....	51.

Figura 40: Primeiro <i>failover</i> na arquitetura (Sete servidores - Estrutura sem dados).....	51.
Figura 41: Quantidade de dados inseridos antes da segunda falha (Sete servidores - Estrutura sem dados).....	52.
Figura 42: Segunda falha e <i>failover</i> bem sucedido (Sete servidores - Estrutura sem dados).....	52.
Figura 43: Quantidade de dados armazenados antes da terceira falha ocorrer (Sete servidores - Estrutura sem dados).....	53.
Figura 44: Terceira falha e <i>failover</i> bem sucedido (Sete servidores - Estrutura sem dados).....	53.
Figura 45: Quantidade total de dados inseridos mesmo com o servidor em falha (Sete servidores - Estrutura sem dados).....	53.
Figura 46: Grupo de replicação antes dos testes de <i>failover</i> (Sete servidores - Dados com 47.2MB).....	54.
Figura 47: <i>Status</i> do grupo na primeira execução do <i>failover</i> (Sete servidores - Dados com 47.2MB).....	55.
Figura 48: Tempo para ocorrência do <i>failover</i> (Sete servidores - Dados com 47.2MB).....	55.
Figura 49: Momento da ocorrência da falha e início do <i>failover</i> (Sete servidores - Dados com 47.2MB).....	55.
Figura 50: <i>Status</i> do grupo no momento da ocorrência da segunda falha (Sete servidores - Dados com 47.2MB).....	56.
Figura 51: Terceiro <i>failover</i> realizado com sucesso (Sete servidores - Dados com 47.2MB).....	56.
Figura 52: <i>Status</i> do grupo no momento da terceira execução do <i>failover</i> (Sete servidores - Dados com 47.2MB).....	56.
Figura 53: Arquitetura totalmente recuperada após ocorrência de falha (Sete servidores - Dados com 47.2MB).....	57.
Figura 54: Grupo antes dos testes de <i>failover</i> (Sete servidores - Dados com 110.2MB).....	57.
Figura 55: Servidor em processo de <i>failover</i> (Sete servidores - Dados com 110.2MB).....	58.

Figura 56: <i>Status</i> do servidor no momento da primeira falha (Sete servidores - Dados com 110.2MB).....	58.
Figura 57: Processo de <i>failover</i> em execução (Sete servidores - Dados com 110.2MB).....	59.
Figura 58: <i>Status</i> do grupo no momento da segunda falha (Sete servidores - Dados com 110.2MB).....	59.
Figura 59: Tempo para execução do processo de <i>failover</i> (Sete servidores - Dados com 110.2MB).....	59.
Figura 60: <i>Status</i> do grupo no momento da terceira falha (Sete servidores - Dados com 110.2MB).....	60.
Figura 61: Primeiro servidor em falha totalmente recuperado (Sete servidores - Dados com 110.2MB).....	60.
Figura 62: Segundo servidor em falha totalmente recuperado (Sete servidores - Dados com 110.2MB).....	61.
Figura 63: Terceiro servidor em falha totalmente recuperado (Sete servidores - Dados com 110.2MB).....	61.
Figura 64: Escalando estrutura de servidores.....	62.
Figura 65: Criação do segundo arquivo de configuração do <i>ProxySQL</i> .....	63.
Figura 66: Atualização da arquitetura do <i>ProxySQL</i> para atender a nova arquitetura.....	63.
Figura 67: Estrutura <i>ProxySQL</i> .....	78.

## Lista de Tabelas

Tabela 1: Níveis de disponibilidade.....	07.
Tabela 2: Número de instâncias por ocorrência de falhas.....	22.

## Lista de Quadros

Quadro 1: Comandos <i>dockerfile</i> .....	27.
Quadro 2: Representação de arquivo <i>YAML</i> .....	34.
Quadro 3: Relação de tempo de <i>failover</i> por arquitetura.....	64.
Quadro 4: Relação de economia de tempo e valor monetário comparado com a arquitetura proposta.....	65.
Quadro 5: Arquivo de configuração para replicação de grupo <i>MySQL</i> .....	80.



# SUMÁRIO

<b>1. INTRODUÇÃO</b>	<b>1</b>
1.1 OBJETIVO GERAL	1
1.2 OBJETIVOS ESPECÍFICOS	2
1.3 ORGANIZAÇÃO DO TRABALHO	2
<b>2. TRABALHOS CORRELATOS</b>	<b>4</b>
<b>3. REFERENCIAL TEÓRICO</b>	<b>6</b>
3.1 ALTA DISPONIBILIDADE	6
<b>3.1.1 Escalabilidade</b>	<b>8</b>
<b>3.1.2 Cluster</b>	<b>8</b>
<b>3.1.3 Tolerância a falhas</b>	<b>9</b>
<b>3.1.4 Balanceamento de carga</b>	<b>11</b>
3.2 COMPUTAÇÃO EM NUVEM	12
3.3 BANCO DE DADOS	15
3.4 A ALTA DISPONIBILIDADE COM <i>MYSQL</i>	16
<b>3.4.1 Replicação <i>master-slave</i> no <i>MySQL</i></b>	<b>17</b>
<b>3.4.2 Replicação de grupo no <i>MySQL</i></b>	<b>17</b>
3.4.2.1 Modo primário	19
3.4.2.2 Multiprimário	20
<b>3.4.3 Estrutura de replicação</b>	<b>21</b>
3.5 <i>CONTAINERS DE SOFTWARE</i>	22
<b>3.5.1 Docker</b>	<b>24</b>
3.5.1.1 Camada de sistema operacional	25
3.5.1.1.1 <i>Namespace</i>	25
3.5.1.1.2 Grupos de Controle - <i>CGroups</i>	25
3.5.1.1.3 <i>Layer Capabilities</i>	25
3.5.1.2 Arquitetura <i>Docker</i>	25
3.5.1.2.1 <i>Docker Engine</i>	25
3.5.1.2.2 <i>Imagens Docker</i>	26
3.5.1.2.3 <i>Container Docker</i>	26
3.5.1.2.4 <i>Dockerfile</i>	27
3.5.1.3 Manipuladores de <i>contêineres</i>	28
3.5.1.3.1 <i>Registros Docker</i>	28
3.5.1.3.2 <i>Docker Compose</i>	28
3.6 <i>KUBERNETES</i>	28
<b>3.6.1 Nó mestre</b>	<b>29</b>
<b>3.6.2 Nó Trabalhador</b>	<b>30</b>
<b>3.6.3 Conceitos gerais</b>	<b>30</b>
3.6.3.1 <i>Node</i>	31

3.6.3.2	<i>Pod</i>	31
3.6.3.3	<i>Controller ReplicaSet</i>	31
3.6.3.4	<i>Controller StatefulSet</i>	32
3.6.3.5	<i>PersistentVolums</i>	32
3.6.3.6	<i>Storage Class</i>	32
3.6.3.7	<i>Kubectl</i>	32
3.6.3.8	<i>Configmap</i>	33
3.6.3.9	<i>YAML</i>	33
3.6.3.10	<i>Service</i>	35
3.6.3.11	<i>Deployment</i>	35
3.6.3.12	<i>Custom Resources</i>	36
<b>4.</b>	<b>METODOLOGIA E EXPERIMENTO</b>	<b>37</b>
4.1	EXPERIMENTO REALIZADO	38
4.1.1	<b>Arquitetura com três servidores</b>	<b>39</b>
4.1.1.1	Grupo de replicação sem dados	39
4.1.1.2	Grupo de replicação com tamanho de 47.2MB	40
4.1.1.3	Grupo de replicação com tamanho de 110.2MB	42
4.1.2	<b>Arquitetura com cinco servidores</b>	<b>44</b>
4.1.2.1	Grupo de replicação sem dados	44
4.1.2.2	Grupo de replicação com tamanho de 47.2MB	46
4.1.2.3	Grupo de replicação com tamanho de 110.2MB	47
4.1.3	<b>Arquitetura com sete servidores</b>	<b>50</b>
4.1.3.1	Grupo de replicação sem dados	50
4.1.3.2	Grupo de replicação com tamanho de 47.2MB	53
4.1.3.3	Grupo de replicação com tamanho de 110.2MB	56
4.2	A ESCALABILIDADE NOS TESTES	61
<b>5.</b>	<b>RESULTADOS</b>	<b>63</b>
<b>6.</b>	<b>CONSIDERAÇÕES FINAIS</b>	<b>65</b>
	<b>REFERÊNCIAS</b>	<b>67</b>
	<b>APÊNDICES A - PROXYSQL</b>	<b>75</b>
	<b>APÊNDICES B - MYSQL NO KUBEDB</b>	<b>77</b>

## 1. INTRODUÇÃO

Com a ascensão da era tecnológica, cada vez mais empresas se vêm obrigadas a manter suas marcas na *Internet*. A cada dia o mundo mostra estar 24 horas conectado a rede de computadores, e um minuto fora do ar em uma empresa conectada a *Internet* pode trazer grandes prejuízos financeiros, físicos ou morais à organização, podendo até mesmo levar à falência (NEWS.COMSCHOOL, 2015).

Em 2008, 90 minutos *offline* causaram à *Amazon* um prejuízo de US\$ 2,79 milhões, segundo o *site* Globo.com (G1.COM, 2008).

Em 2013, o Canaltech informou que para os varejistas um minuto fora do ar pode levar US\$ 8 mil em prejuízos durante a *Black Friday* (CANALTECH.COM, 2013).

Conforme a *Psafe* um período de 20 minutos *offline* causou ao *Facebook* um prejuízo total de US\$ 500 milhões (NOVAES, 2014).

No ano de 2015 um estudo realizado pela *Google* Brasil, retratou que a cada um minuto fora do ar, R\$ 1,5 milhões são perdidos em consumo para empresas de varejo na *Black Friday* (CIRCUITOMT.COM, 2015).

De acordo com *Ecommerce* Brasil um estudo levantado pela *Sofist*, lojas virtuais perdem ao menos R\$ 3,1 milhões em quatro horas por instabilidade em seus *sites* durante a *Black Friday* (E-COMMERCEBRASIL, 2018).

Assim sendo, para amenizar essa situação, as empresas têm investido fortemente em tecnologias de alta disponibilidade, buscando manter seus sistemas a maior parte do tempo possível *online* e operacionais.

No entanto, tais estruturas, geralmente, são realizadas somente na aplicação, deixando de lado o banco de dados, devido a sua complexidade de implementação.

### 1.1 OBJETIVO GERAL

O objetivo deste trabalho é apresentar uma arquitetura de alta disponibilidade para banco de dados, sendo esta, de fácil implementação e dimensionamento utilizando *containers* orquestrados pela ferramenta *Kubernetes*, onde é apresentado

a viabilidade dessa tecnologia para serviços de missão crítica, ou seja, serviços que precisam estar sempre operacionais no momento solicitado.

Dentre as soluções de banco de dados disponíveis, será avaliado neste trabalho o banco *MySQL*, onde será implementado um grupo de replicação com apenas um servidor de escrita e outros dois de leitura, analisando sua estrutura quanto a consistência e integridade após a ocorrência de falhas dentro da arquitetura a ser montada.

Com exceção da nuvem escolhida para os testes, todas as ferramentas usadas são *open source*, o que garante na arquitetura a descentralização de ferramentas, podendo ser implementada com as configurações desejáveis e em qualquer servidor que contemple os requisitos necessários.

A contribuição pretendida deste trabalho é apresentar uma nova visão referente a alta disponibilidade para banco de dados, demonstrando as vantagens de se trabalhar com ferramentas de containerização em nuvens computacionais.

## 1.2 OBJETIVOS ESPECÍFICOS

A partir dos objetivos gerais, podem-se descrever os seguintes objetivos específicos:

- Apresentar os conceitos de alta disponibilidade;
- Apresentar a ferramenta *Kubernetes*;
- Comprovar que com a ajuda do *Kubernetes* é possível obter uma estrutura altamente disponível em banco de dados;
- Obter uma estrutura de fácil implementação e altamente escalável;
- Comprovar que é possível utilizar banco de dados relacionais em ambientes containerizados.

## 1.3 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está organizado em capítulos que abordam os principais temas, distribuídos da seguinte maneira:

No Capítulo 2 são abordados os principais trabalhos que servirão como base para o desenvolvimento do pensamento técnico para a realização dos objetivos que norteiam esse trabalho;

No Capítulo 3 é apresentado o referencial teórico, abordando os principais conceitos e técnicas que servirão como base para o melhor entendimento do trabalho apresentado;

No Capítulo 4 são abordados a metodologia e o experimento realizados no trabalho, onde será apresentada a arquitetura de alta disponibilidade e os testes realizados;

No Capítulo 5 são apresentados os resultados obtidos no experimento realizado no Capítulo 4, comparando os resultados com os objetivos pretendidos;

No Capítulo 6 expõe as considerações finais em relação ao estudo, e apresenta a proposta de trabalhos futuros.

## 2. TRABALHOS CORRELATOS

Neste capítulo é apresentado um levantamento da bibliografia correlata, listando os principais trabalhos que foram norteadores para o desenvolvimento desse trabalho, onde serão analisados as contribuições de cada trabalho. Tais informações auxiliam a definir de forma sucinta, a contribuição real desse trabalho no meio acadêmico.

Gregol (2011) apresenta um estudo demonstrando o uso de recursos para escalar uma aplicação *Web* e obter alta disponibilidade, abordando de forma teórica e prática alguns conceitos e ferramentas para auxiliar na obtenção de sistemas altamente escaláveis.

Bruschi (2014) apresenta uma solução de baixo custo para alta disponibilidade em banco de dados, utilizando LVS (*Linux Virtual Server*), tal trabalho levanta os pontos principais de uma estrutura de alta disponibilidade e conclui que mesmo com ferramentas de baixo custo, pode-se obter resultados satisfatórios quanto à alta disponibilidade.

Freitas (2014) elabora um modelo para garantia de consistência para banco de dados em nuvem, apresentando a relação entre disponibilidade, consistência e tolerância a falhas, para banco de dados distribuídos, conclui-se nesse trabalho a possibilidade de modelos que garantam formas de consistência, alta disponibilidade e tolerância a falhas, mesmo que de forma eventual.

Albuquerque Filho (2016) traz um estudo comparativo entre plataformas de orquestração de *containers* em arquiteturas de microsserviços, analisando as ferramentas *Kubernetes* e *Docker Swarm*, o estudo conclui que o *Kubernetes* leva vantagem quando se trata de escalabilidade, possuindo mais recursos para realizar a operação de forma automática, já o *Docker Swarm* possui mais performance para levantar novos *containers*, tal trabalho auxiliou na escolha da ferramenta a ser usada para a orquestração dos contêineres na estrutura criada nos testes.

Santos (2016) apresenta o uso do *container Docker* como estratégia de virtualização para a *cloud* privada do Itaú Unibanco, no trabalho é apresentado

conceitos técnicos quanto ao funcionamento do *Docker*, norteados quanto a suas principais funcionalidades e riscos.

Costa (2018) apresenta uma solução de alta disponibilidade para banco de dados *MySQL* e aborda o uso de ferramentas que complementam a alta disponibilidade para banco de dados, utilizando uma ferramenta de balanceamento de carga e *cluster* de replicação.

Larsson (2019), apresenta um estudo de caso demonstrando na prática a execução de banco de dados no *Kubernetes*, onde foram executados procedimentos para análise quanto a performance, redimensionamento e *backups* de uma aplicação. Os resultados apresentados, demonstram que o *Kubernetes* é uma plataforma propícia para implementar banco de dados, todavia falha quanto à realização de operações administrativas.

Embora os trabalhos tenham abordado a temática de alta disponibilidade, banco de dados e *containers*, nenhum deles retrata o uso do *Kubernetes* para aplicar alta disponibilidade em banco de dados, utilizando ferramentas que auxiliam na obtenção das características de sistemas altamente disponíveis.

### 3. REFERENCIAL TEÓRICO

Neste capítulo é apresentado os principais conceitos, métodos e técnicas que fundamentaram a base técnica para o entendimento da arquitetura e testes realizados neste trabalho.

#### 3.1 ALTA DISPONIBILIDADE

A alta disponibilidade ou HA (*High Availability*), é um conceito referente a quão algo se mantém funcional, operacional ou disponível. Na informática, esse conceito está relacionado ao tempo que um sistema se mantém disponível ao uso de forma ininterrupta.

Segundo Bruschi et. al (2014), a alta disponibilidade é a característica de um sistema em permanecer disponível o maior tempo possível durante a execução de suas tarefas, tendo como características principais: tolerância a falhas de *hardware*, *software* e energia.

Existem várias formas de implementar a alta disponibilidade, e sua aplicação depende diretamente do quanto a empresa quer investir para manter seu sistema totalmente disponível, reduzindo os pontos únicos de falha, que em inglês, é denominado pela sigla SPOF (*Single Point of Failure*).

Uma métrica para avaliar o tempo desejado de disponibilidade é dado na Tabela 1, onde é listado a conversão de uma determinada porcentagem de disponibilidade para a quantidade correspondente de tempo que um sistema não estaria disponível:



Tabela 1 - Níveis de disponibilidade.

<b>Nível</b>	<b>Uptime</b>	<b>Downtime por ano</b>
1	90%	36,5 dias
2	98%	7,3 dias
3	99%	3,65 dias
4	99,8%	17 horas e 30 minutos
5	99,9%	8 horas e 45 minutos
6	99,99%	52,5 minutos
7	99,999%	5,25 minutos
8	99,9999%	31,5 minutos

Fonte: Emer (2016).

Nota: Adaptado pelo autor.

Com base na Tabela 1, a empresa deve avaliar qual porcentagem quer alcançar, a porcentagem decidida estará diretamente ligada ao custo final da arquitetura de alta disponibilidade.

Souza e Campus (2008, apud CANALI et al 2015) definem que um percentual de 100% de disponibilidade contínua, ocorre quando até mesmo paradas para manutenção são mascaradas, todavia esse percentual é apenas teórico e conforme mencionado por Costa (2009, p.12) “[...] falhas são inevitáveis em ambientes computacionais [...]”.

Alta disponibilidade é caracterizada por usar técnicas para eliminar qualquer SPOF dentro da aplicação, as técnicas definidas podem se diferenciar dependendo da aplicação e do orçamento estipulado.

Em suma, as técnicas para alcançar alta disponibilidade são: Redundância, Tolerância a falhas, Escalabilidade, Balanceamento de Carga, *Backup* e *Clusters* (CANALI, 2015), (COSTA F., 2018), (COSTA H., 2009), (EMER, 2016), (HASHIMOTO, 2009), (LETTERMAN, 2003).

### 3.1.1 Escalabilidade

Escalabilidade segundo Gregol (2011), pode ser definida como a capacidade com que um sistema ou componente pode ser modificado para atender a um determinado problema.

No meio computacional, a escalabilidade está ligada à facilidade com que um sistema ou servidor pode ser alterado para executar determinada demanda, ou seja, de aumentar os recursos necessários para suportar a carga de trabalho solicitada.

Conforme Paula Silva (2016) existem dois tipos de escalabilidade, a horizontal e vertical:

- **Escalabilidade Horizontal:** Entende-se como escalabilidade horizontal, o ato de disponibilizar novos dispositivos em uma determinada estrutura, por exemplo, aumentar de forma significativa os servidores dentro de um *cluster*, aumentando assim sua capacidade e desempenho;
- **Escalabilidade Vertical:** É o ato de aumentar os recursos físicos dentro de uma mesma máquina, podendo aumentar, sua memória, armazenamento, processamento, etc. Um exemplo de escalabilidade vertical, é o ato de aumentar o armazenamento interno em um servidor para suprir necessidade de armazenamento de dados em uma aplicação.

Devido a sua complexidade de implementação a escalabilidade é melhor aplicada em arquiteturas virtuais, como contêineres, termo abordado na seção 3.5, sendo na computação em nuvem um requisito obrigatório.

### 3.1.2 Cluster

Segundo Sá e Neves (2012, p.6) “*Cluster* é um conjunto de computadores ou sistemas interconectados entre si, trabalhando em conjunto. [...]”, tendo como finalidade trabalhar em sincronia em prol de um mesmo resultado.

Para se formar um *cluster* computacional, é necessário montar uma estrutura de servidores independentes ou que trabalhem em modo mestre escravo.

Na estrutura independente, cada servidor dentro do *cluster* pode realizar qualquer operação, sendo o resultado igual, independente de qual servidor esteja executando.

Para os *clusters* mestre-escravo, o mestre é responsável pela execução principal da aplicação, e em caso de falhas os servidores escravos operam as solicitações.

Dentre as vantagens na construção de um *cluster* é possível destacar, o alto desempenho, escalabilidade, tolerância a falhas e a independência de recursos (PITANGA 2008) (VAZ e MONKS 2014):

- **Alto desempenho:** Devido ao conjunto de servidores trabalharem de forma conjunta, é possível obter um resultado para perguntas ou operações complexas em pouco tempo;
- **Escalabilidade:** É a capacidade de incluir novos servidores ao *cluster* e aumentar seu desempenho;
- **Tolerância a falhas:** Devido a existência de vários servidores, em caso de falha, o *cluster* pode rapidamente realizar uma ação de recuperação para corrigir o problema;
- **Independência de recursos:** Por se tratarem de servidores, os *cluster* podem ser estruturados de forma independente de *software* ou *hardware* proprietários, podendo um *cluster* ser o conjunto de várias máquinas de baixo orçamento, balanceando seus processos dentro de sua estrutura de forma a obter o resultado de um servidor mais potente.

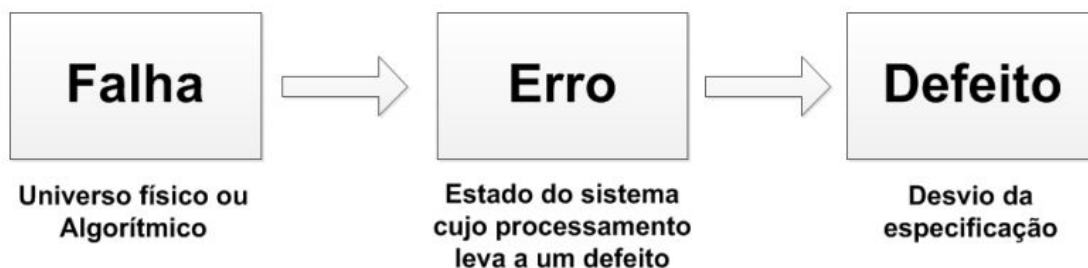
### 3.1.3 Tolerância a falhas

O conceito de tolerância a falhas se baseia em quanto um sistema pode continuar operacional mesmo após ocorrer uma falha, sendo ela crítica ou inofensiva.

Segundo Medeiros (2018), para se entender melhor as técnicas de tolerância a falhas, é necessário conceituar a relação entre falha, erro e defeito.

Na Figura 1, é descrito que a falha está ligada diretamente ao estado físico ou lógico da aplicação, o que leva ao estado de erro, sendo esse, a manifestação da

falha dentro da aplicação, por último o defeito que é o desvio do resultado original da operação.



**Figura 1 - Relação entre falha, erro e defeito.**  
 Fonte: Medeiros (2018).

As classificações de falhas podem seguir diversos critérios, podendo ser física, humana, transiente, intermitente ou permanente (SANTOS, 2015), com base nessas classificações um dos três processos de resolução pode ser aplicado, sendo eles *failover*, *failback* e *switchover* (MCKEAN et al, 2004), (ESHEL et al, 2011), (HIRST et al, 2001), (SYBASE, 2020).

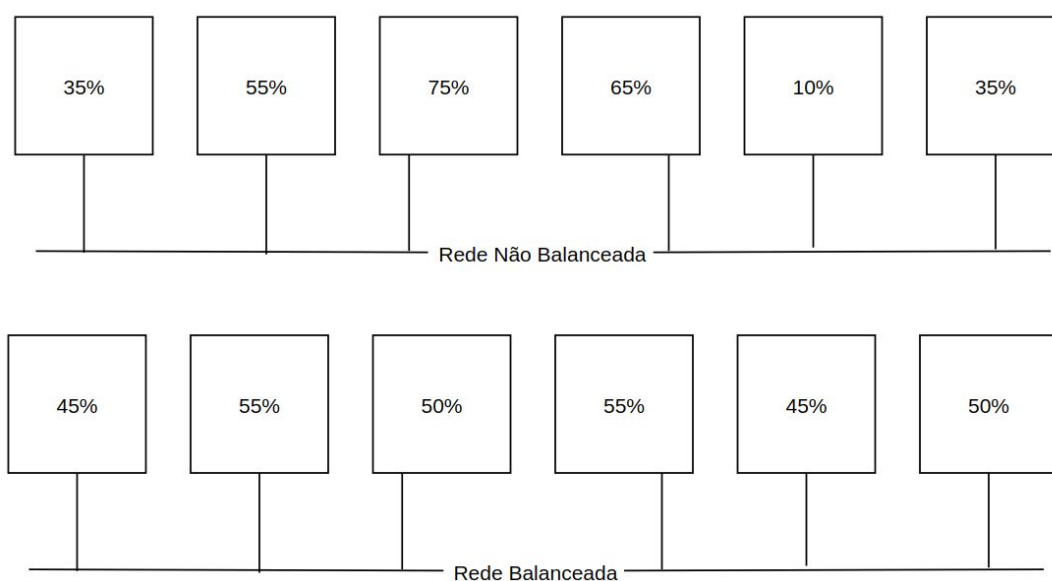
- **Failover:** A aplicação migra sua rotina sem intervenção humana, onde na ocorrência de um problema, o sistema automaticamente direciona suas rotinas para um novo servidor ou serviço, sem que isso seja perceptível ao usuário (HANWHA-SECURITY.COM, 2017), por exemplo. Um sistema de replicação de banco de dados, que na falha de um servidor mestre, automaticamente elege um novo mestre e a aplicação passa as requisições para este novo servidor;
- **Failback:** A aplicação retorna ao seu estado original devido a ocorrência de uma *failover*, sem a intervenção humana, alterando toda a estrutura para uma nova ou retornando a arquitetura para o mesmo estado antes da ocorrência da falha (FEDERICI et al, 2014), voltando ao exemplo anterior, após a ocorrência do *failover* a arquitetura de replicação volta a seu estado original, voltando os servidores a sua capacidade total definida nas configurações da arquitetura;
- **Switchover:** A aplicação migra sua rotina de forma manual, ou seja, com a intervenção de um profissional responsável (DOCS.MICROSOFT, 2020), por

exemplo, um erro na aplicação que necessite de uma atualização para corrigir o problema.

### 3.1.4 Balanceamento de carga

Balanceamento de carga, é a técnica utilizada para distribuição de carga entre processos ou servidores, alternando a rotina de trabalho entre os dispositivos disponíveis, obtendo assim uma maior quantidade de resultados com um menor tempo.

A Figura 2 ilustra o papel do balanceamento de carga em uma rede de servidores, onde na rede não balanceada nota-se uma grande variação na utilização de recursos entre servidores, tornando a arquitetura utilizada pouco produtiva.



**Figura 2 - Balanceamento de Carga dentro de uma rede de servidores.**

**Fonte: Angonese (2012).**

**Nota: Adaptado pelo autor.**

Dentre as vantagens de se utilizar balanceadores de carga, estão os fatores de desempenho e escalabilidade, onde, ao se balancear a carga entre servidores é possível obter o descongestionamento de rede e processos, e também manter de forma constante o tempo de resposta da aplicação.

Como os mecanismos de balanceamento realizam constantes verificações na estrutura a qual pertencem, se tornam grandes aliadas contra a ocorrência de falhas na aplicação, evitando e ignorando o servidor que estiver com falha, (COSTA, 2009), (ANGONESE, 2012).

### 3.2 COMPUTAÇÃO EM NUVEM

A computação em nuvem pode ser definida como um modelo de negócio, que disponibiliza ao cliente recursos como *hardware*, *software*, processamento, armazenamento, aplicações, dentre outros.

Os recursos são disponibilizados através de uma plataforma *web*, onde o cliente paga pelos recursos utilizados de forma rápida e com o mínimo esforço (ALLES, 2018), (ANGONESE, 2012), (BORGES, 2011), (SANTOS, 2016).

Tal abordagem é de extrema atratividade para empresas que necessitam de alta disponibilidade, pois todo o trabalho manual nos servidores contratados é realizado pela empresa prestadora do serviço, sendo o papel da empresa apenas monitorar os recursos contratados e dimensionar em caso de novas demandas.

Dentre as características da computação em nuvem destaca-se *pool* de recursos, escalabilidade, alta disponibilidade e monitoramento de recursos (PAULA SILVA, 2016), (POSSOBOM, 2010), (SANTOS, 2016):

- **Pool de recursos:** As plataformas de computação em nuvem costumam contar com um amplo catálogo de recursos, como armazenamento, balanceamento de carga, *cluster* computacionais, dentre outros;
- **Escalabilidade:** Devido a sua estrutura abstrata, para escalonar um serviço basta o cliente realizar uma solicitação na plataforma que em poucos minutos tais operações já estarão ativas;
- **Alta disponibilidade:** Nuvens como *Google Cloud*, *Amazon Web Service* e *Microsoft Azure* estão no mercado há um bom tempo, e por conta disso possuem uma estrutura capaz de suportar grandes períodos de atividade sem interrupções, disponibilizando assim recursos altamente disponíveis;
- **Monitoramento de recursos:** As grandes vantagens na computação em nuvem são suas ferramentas administrativas, que facilitam o monitoramento

de uma arquitetura computacional, encurtando a tomada de decisão em casos de necessidade.

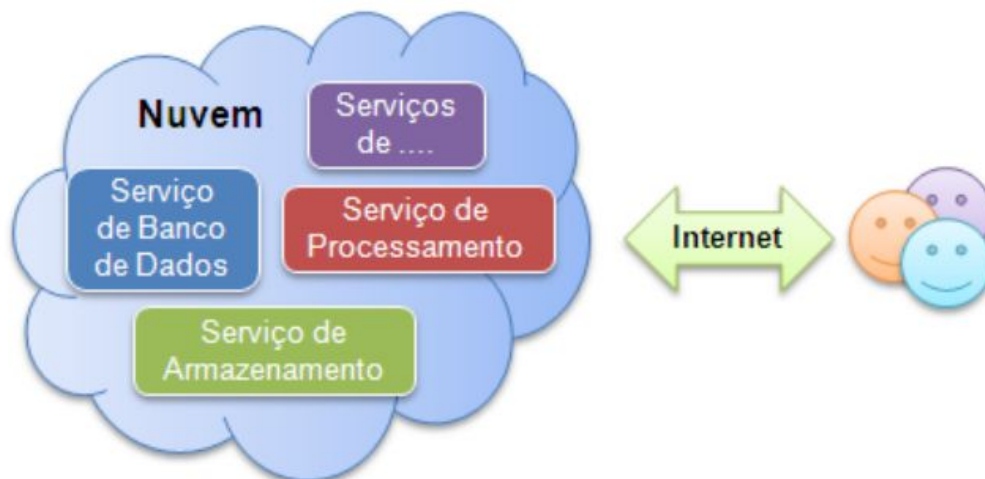
Para prover todos os benefícios citados de forma correta e especializada, a nuvem conta com 3 modelos principais de serviços, sendo eles:

- **SaaS (*Software as a Service*):** O provedor de nuvem, fornece serviços aos clientes, sendo possível somente o uso e algumas configurações, onde toda a infraestrutura fica a cargo da empresa provedora do serviço, alguns exemplos de SaaS são: *Google Docs*, *Facebook* e *Gmail* (SANTOS 2016);
- **PaaS (*Platform as a Service*):** O provedor fornece a capacidade do cliente instalar e gerenciar ferramentas, como banco de dados, serviços de mensagens, *WebService*, sistemas operacionais, dentre outras, ou seja, a capacidade de montar uma infraestrutura de *hardware* em nuvem;
- **IaaS (*Infrastructure as a Service*):** Tem como principal objetivo tornar mais fáceis e acessíveis o fornecimento de recursos, como servidores, redes e armazenamento (SOUSA 2009 apud BORGES 2011).

A nuvem computacional não segue um padrão de arquitetura ou serviços para disponibilizar seus recursos, cada nuvem pode ter sua implementação seguindo um determinado objetivo, podendo ele ser geográfico, financeiro ou estrutural, dentre os modelos de implementação mais conhecidos destaca-se a nuvem pública e nuvem privada.

As nuvens públicas provém acesso aberto ao público em geral, tais nuvens podem ser próprias, de instituições ou governos.

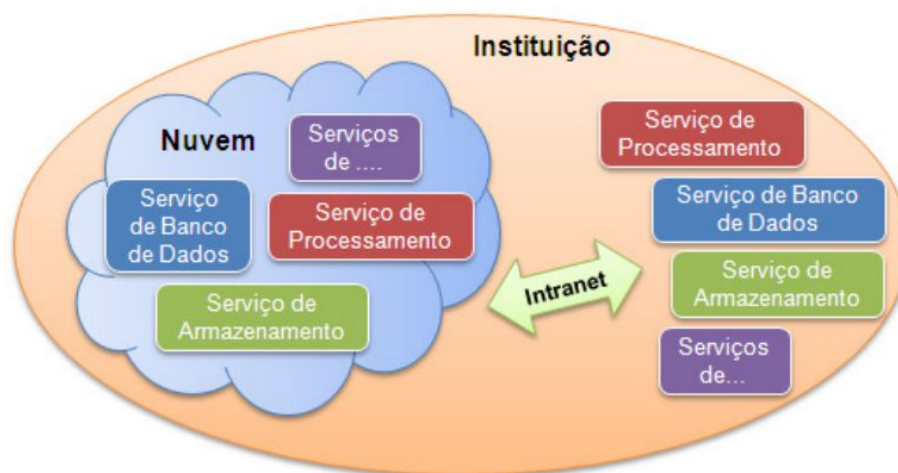
Tem como seu principal objetivo fornecer uma infraestrutura em nuvem para o usuário, cobrando seu uso por horas ou por quantidade de requisições transmitidas/recebidas em sua plataforma (PAULA SILVA, 2016), uma melhor visão do funcionamento das nuvens públicas é dado pela Figura 3.



**Figura 3 - Nuvem pública.**  
**Fonte: Borges (2011).**

As nuvens privadas podem ser definidas como nuvens empresariais, sendo infraestruturas particulares de uma empresa ou organização.

Seu uso e administração é totalmente provido pela empresa detentora da nuvem, podem estar fisicamente ou em servidores remotos, sendo o acesso e a comunicação com os serviços prestados restrito aos usuários da corporação (BORGES 2011), a arquitetura de uma nuvem privada é dada pela Figura 4.



**Figura 4 - Nuvem privada.**  
**Fonte: Borges (2011).**



Em um contexto geral, a nuvem exerce um grande papel para serviços de alta disponibilidade, contando com várias camadas de recursos e possibilidades.

Sendo assim, a utilização de um ambiente em nuvem deve sempre ser avaliada no projeto de um sistema, mesmo seu custo podendo ser mais elevado em consideração a uma infraestrutura local, as vantagens fornecidas se tornam um fator diferencial para o seu uso e o sucesso de uma aplicação.

### 3.3 BANCO DE DADOS

Para Elmasri e Navathe (2011, p. 3) “Um banco de dados é uma coleção logicamente coerente de dados com algum significado inerente”, onde um banco de dados é arquitetado e construído para atender a um propósito específico.

Em outras palavras, um banco de dados é um agrupamento de informações persistentes para um propósito final, onde as informações ali contidas devem ser legíveis, consistentes e disponíveis.

Em sua trajetória surgiram diversos conceitos de banco de dados sendo alguns deles, o modelo relacional que utiliza em suas operações a linguagem SQL (*Structured Query Language*), linguagem essa que por sua importância se tornou um padrão ISO (*International Organization for Standardization*).

O modelo orientado a documentos é mais conhecido como *NoSQL*, devido a linguagem de utilização não ser SQL.

Por último, o paradigma *NewSQL* o qual tenta fazer a junção dos modelos relacionais e não relacionais (KNOB, 2018).

Com a chegada do banco de dados, surgiram as necessidades de manipular esses dados de forma prática, para isso surgiram os SGBD (Sistemas Gerenciadores de Banco de Dados).

Os SGBD's não somente vieram com o intuito de facilitar a manipulação das bases de dados, como também vieram com intuito de estruturar esses bancos de forma a mantê-los sempre consistentes, disponíveis e seguros (ELMASRI e NAVATHE, 2011, p. 19).

Dentre as principais características de um SGBD destaca-se o controle de concorrência, segurança, recuperação de falhas, mecanismos de gerenciamento de

armazenamento de dados, controle das restrições de integridade e as transações (ELMASRI e NAVATHE, 2011, p. 19).

As transações exercem um dos principais papéis, por estarem diretamente ligadas à manipulação de dados, e por essa razão devem ser atômicas, consistentes, isoladas e duráveis. Na computação esse conceito é denominado ACID (*Atomicity - Consistency - Isolation - Durability*), definindo-se:

- **Atomicity:** Todas as operações presentes nas transações devem ser executadas, ou nada será executado;
- **Consistency:** Após uma transação ser concluída, o banco de dados deve permanecer em um estado consistente, ou seja, deve satisfazer as condições de consistência e restrições de integridade previamente assumidas;
- **Isolation:** Se duas transações estão sendo executadas concorrentemente, seus efeitos devem ser isolados uma da outra. Esta propriedade está relacionada ao controle de concorrência do SGBD;
- **Durability:** Uma vez que uma transação ocorreu com sucesso, seu efeito não poderá mais ser desfeito, mesmo em caso de falha. Esta propriedade está relacionada à capacidade de recuperação de falhas do SGBD.

### 3.4 A ALTA DISPONIBILIDADE COM MYSQL

O *MySQL* é um banco de dados relacional de código fonte aberto com base na licença GPL (*General Public License*), com versões comerciais e gratuitas.

Atualmente sendo desenvolvida e distribuída pela *Oracle Corporation*, tem por finalidade ser um sistema gerenciador de banco de dados robusto e consistente, manipulando grandes volumes de dados de maneira eficiente.

Dentre as diversas técnicas para se alcançar alta disponibilidade em banco de dados no *MySQL*, destaca-se as técnicas de replicação *master-slave* e *replication group*.

### 3.4.1 Replicação *master-slave* no *MySQL*

Sendo a topologia mais simples do *MySQL*, a replicação mestre-escravo, consiste em um servidor principal que receberá instruções, sendo elas de escrita ou leitura, e um servidor secundário que ficará responsável por replicar os dados do mestre por meio de replicação assíncrona ou semissíncrona, podendo este ser utilizado para fins de leitura ou substituição do mestre em caso de falha, na Figura 5 é ilustrado o funcionamento de uma replicação *master-slave*.

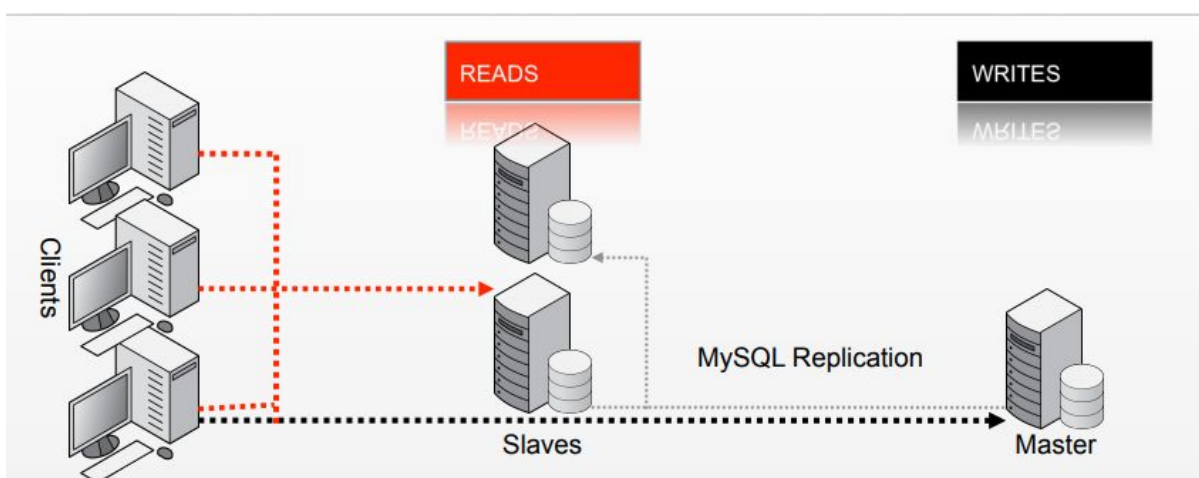


Figura 5 - Replicação *MySQL master-slave*.  
Fonte: Oracle.com (2020).

Na Figura 5, identifica-se como a estrutura da replicação é efetivada, onde os clientes podem se comunicar com os servidores escravos para consulta e gravação através do servidor mestre, esse que por sua vez realiza a operação e transcreve para um arquivo, que será usado como mecanismo de replicação.

### 3.4.2 Replicação de grupo no *MySQL*

A replicação de grupo é um conjunto de servidores que cada um tem sua própria cópia inteira dos dados, e interage entre si através de mensagens.

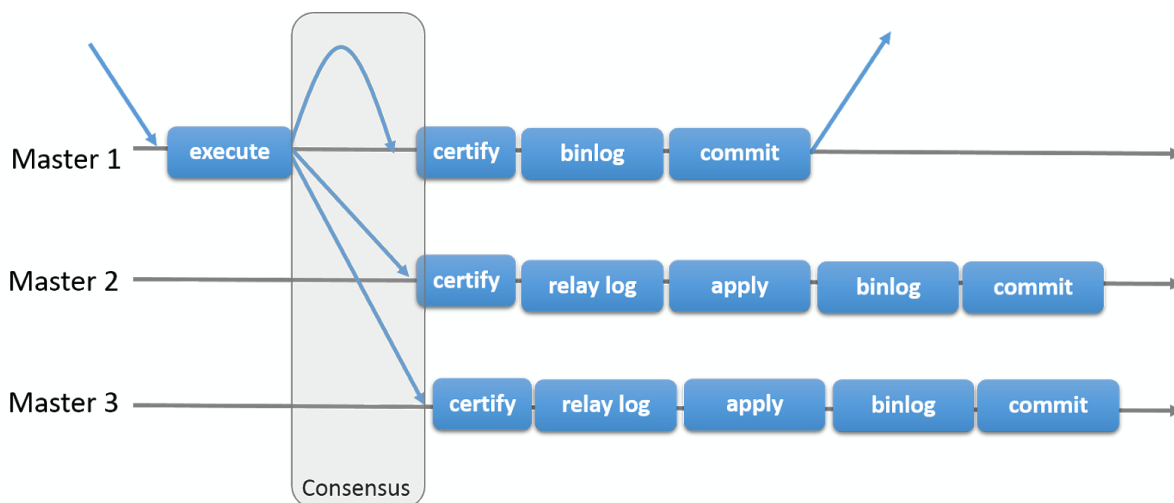
O grupo de servidores pode executar transações a qualquer momento, entretanto todas as transações de gravação precisam ser aprovadas pelos membros do grupo para serem efetivadas.

Já as transações de leitura são confirmadas imediatamente sem a necessidade de validação pelo grupo (MYSQL.COM, 2019a).

O conjunto de servidores presentes no grupo de replicação são independentes e recebem uma chave de identificação de usuário, denominado UID (*User Identifier*), podendo entrar ou sair do grupo sem afetar o ecossistema do grupo, quando um servidor sai do grupo automaticamente, os demais participantes notam sua ausência e reconfiguram o grupo.

Após a volta desse servidor o mesmo se atualiza através da replicação assíncrona do *MySQL* usando o estado de algum servidor ativo no grupo.

Toda a estrutura do grupo é feita através dos protocolos GCS (*Group Communication System*), eles fornecem um mecanismo de detecção de falhas, associação ao grupo e comunicação segura e ordenada aos integrantes. Na Figura 6 é dado uma breve ilustração de como funciona a replicação de grupo.



**Figura 6 - Replicação de Grupo.**  
**Fonte: Mysql.com (2019a).**

Como apresentado na Figura 6, o cliente realiza uma operação, que será verificada pelo grupo.

Após a verificação e aprovação, o servidor que está executando a transação grava a mesma em um arquivo de replicação e conclui a alteração, os demais servidores aguardam a atualização do arquivo de replicação e aplicam as alterações assim que atualizado.

### 3.4.2.1 Modo primário

Neste modo o grupo possui somente um servidor marcado com permissão para gravação, os demais são definidos para somente leitura.

Para eleger um novo servidor principal, é necessário o grupo passar por um processo de falha, conforme ilustrado na Figura 7, ou eleger manualmente um novo mestre.

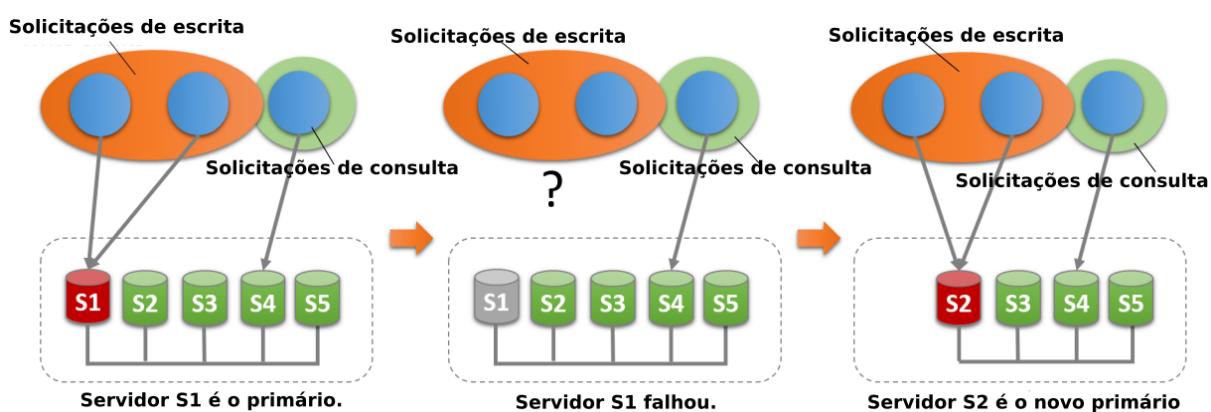


Figura 7 - Eleição de novo primário após *failover*.

Fonte: Mysql.com (2019b).

Nota: Adaptado pelo autor.

O processo de eleição de um novo mestre envolve cada servidor do grupo analisando uma nova visão do grupo, avaliando os possíveis novos membros primários e ordenando pelo mais qualificado.

Cada membro toma a decisão localmente utilizando um algoritmo de eleição em sua versão do *MySQL*, a eleição do novo mestre somente será dada caso todos os membros tomem a mesma decisão.

Conforme descrito pelas normas ACID o grupo precisa manter a consistência mesmo após passar por um processo de falha, para evitar a inconsistência o *MySQL* possibilita variáveis de controle, que podem mudar de versão para versão.

Em casos onde possa haver uma lista de pendências a serem aplicadas pelo servidor antigo, pode-se configurar para que o novo servidor seja disponibilizado imediatamente para o tráfego do aplicativo ou restringido o acesso até a lista ser totalmente aplicada.

Com a primeira abordagem, estima-se garantir uma associação estável com o tempo mínimo possível, após a mudança de líder, dessa forma a consistência de gravação é garantida, enquanto as leituras exibem os dados antigos até o novo primário aplicar a lista de pendências.

Já para a segunda alternativa, o sistema assegura uma associação estável ao grupo após a falha conforme o primeiro caso, todavia somente libera o grupo para operação após a aplicação da lista de pendências.

### 3.4.2.2 Multiprimário

Neste modo de configuração, nenhum servidor exerce uma função especial, qualquer membro que seja compatível com o grupo poderá realizar leitura e gravação.

Em caso de falha, os clientes podem ser redirecionados para qualquer outro membro em operação, o *MySQL Group* não realiza balanceamento de carga do lado do cliente, por esta razão é aconselhado a utilização de um *proxy* reverso ou balanceamento de carga para exercer esse papel.

Na Figura 8 é dado um exemplo em caso de falha, onde é possível identificar o processo executado pelo cliente após a falha do servidor.

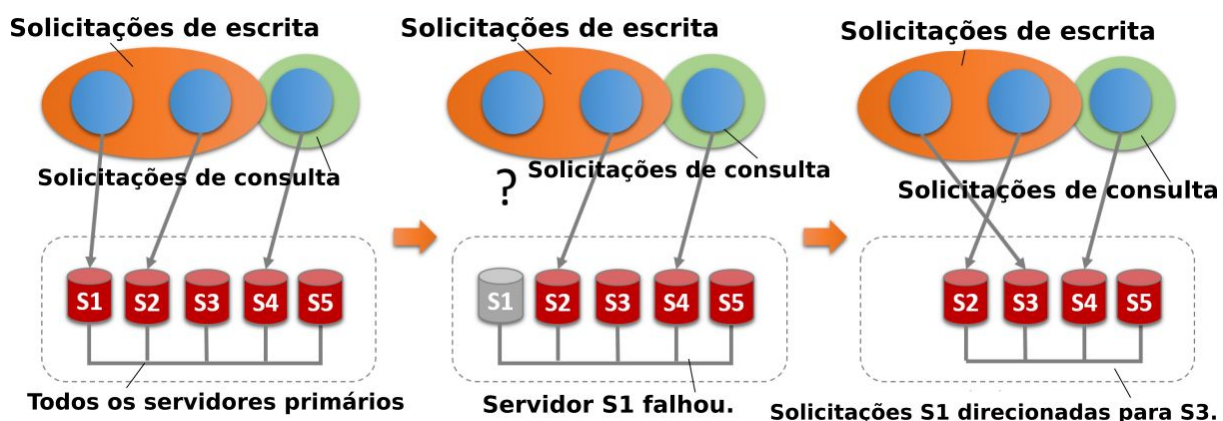


Figura 8 - Ocorrência de *failover* em estrutura multiprimário.

Fonte: Mysql.com (2019c).

Nota: Adaptado pelo autor.

Conforme identificado na Figura 8, são ilustrados dois clientes realizando escritas em dois servidores, onde um deles sofre uma falha. Ao ser identificado a falha, esse cliente é automaticamente direcionado para outro servidor em operação, mantendo assim o sistema sempre disponível.

Assim como o modo primário único, é possível configurar garantias para revisar dados obsoletos causados por tráfego de rede, essas configurações podem diferir entre versões do *MySQL*.

Neste modo também é realizado um processo de verificação rigoroso para a consistência de dados, onde, em caso de uma transação ser executada a nível de isolamento, sua confirmação falhará automaticamente ao sincronizar ao grupo.

Outra medida de verificação está a nível de estrutura, onde caso uma tabela possua chaves estrangeiras com restrição em cascata, a confirmação de alteração ao sincronizar falhará.

### **3.4.3 Estrutura de replicação**

Para se configurar uma estrutura de replicação *MySQL*, deve-se levar em consideração alguns pontos, nessa seção será ilustrado como funciona a estrutura de configuração e alertas em relação a configurações indevidas.

Como ponto de partida, observa-se que para se utilizar a replicação de grupo do *MySQL*, deve-se armazenar todos os dados em formato *InnoDB*, o uso de outros meios de armazenamento podem acarretar em erros na replicação.

Para manter a consistência do grupo, deve-se obter um consenso entre eles, e para que a concordância seja obtida é necessário que a maioria do grupo aceite determinada decisão.

Caso um grupo seja configurado de maneira incorreta, essa consonância pode ser perdida e acarretar em um *split-brain* (cérebro dividido), ocasionando na perda total do grupo.

A Tabela 2 ilustra o número de instâncias necessárias para se obter tolerância a falhas em um grupo de replicação *MySQL*.

Tabela 2 - Número de instâncias por ocorrência de falhas.

Tamanho do grupo	Maioria	Falhas instantâneas toleradas
1	1	0 0
2	2	0 0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

Fonte: Mysql.com (2019d).

### 3.5 CONTAINERS DE SOFTWARE

Os contêineres de *software* são arquiteturas que possuem como objetivo deixar as arquiteturas de *software* isoladas, de fácil implementação e fácil manuseio.

Conforme RUBENS, P. (2017), contêineres são uma maneira de prover isolamento e garantir uma execução consistente e portátil de aplicações.

Um contêiner de *software* em sua essência roda acima do *kernel* do sistema operacional, tornando sua implementação e execução mais rápidos que uma máquina virtual padrão.

Na Figura 9, é ilustrado como é a arquitetura de um *container*, onde está sendo executado dois contêineres, onde cada um deles executa duas aplicações isoladas.



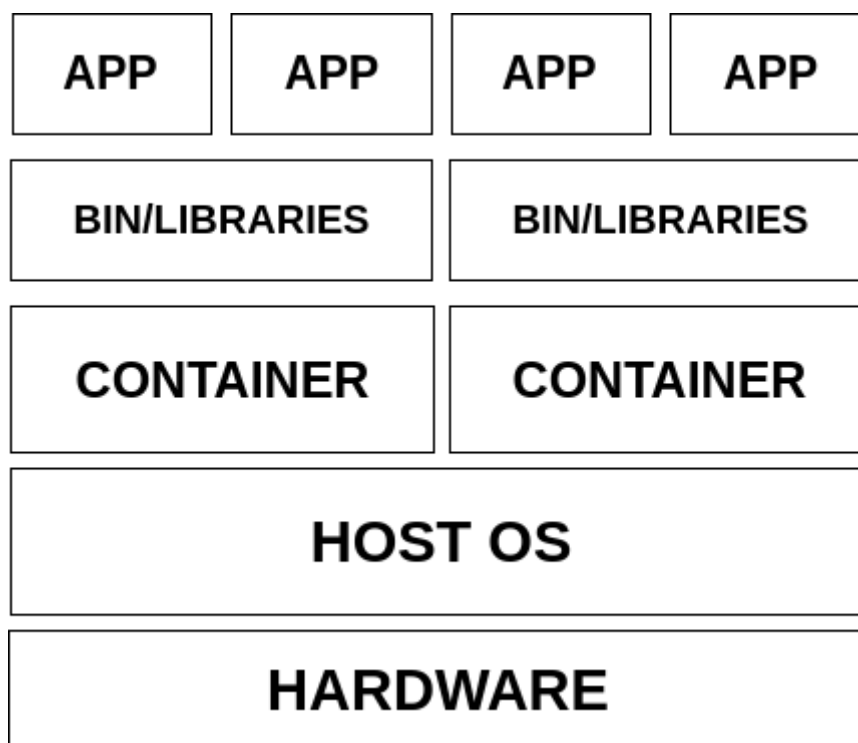


Figura 9 - Arquitetura de um *container de software*.

Fonte: Autor.

Devido às suas estruturas isoladas, os *containers* podem ser usados para servir um único propósito, tornando-se uma ferramenta extremamente versátil quando o assunto é alta disponibilidade.

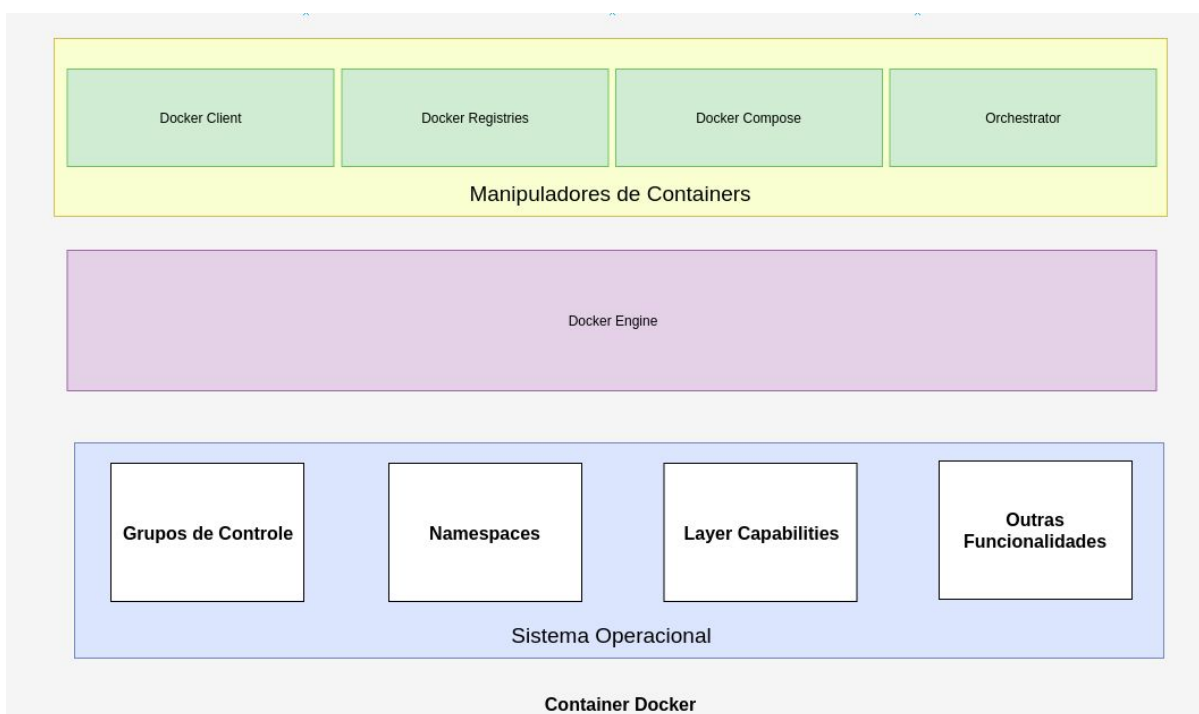
Dentre os principais mecanismos de containerização, destaca-se o *Docker* como principal *engine* de criação de ambientes containerizados. Dentre suas vantagens, fornece alguns isolamentos de recursos como:

- Limites de uso de memória;
- Limites de uso de CPU (*Central Process Unit*);
- Limites de uso de entrada e saída;
- Limites de uso de rede;
- Isolamento da rede;
- Isolamento do *file system*;
- Permissões e Políticas;
- Capacidades do *kernel*.

### 3.5.1 Docker

Sendo sua primeira aparição em 2013, durante a *PyCon US*, é um projeto de código aberto que automatiza a implantação de aplicações dentro de ambientes containerizados utilizando recursos existentes no *Kernel Linux* (MESSINA, 2018).

Segundo Corrêa, J. (2016), o *Docker* utiliza funcionalidades de isolamento de recursos do *Kernel Linux* para a criação de *containers* independentes, com o *namespaces* (nome do espaço de trabalho) do *kernel* isolando a visão de uma aplicação em relação ao sistema operacional vigente na máquina, enquanto o *cgroups* (grupos de controle) fornece limitações de recursos. Na Figura 10 é ilustrado como é formado a estrutura *Docker*.



**Figura 10 - Imagem ilustrativa da estrutura geral do Docker.**

**Fonte: Autor**

Nas próximas seções, serão abordados os principais conceitos para se entender a arquitetura e funcionamento do *Docker*, tal entendimento é de grande valia para a compreensão do funcionamento do *Docker* em relação ao *Kubernetes*.

### 3.5.1.1 Camada de sistema operacional

#### 3.5.1.1.1 *Namespace*

É uma abstração do *Kernel Linux*, que faz com que os processos ali contidos fiquem isolados e só podem ser visualizados por outros processos no mesmo espaço de nome.

#### 3.5.1.1.2 Grupos de Controle - *CGroups*

São recursos do *Kernel* do sistema operacional *Linux* que permitem organizar os processos em grupos hierárquicos cuja utilização pode ser limitada e monitorada.

#### 3.5.1.1.3 *Layer Capabilities*

São os formatos de armazenamento ou sistemas de arquivos que permitem aos usuários especificar um conjunto de diretórios, que são apresentados aos usuários como um único diretório virtual, no *Docker* o mais utilizado é o *Union Filesystem*, todavia pode-se utilizar diversas alternativas como AUFS, btrfs, vfs e *DeviceMapper*.

### 3.5.1.2 Arquitetura *Docker*

#### 3.5.1.2.1 *Docker Engine*

O *Docker Engine* é uma aplicação cliente-servidor que utiliza em sua estrutura um *daemon* (sistema ou operação executado em segundo plano); Uma API (*Application Programming Interface*), que utiliza para suas operações de listagem, inserção, alteração e remoção de dados, a abstração de *software* REST (*Representational State Transfer*), na computação esse conceito é denominado *API Rest*;

Por último, um CLI (*Command Line Interface*), onde o CLI através da *API Rest* controla ou interage com o *daemon* por meio de *scripts* ou comandos diretos (DOCKER.COM, 2019a).

O *daemon* é responsável pelo trabalho de construir, executar e distribuir seus *containers Docker*, o CLI e o *daemon* podem ser executados no mesmo sistema ou podem ser conectados a um outro *daemon Docker* remoto, na Figura 11 ilustra-se como funciona o *Docker Engine*.

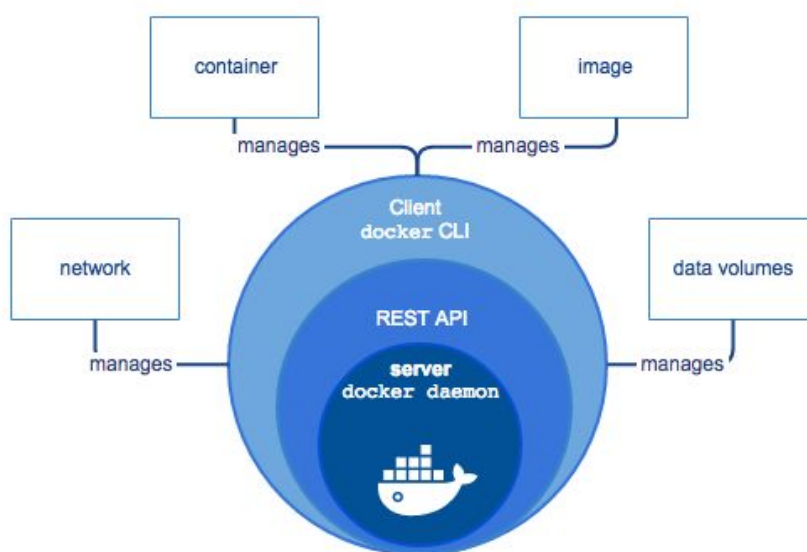


Figura 11 - Arquitetura *Docker Engine*.  
Fonte: Docker.com (2019b).

### 3.5.1.2.2 Imagens *Docker*

Uma imagem é um modelo somente leitura com instruções para criar um contêiner do *Docker*, esses modelos são criados a partir de instruções inseridas em um documento chamado *Dockerfile* (DOCKER.COM 2019a).

Cada imagem consiste em uma série de camadas (*layers*), essas camadas são combinadas em um único *filesystem* consistente por meio da tecnologia *UnionFS*, quando a imagem é atualizada ou recriada, somente as camadas afetadas são atualizadas (SILVA, F. 2017).

### 3.5.1.2.3 *Container Docker*

Os *containers Docker* são a junção de toda a arquitetura, encapsulando todos os recursos necessários para a execução das aplicações, após o contêiner estar em execução diversas operações podem ser realizadas, tais como: iniciar, parar, reiniciar ou excluir.

### 3.5.1.2.4 *Dockerfile*

São arquivos onde se inserem todas as instruções necessárias para a criação de uma imagem *Docker*, as instruções seguem um conjunto de passos onde se informa os requisitos necessários como versão, nome e sistema operacional, que serão usados para a montagem e instalação da imagem *Docker*.

A estrutura de um *Dockerfile* deve seguir um conjunto de instruções e argumentos, seguindo uma ordem específica de execução, o Quadro 1 relaciona algumas das instruções mais relevantes para a composição do arquivo.

**Quadro 1 - Comandos *dockerfile*.**

Instrução	Definição
<i>From</i>	Define a imagem base que será usada para a criação da nova imagem. Esta deve ser a primeira instrução a ser informada no arquivo.
<i>Run</i>	Ir� executar qualquer comando em uma nova camada criada na parte superior da imagem atual e gravar o resultado final, a imagem resultante ser� usada pela pr�xima instrução do <i>DockerFile</i> .
<i>CMD</i>	Fornece o processo padr�o para o cont�iner em execu�o, no caso o comando que ser� realizado ao executar a imagem.
<i>Expose</i>	Informa ao <i>docker</i> que o cont�iner escuta instru�es vindo de uma determinada porta passada como argumento da instru�o.
<i>ENV</i>	Define uma vari�vel de ambiente.
<i>ADD/COPY</i>	C�pia novos arquivos ou diret�rios e os adiciona ao sistema de arquivos da imagem no caminho especificado.
<i>Volume</i>	Cria um ponto de montagem com o nome especificado e o marca como um diret�rio a ser montado externamente no <i>host</i> ou outros <i>containers</i> .
<i>User</i>	Define o nome de usu�rio a ser usado ao executar a imagem.

Fonte: Silva (2017).

Nota: Adaptado pelo autor.

### 3.5.1.3 Manipuladores de *containers*

Os manipuladores de *containers* tem como papel executar ações fora da arquitetura dos *containers*, podendo ir desde armazenar imagens *Docker* á orquestrar um *cluster* de *containers*.

#### 3.5.1.3.1 Registros *Docker*

Os registros *Docker*, são responsáveis por armazenar e distribuir imagens *Docker's*, por padrão o *Docker* está configurado para buscar da biblioteca pública *Docker Hub*, todavia é possível criar um registro privado usando o DDC (*Docker Datacenter*) (DOCKER.COM, 2019a).

#### 3.5.1.3.2 *Docker Compose*

O *compose* é uma ferramenta para definir e executar aplicativos *Docker* de várias *containers*, ao utilizar o *compose*, utiliza-se um arquivo *YAML* para configurar os serviços do aplicativo, onde com ele será possível a criação conjunta de grupos de *containers* (DOCKER.COM, 2019c).

## 3.6 KUBERNETES

O *Kubernetes* é um sistema de código aberto, que visa automatizar a implantação, dimensionamento e gerenciamento de aplicativos em *containers* (KUBERNETES.IO, 2020a).

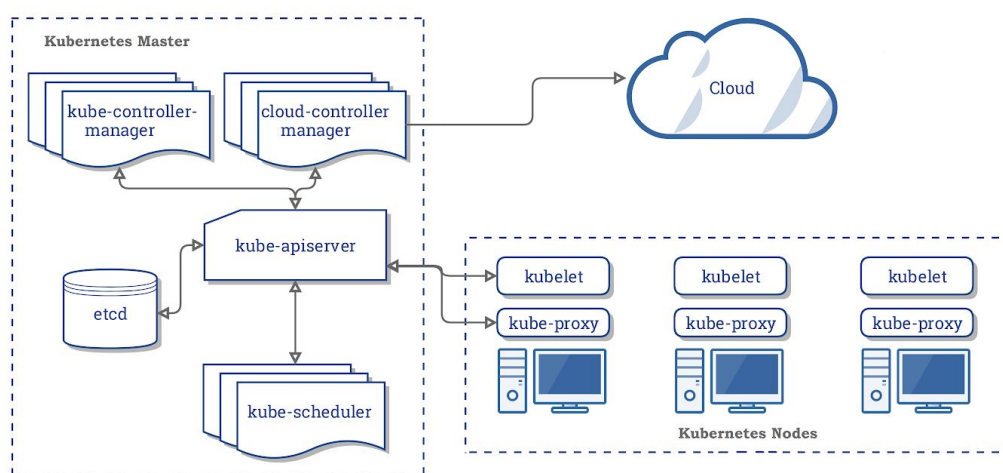
Seu surgimento se deve a partir da experiência que o *Google* obteve ao utilizar outras soluções, e tornou-se atualmente uma das soluções mais utilizadas para orquestração de *containers* no mercado. Dentre os principais recursos oferecidos, destaca-se:

- Descoberta de serviço e balanceamento de carga;
- Orquestração de armazenamento;
- Implementação e reversão automatizadas;
- Autocorreção;
- Gerenciamento e configuração de segurança.

Ao realizar a implantação do *Kubernetes*, é obtido um *cluster*, que possui um nó de trabalho e um mestre.

O nó mestre é a máquina que disponibiliza os serviços do *cluster* através de uma *API*, e é responsável por coordenar suas atividades.

Os nós de trabalho, são máquinas que funcionam como os trabalhadores do *cluster*, sendo responsáveis pela execução dos contêineres. Na Figura 12 é ilustrado como funciona um *cluster Kubernetes* com todos seus componentes conectados.



**Figura 12 - Cluster Kubernetes.**  
**Fonte: Kubernetes.io (2020b).**

### 3.6.1 Nó mestre

Os componentes do nó mestre, são responsáveis por fornecer o plano de controle do *cluster*, fica a cargo do mestre tomar decisões globais, e responder a eventos do *cluster*, como se recuperar de uma falha ou direcionar as requisições para outras máquinas de trabalho, os componentes principais que compõem o nó mestre são:

- **Kube-apiserver:** É o componente que expõe a *API* do *Kubernetes* (KUBERNETES.IO 2020b), onde ficará responsável por prestar os serviços às operações *REST*, fornecendo assim, o *frontend* entre a máquina de trabalho e os serviços de controle;

- **Etc**: É um tipo de armazenamento de chave-valor consistente e altamente disponível (ETCD, 2020), é responsável por armazenar os dados de configuração do *cluster* e o estado do mesmo;
- **Kube-scheduler**: É o componente responsável por executar tarefas de agendamento, é também responsável por agendar novas tarefas dentro do *cluster*;
- **Kube-controller-manager**: É o componente que executa *loops* de controle que observam o estado do *cluster*, sendo responsável por manter o estado original da arquitetura conforme foi designado em sua configuração;
- **Cloud-controller-manager**: É um *daemon* que incorpora *loops* de controle específicos da nuvem, é responsável por executar controladores que interagem com provedores de nuvem.

### 3.6.2 Nó Trabalhador

Os componentes do nó trabalhador, são responsáveis por manter os grupos de *containers* em execução seguros e fornecer o tempo de execução e o *status* dos *containers* ao *cluster*, sendo seus principais componentes:

- **Kubelet**: É um agente executado em cada nó do *cluster*, responsável por garantir que os contêineres estejam em execução;
- **kube-proxy**: É um *proxy* de rede executado em cada nó do *cluster*, é responsável por realizar o gerenciamento de rede dos nós;
- **Container Runtime**: São os *software* de contêineres executados em cada nó.

### 3.6.3 Conceitos gerais

Além dos já abordados, alguns conceitos importantes para o entendimento do *Kubernetes* são: *Node*, *Pods*, *Controller ReplicaSet*, *Controller StatefulSet*, *PersistentVolums*, *Storage Class*, *Kubectl*, *Configmap*, *YAML*, *Service*, *Deployment*, e *Custom Resources*.



### 3.6.3.1 Node

Um *node*, ou nó em português, é uma máquina de trabalho que compõe a estrutura do *Kubernetes*, podendo ser uma máquina virtual ou física, os nós são responsáveis por estruturar e executar os *pod's* dentro da arquitetura (KUBERNETES.IO, 2020c).

### 3.6.3.2 Pod

Um *pod* é a unidade básica de execução de um aplicativo *Kubernetes*. Os *pod's* representam processos em execução no *cluster* e correspondem a uma unidade de implantação e uma única instância, que pode consistir em um único contêiner ou um grupo de *containers* fortemente acoplados e que compartilham recursos. (KUBERNETES.IO, 2019a).

Os contêineres presentes em um *pod* são vistos externamente como uma entidade única e dividem o mesmo domínio de rede, possuindo um mesmo endereço IP (*Internet Protocol*) e compartilhando todo o conjunto de portas, como se fossem um único contêiner ou máquina.

Cada *pod* recebe um endereço IP exclusivo. Cada contêiner em um *pod* compartilha o *namespace* da rede, incluindo o endereço IP e as portas de rede. Os contêineres dentro de um *pod* podem se comunicar usando o *host*.

Quando os contêineres em um *pod* se comunicam com entidades fora do *pod*, eles devem utilizar recursos de rede compartilhados (como portas) (KUBERNETES.IO, 2019a).

Um *pod* pode especificar um conjunto de armazenamento compartilhado. Todos os contêineres no *pod* podem acessar os volumes compartilhados, permitindo que esses contêineres compartilhem dados (KUBERNETES, 2019a).

### 3.6.3.3 Controller ReplicaSet

O objetivo do *controller replicaset* é manter um conjunto estável de *pod's* replicados em execução a qualquer momento.

E definido no *replicaset* a quantidade de *pod's* que deve ser mantido em execução, quantos *pod's* serão criados e quais critérios serão usados para criação.

Seu papel é manter em operação a quantidade exata de *pod's* informados, podendo destruir e criar *pod's* conforme sua necessidade (KUBERNETES.IO, 2020d).

#### 3.6.3.4 *Controller StatefulSet*

*Controller Statefulset* é o objeto da *API* usado para gerenciar aplicações com estado (KUBERNETES.IO, 2019b). Eles representam um conjunto de *pod's* com identidades únicas e permanentes, garantindo a ordem e exclusividades dos *pod's*.

Sendo as informações de estado e outros dados resilientes de qualquer *pod* usando *statefulset*, mantidos em *PersistentVolumns*.

#### 3.6.3.5 *PersistentVolums*

*PersistentVolume* são tipos de armazenamento durável, eles são recursos que existem independentes dos *pod's*, ou seja, mesmo se o *pod* for excluído os dados que continham nele estarão armazenados nos volumes persistentes, e após um novo *pod* criado com a mesma configuração, o mesmo poderá usufruir desses dados já armazenados, sendo totalmente gerenciados pelo *cluster Kubernetes*.

Sua implementação é dada pelo *StorageClass*, que fornece uma maneira de descrever as classes de armazenamento e será explicado a seguir (KUBERNETES.IO, 2020e).

#### 3.6.3.6 *Storage Class*

É a forma com o qual os administradores do *cluster* descrevem as classes ou perfis de armazenamento ofertadas.

#### 3.6.3.7 *Kubectl*

O *kubectl* é uma interface de comando, utilizada para executar ações nos *clusters* do *Kubernetes*, é através dele que visualiza-se as informações do *cluster*, como *status*, atualizações, quantidade, dentre outros.

### 3.6.3.8 Configmap

Um *configmap* na arquitetura *Kubernetes* é um arquivo de configuração que pode ser usado por *pod's* para armazenar dados de configurações, essas configurações são imutáveis, servindo como base para ocorrência de *failover* ou novos *pod's* adicionados a estrutura.

Para se atualizar os *configmap's* é necessário a criação de um novo *configmap* e realizar a migração dos *pod's* para consumirem o novo arquivo configurado mantendo assim a escalabilidade da estrutura (KUBERNETES.IO, 2020f).

### 3.6.3.9 YAML

*YAML* é uma linguagem de serialização de dados amigável padrão para todas as linguagens de programação (YAML, 2020).

Dentro do *Kubernetes* existem formas de configurar o *cluster* através de arquivos de configuração, sendo o *YAML* o recomendado na documentação do *Kubernetes*, devido a sua familiaridade com padrões de configuração, no Quadro 2 é apresentado um exemplo de arquivo *YAML* aplicado ao *Kubernetes*.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

**Quadro 2 - Representação de arquivo YAML.**  
Fonte: Kubernetes 2019c.

Conforme apresentado no Quadro 2, as configurações para se criar um *cluster Kubernetes* utilizando um arquivo *YAML* possui um padrão de implementação, sendo os campos principais *apiVersion*, *kind*, *metadata* e *spec*, onde:

- **ApiVersion:** Versão da *API Kubernetes* utilizada para criar a configuração;
- **Kind:** Tipo de configuração;
- **Metadata:** Dados para identificar a configuração, como nome, *id*, etc;
- **Spec:** As configurações desejadas para a configuração.

Dentro do *Kubernetes* esses arquivos de configuração são denominados “objetos”, que para o *Kubernetes* são entidades persistentes, na qual servem como

representação do estado desejado do *cluster Kubernetes* (KUBERNETES.IO, 2019d).

Para a *API Kubernetes* o *cluster* deve sempre manter seu estado de acordo com a configuração descrita no arquivo *YAML*.

### 3.6.3.10 *Service*

No *Kubernetes*, serviços são uma maneira abstrata de expor um aplicativo em execução em um conjunto de *pod's*, implementando-os como um serviço de rede (KUBERNETES.IO, 2020g).

Os *pod's* são mortais, eles nascem e caso morram outros são colocados em seu lugar, devido a esse fator, para manter os processos ali executados de maneira consistentes são definidos os serviços.

Esses serviços podem ser expostos de quatro formas diferentes, que no *Kubernetes* são denominados *ServiceTypes*, sendo eles:

- **ClusterIp**: Expõe o serviço em um IP interno do *cluster*. Ao optar por esse *ServiceType* o serviço ficará acessível apenas de dentro do *cluster*;
- **NodePort**: Expõe o serviço no IP de cada nó em uma porta estática;
- **LoadBalancer**: Expõe o serviço externamente usando o balanceador de carga disponível na arquitetura;
- **ExternalName**: Mapeia o serviço para um nome especificado, por exemplo: `servico.mysql.com`.

### 3.6.3.11 *Deployment*

No *deployment* são realizados as criações, alterações ou novas implantações para os *pod's*, através dele é possível escalar o *cluster*, podendo aumentar sua capacidade ou diminuir dependendo do estado da aplicação (KUBERNETES.IO, 2019c).

Na criação de um *deployment* é definido os parâmetros de implementação ou atualização, dos quais incluem: número de *pod's* que devem ser criados, número mínimo que deve estar disponível, entre outros. Para se aplicar uma nova atualização a uma aplicação existente, utiliza-se a estrutura do arquivo *YAML* e o *Kubectl*.

### 3.6.3.12 Custom Resources

No *Kubernetes* é possível incluir recursos personalizados, esses recursos são um ponto de extremidade na *API* que armazena uma coleção de objetos de um determinado tipo, ou seja, representa uma personalização de instalação específica do *Kubernetes*, tornando-o mais modular.

Existem duas maneiras de adicionar um recurso personalizado ao *cluster*, sendo eles o *custom resources definition* e a *api server aggregation*:

- CRD (*Custom Resources Definition*), é a maneira mais simples de se utilizar, ele permite criar novos tipos de recursos através de objetos, definindo o recurso através da *API CustomResourceDefinition*, esse novo recurso ao ser criado, recebe o nome e o esquema que foi especificado (KUBERNETES.IO, 2019e).
- ASA (*API Server Aggregation*), permite criar implementações personalizadas, escrevendo e implementando um servidor de *API* próprio e independente, o servidor principal delega solicitações para o recurso personalizado disponibilizando-os para o *cluster*, para se utilizar esse mecanismo de inclusão de recurso é necessário realizar programação na linguagem *GO* e a construção de um binário (KUBERNETES.IO, 2019f).

## 4. METODOLOGIA E EXPERIMENTO

Para a execução deste trabalho, foi utilizado o recurso *Kubernetes Engine* na nuvem *Google Cloud* na versão 1.13.11-gke.23, sendo o tipo de máquina n1-standard-2 contendo 2vCPU, 7,5 de memória, utilizando como base o sistema operacional COS (*Container-Optimized OS*) com um tamanho total de três *node's*.

A mesma configuração de máquina é aplicada aos três *node's* para todos os testes realizados.

Optou-se por utilizar o banco de dados *MySQL* por se tratar de um banco de dados gratuito de alto desempenho, e possuir mecanismos de alta disponibilidade, sendo trabalhado na versão 5.7.25.

Para a criação do grupo de replicação no *Kubernetes*, foi utilizado uma ferramenta denominada *KubeDB*, que tem como objetivo tornar a implementação de banco de dados em ambientes *Kubernetes* mais fácil e prática, uma melhor visão da ferramenta é abordada no Apêndice B.

Para o trabalho, utilizou-se a versão 0.12.0 do *KubeDB* para a replicação *master-slave*, que até o momento deste trabalho é a única disponível, sendo a implementação do *multi-master* prometido para versões futuras.

A escolha dessa ferramenta para utilização dos testes, se dá por se tratar de uma ferramenta *open source* e de fácil utilização, não sendo necessário entendimento especializado para sua utilização.

Para a realização do balanceamento de carga, a nível de aplicação, optou-se pela ferramenta *ProxySQL*, devido ao seu tempo de maturidade, trabalhar na camada de aplicação, possuir regras iniciais de segurança e se tratar de uma ferramenta *open source*.

No trabalho utilizou-se a versão 2.0.4 contido em uma imagem *Docker* disponibilizado na plataforma *Docker Hub*, a ferramenta é melhor descrita no Apêndice A.

A nuvem *Google Cloud*, foi escolhida para o trabalho devido a sua facilidade de uso, e criar um ambiente totalmente preparado para o uso sem a necessidade de configurar todo o *cluster*, outro motivo para a sua escolha foi devido a mesma

oferecer 300 dólares gratuitos para o primeiro cadastro, sendo esses créditos usados para realização dos testes.

Deve-se levar em consideração que toda a arquitetura montada foi apenas utilizada para demonstrar a criação de um ambiente de alta disponibilidade, sendo as configurações e recursos utilizados apenas uma forma de ilustrar uma configuração funcional, onde as ferramentas aqui utilizadas foram escolhidas por praticidade, para execução do objetivo final de apresentar o ambiente em operação.

## 4.1 EXPERIMENTO REALIZADO

Nesta seção será demonstrado casos de *failover* no *cluster* de alta disponibilidade, para a realização dos testes, foi criada uma arquitetura de grupo de replicação no *cluster Kubernetes* utilizando três configurações de arquitetura.

A primeira contendo três servidores, uma segunda arquitetura contendo cinco servidores e a última configuração contendo sete servidores.

Todas as configurações utilizaram como base a mesma quantidade de memória, armazenamento e processamento. As configurações foram montadas mantendo a quantidade de *failover* suportados, por estrutura, conforme Tabela 2.

Para a execução dos testes de funcionamento do *cluster*, foram utilizados dois *datasets* de banco de dados, um primeiro *dataset* para realização dos *inserts* simultâneos e outro *dataset* como forma de representar um banco de dados real que já possui muitos dados a serem preservados.

Os testes realizados estarão distribuídos em três seções, onde será apresentado uma ocorrência de *failover* em cada ambiente: Testes com três servidores, cinco e sete respectivamente.

Para cada teste aplicou-se uma arquitetura de armazenamento diferente, sendo elas:

- Um banco sem dados;
- Um banco com aproximadamente 47.2 megabytes contendo em torno de 1.267.354 registros;
- Um banco com aproximadamente 110.2 megabytes contendo em torno de 3.144.071 registros.



Também será apresentado o tempo que cada arquitetura demorou para eleger um novo mestre e voltar a disponibilizar alterações, como *inserts* ou *deletes*, sendo os *selects* usados para comprovar que o servidor continua operacional mesmo sem o servidor mestre.

#### 4.1.1 Arquitetura com três servidores

##### 4.1.1.1 Grupo de replicação sem dados

Na Figura 13, é ilustrado a estrutura da arquitetura antes do *failover* ocorrer, sendo listado qual o servidor mestre e como é formado o grupo de replicação.



```

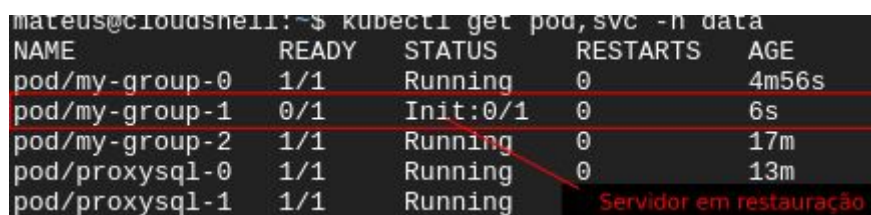
group_replication_primary_member | 3db89fe2-37ea-11ea-8daa-c6d43ccdedb2 |
row in set (0.24 sec)
mysql> select * from performance_schema.replication_group_members;

```

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	1a4cff23-37ea-11ea-9384-96e12d869bfe	my-group-0.my-group-gvr.data	3306	ONLINE
group_replication_applier	3db89fe2-37ea-11ea-8daa-c6d43ccdedb2	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	598fec4f-37ea-11ea-af7e-d2017c714939	my-group-2.my-group-gvr.data	3306	ONLINE

Figura 13 - Arquitetura do grupo (Três Servidores - Estrutura sem dados).

A Figura 14 representa o momento da ocorrência do *failover* onde o servidor mestre foi perdido e o *cluster* já identificou, e está em processo de restauração do servidor.



```

mateus@cloudshell:~$ kubectl get pod,svc -n data

```

NAME	READY	STATUS	RESTARTS	AGE
pod/my-group-0	1/1	Running	0	4m56s
pod/my-group-1	0/1	Init:0/1	0	6s
pod/my-group-2	1/1	Running	0	17m
pod/proxysql-0	1/1	Running	0	13m
pod/proxysql-1	1/1	Running		

Figura 14 - Servidor em recuperação após falha (Três Servidores - Estrutura sem dados).

A Figura 15, ilustra o tempo gasto pelo servidor para se recuperar da falha e voltar a execução de uma operação de alteração, no caso está sendo realizado uma operação de *delete* logo após o momento do *failover*, onde, a diferença entre a hora que ocorreu a falha e a da recuperação leva o tempo aproximado de 11 segundos, conforme destacado em verde.

88	20:13:49	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
89	20:13:50	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	Error Code: 2013. L
90	20:14:01	delete from dataset.mydataset	86 row(s) affected
91	20:14:02	show status like '%primary%'	1 row(s) returned

**Figura 15 - Tempo gasto para eleição de novo mestre e execução de operação (Três Servidores - Estrutura sem dados).**

Após todo o processo de recuperação, nota-se na Figura 16 a nova estrutura do grupo de replicação, contendo um novo mestre e com toda a arquitetura em funcionamento.

```
mysql> show status like '%primary%';
```

Variable_name	Value
group_replication_primary_member	1a4cff23-37ea-11ea-9384-96e12d869bfe

1 row in set (0.25 sec)

```
mysql> select * from performance_schema.replication_group_members;
```

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	1a4cff23-37ea-11ea-9384-96e12d869bfe	my-group-0.my-group-gvr.data	3306	ONLINE
group_replication_applier	3db89fe2-37ea-11ea-8daa-c6d43ccdedb2	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	598fec4f-37ea-11ea-af7e-d2017c714939	my-group-2.my-group-gvr.data	3306	ONLINE

3 rows in set (0.25 sec)

**Figura 16 - Grupo após recuperação de *failover* (Três Servidores - Estrutura sem dados).**

#### 4.1.1.2 Grupo de replicação com tamanho de 47.2MB

Após a realização da inserção de informações ao banco de dados, foi realizado uma nova preparação para ocorrência de *failover*, a Figura 17 ilustra a arquitetura antes do processo de failover.

```
group_replication_primary_member | 9cae502b-3977-11ea-839d-86d7e67ed502 |
```

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	9cae502b-3977-11ea-839d-86d7e67ed502	my-group-0.my-group-gvr.data	3306	ONLINE
group_replication_applier	c0f0efd6-3977-11ea-afc2-c2c130ee2f55	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	dac8421b-3977-11ea-a493-aa46d02b1d05	my-group-2.my-group-gvr.data	3306	ONLINE

**Figura 17 - Servidores antes da ocorrência de *failover* (Três Servidores - Dados com 47.2MB).**

Com o grupo preparado foi realizado um novo *failover* na arquitetura, neste teste foi possível capturar o *status* do grupo no período de *failover*, o mesmo é ilustrado na Figura 18, onde é possível identificar que o grupo removeu o antigo servidor mestre da lista de servidores que participam do grupo, ao realizar essa operação o grupo escolhe entre os dois membros restante, o novo mestre.

```
mysql> Select @@hostname;
ERROR 2013 (HY000): Lost connection to MySQL server during query
mysql> show status like '%primary%';
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id: 5
Current database: *** NONE ***
```

Variable_name	Value
group_replication_primary_member	c0f0efd6-3977-11ea-afc2-c2c130ee2f55

```
1 row in set (2.98 sec)
```

```
mysql> select * from performance_schema.replication_group_members;
```

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	c0f0efd6-3977-11ea-afc2-c2c130ee2f55	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	dac8421b-3977-11ea-a493-aa46d02b1d05	my-group-2.my-group-gvr.data	3306	ONLINE

Figura 18 - Status do grupo no momento do *failover* (Três Servidores - Dados com 47.2MB).

A Figura 19, ilustra o tempo gasto para o grupo identificar um novo mestre e aceitar operações de alteração, neste teste em questão, alterou-se a ordem das operações, realizando uma operação de *select* e logo após uma operação de *delete* na mesma transação, onde, o comando *select* foi realizado no momento do *failover* e o comando *delete* após a recuperação.

180	20:17:09	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
181	20:17:10	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	Error Code: 2013. Lo
182	20:17:21	Select @@hostname	1 row(s) returned
183	20:17:21	show status like '%primary%'	1 row(s) returned
184	20:17:21	select * from performance_schema.replication_group_members	2 row(s) returned
185	20:17:22	select count(emp_no) from employees.salaries	1 row(s) returned
186	20:17:39	delete from dataset.mydataset	167 row(s) affected

Figura 19 - Ocorrência do *failover* (Três Servidores - Dados com 47.2MB).

Na Figura 19 destaca-se em vermelho o momento da ocorrência da falha e o início do processo de *failover*, onde perdeu-se a transação de inserção derrubando a conexão ao servidor.

Em amarelo é destacado o tempo para que fosse realizado uma nova abertura de transação para a conexão com o servidor, nota-se que para a realização de um *select* levou-se aproximadamente 11 segundos, sendo esse tempo destacado em verde no registro 182.

Ainda na Figura 19, para a realização da remoção dos dados pelo comando *delete*, foram necessários 29 segundos, conforme destacado em verde no registro

186, na Figura 20 obtém-se o *status* do servidor após a ocorrência da recuperação da arquitetura.

```
mysql> show status like '%primary%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| group_replication_primary_member | c0f0efd6-3977-11ea-afc2-c2c130ee2f55 |
+-----+-----+
1 row in set (0.00 sec)

mysql> select * from performance_schema.replication_group_members;
+-----+-----+-----+-----+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+-----+-----+-----+-----+
| group_replication_applier | 9cae502b-3977-11ea-839d-86d7e67ed502 | my-group-0.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | c0f0efd6-3977-11ea-afc2-c2c130ee2f55 | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | dac8421b-3977-11ea-a493-aa46d02b1d05 | my-group-2.my-group-gvr.data | 3306 | ONLINE |
+-----+-----+-----+-----+-----+
```

Figura 20 - *Status* do grupo após recuperação de falha (Três Servidores - Dados com 47.2MB).

#### 4.1.1.3 Grupo de replicação com tamanho de 110.2MB

Para essa seção analisou-se o tempo gasto para restauração das execuções de alteração, e a consistência do antigo servidor mestre após ingressar novamente no grupo depois da ocorrência da falha.

Na Figura 21, é ilustrado uma transação de *select* no momento que ocorreu o *failover*, fazendo com que o balanceamento de carga direcione a execução do comando para os servidores de leitura.

```
mysql> select @@hostname
+-----+
| @@hostname |
+-----+
| my-group-0 |
+-----+
1 row in set (0.17 sec)

mysql> show status like 'primary%'
+-----+-----+
| Variable_name | Value |
+-----+-----+
| group_replication_primary_member | c0f0efd6-3977-11ea-afc2-c2c130ee2f55 |
+-----+-----+
1 row in set (0.20 sec)

mysql> select * from performance_schema.replication_group_members
+-----+-----+-----+-----+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+-----+-----+-----+-----+
| group_replication_applier | 9cae502b-3977-11ea-839d-86d7e67ed502 | my-group-0.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | c0f0efd6-3977-11ea-afc2-c2c130ee2f55 | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | dac8421b-3977-11ea-a493-aa46d02b1d05 | my-group-2.my-group-gvr.data | 3306 | UNREACHABLE |
+-----+-----+-----+-----+-----+
3 rows in set (0.22 sec)

mysql> select count(emp_no) from employees.salaries
+-----+
| count(emp_no) |
+-----+
| 2844047 |
+-----+
1 row in set (0.85 sec)
```

Figura 21 - Consulta no momento do *failover* (Três servidores - Dados com 110.2MB).



Na Figura 21 é possível identificar o *status* atual do antigo servidor mestre a qual está ingressando novamente na arquitetura, porém ainda não está totalmente disponível para utilização.

Na Figura 22, destaca-se em vermelho o momento da ocorrência da falha e processo de *failover*, onde ocorre a perda da transação. Em amarelo é destacado o tempo gasto para o servidor conseguir realizar uma nova bateria de inserções no novo servidor mestre.

Destacado em verde na Figura 22, é ilustrado a hora em que os comandos foram executados, sendo sua diferença o tempo gasto para a recuperação da falha, que no caso foi de aproximadamente 15 segundos.

✓	12...	21:42:28	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✗	12...	21:42:29	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	Error Code: 2013. Lost connection
✓	12...	21:42:44	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✓	12...	21:42:44	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected

**Figura 22 - Ocorrência de falha no grupo (Três servidores - Dados com 110.2MB).**

Com a recuperação bem sucedida analisou-se o mestre antigo após ingressar novamente no grupo, quanto a sua consistência após sua recuperação, tal informação é destacada na Figura 23, onde, é possível identificar que o servidor foi recuperado totalmente da falha, estando consistente em relação aos demais servidores, listados na consulta da Figura 21.

```

mysql> show status like '%primary%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| group_replication_primary_member | c0f0efd6-3977-11ea-afc2-c2c130ee2f55 |
+-----+-----+
1 row in set (0.01 sec)

```

**Servidor primário**

```

mysql> select * from performance_schema.replication_group_members;
+-----+-----+-----+-----+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+-----+-----+-----+-----+
| group_replication_applier | 9cae502b-3977-11ea-839d-86d7e67ed502 | my-group-0.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | c0f0efd6-3977-11ea-afc2-c2c130ee2f55 | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | dac8421b-3977-11ea-a493-aa46d02b1d05 | my-group-2.my-group-gvr.data | 3306 | ONLINE |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

```

**Grupo de replicação**

```

mysql> select count(emp_no) from employees.salaries;
+-----+
| count(emp_no) |
+-----+
| 2844047 |
+-----+
1 row in set (1.21 sec)

```

**Quantidade de dados após recuperação**

**Figura 23 - Consistência em servidor recuperado após *failover* (Três servidores - Dados com 110.2MB).**

## 4.1.2 Arquitetura com cinco servidores

Para a realização dos testes, diferente da seção 4.1.1, foi realizado dois *failover* simultâneos dentro do grupo atendendo à Tabela 2, quanto a quantidade máxima de *failover* suportado.

### 4.1.2.1 Grupo de replicação sem dados

Na Figura 24 é ilustrado o grupo de replicação com cinco servidores antes da ocorrência de falha.

```

> mysql -u master -p -h 35.223.214.175 -P6033 -e "Select @@hostname;
show status like '%primary%';
select * from performance_schema.replication_group_members;"
Enter password:
+-----+
| @@hostname |
+-----+
| my-group-0 |
+-----+
+-----+-----+
| Variable_name | Value |
+-----+-----+
| group_replication_primary_member | 1e69e731-39e8-11ea-890a-2a985cbae710 |
+-----+-----+
+-----+-----+-----+-----+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+-----+-----+-----+-----+
| group_replication_applier | 1e69e731-39e8-11ea-890a-2a985cbae710 | my-group-0.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 4077dcda-39e8-11ea-8c08-c6e0b1406b49 | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 5bae8e6d-39e8-11ea-b851-fa5c762672fe | my-group-2.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 757a9eac-39e8-11ea-b657-f2065b93f641 | my-group-3.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 90464e20-39e8-11ea-9354-a6cf5d00d3d3 | my-group-4.my-group-gvr.data | 3306 | ONLINE |
+-----+-----+-----+-----+-----+

```

Servidor Primário

Grupo de replicação

Figura 24 - Grupo de replicação (Cinco servidores - Estrutura sem dados).

Na Figura 25 é dada a execução dos comandos realizados para executar o *failover*, inicialmente foi realizado a operação no servidor primário, com o servidor primário inativo a arquitetura realizou o primeiro processo de *failover*, e com a eleição do novo primário, realizou-se a segunda execução do comando.

```

mateus@cloudshell:~$ kubectl delete pod my-group-0 -n data
pod "my-group-0" deleted
mateus@cloudshell:~$ kubectl delete pod my-group-1 -n data
pod "my-group-1" deleted
mateus@cloudshell:~$ kubectl get pod,svc -n data
NAME          READY   STATUS    RESTARTS   AGE
pod/my-group-0 0/1     Running   0           28s
pod/my-group-2 1/1     Running   0           21m
pod/my-group-3 1/1     Running   0           21m
pod/my-group-4 1/1     Running   0           20m
pod/proxysql-0 1/1     Running   0           16m
pod/proxysql-1 1/1     Running   0           12m

NAME          TYPE          CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/my-group  ClusterIP    10.12.8.147  <none>        3306/TCP         23m
service/my-group-gvr  ClusterIP    None         <none>        3306/TCP         23m
service/proxysql    LoadBalancer 10.12.2.71   35.223.214.175 6033:30033/TCP,6032:30032/TCP 16m

mateus@cloudshell:~$ kubectl get pod,svc -n data
NAME          READY   STATUS    RESTARTS   AGE
pod/my-group-0 0/1     Running   1           58s
pod/my-group-2 1/1     Running   0           22m
pod/my-group-3 1/1     Running   0           21m
pod/my-group-4 1/1     Running   0           20m
pod/proxysql-0 1/1     Running   0           16m
pod/proxysql-1 1/1     Running   0           12m

```

Figura 25 - Derrubando os servidores primários (Cinco servidores - Estrutura sem dados).

Com o processo de queda do servidor, realizou-se uma verificação do *status* do grupo durante o processo de recuperação, identificando a remoção dos dois servidores primários do grupo de replicação, tal ação é ilustrada pelas Figuras 26 e 27.

Variable_name	Value
group_replication_primary_member	4077dcda-39e8-11ea-8c08-c6e0b1406b49

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	4077dcda-39e8-11ea-8c08-c6e0b1406b49	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	5bae8e6d-39e8-11ea-b851-fa5c762672fe	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	757a9eac-39e8-11ea-b657-f2065b93f641	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	90464e20-39e8-11ea-9354-a6cf5d00d3d3	my-group-4.my-group-gvr.data	3306	ONLINE

Figura 26 - Status do grupo na primeira falha (Cinco servidores - Estrutura sem dados).

Variable_name	Value
group_replication_primary_member	5bae8e6d-39e8-11ea-b851-fa5c762672fe

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	5bae8e6d-39e8-11ea-b851-fa5c762672fe	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	757a9eac-39e8-11ea-b657-f2065b93f641	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	90464e20-39e8-11ea-9354-a6cf5d00d3d3	my-group-4.my-group-gvr.data	3306	ONLINE

Figura 27 - Status do grupo na segunda falha (Cinco servidores - Estrutura sem dados).

Como forma de identificar o tempo gasto para a arquitetura estar novamente com um servidor mestre, foi capturado o último *failover* entre as ocorrências de falha, conforme Figura 28.

#	Time	Action	Message
284	09:08:45	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
285	09:08:45	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
286	09:08:45	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	Error Code: 2013. Lost connection to MySQL server d...
287	09:08:56	delete from dataset.mydataset	278 row(s) affected
288	09:09:35	show status like '%primary%'	1 row(s) returned
289	09:09:36	select * from performance_schema.replication_group_members	3 row(s) returned
290	09:10:54	select * from performance_schema.replication_group_members	5 row(s) returned

**Figura 28 - Recuperação após *failover* (Cinco servidores - Estrutura sem dados).**

Na Figura 28, destacado em verde, é possível identificar que o tempo para o servidor eleger um novo primário após o *failover* foi de aproximadamente 11 segundos, sendo este o tempo para que pudesse novamente realizar operações de alteração no banco de dados.

Destacado de azul identifica-se dois comandos de consulta, sendo no primeiro retornado apenas três linhas onde representam a quantidade de servidores no momento da consulta.

No segundo comando, é retornado cinco linhas, representando que a arquitetura está totalmente recuperada, sendo o tempo para a recuperação total da arquitetura de aproximadamente dois minutos, conforme destacado em amarelo no registro 290 em relação ao registro 286.

#### 4.1.2.2 Grupo de replicação com tamanho de 47.2MB

Na Figura 29 é ilustrado o *status* do grupo de replicação antes da ocorrência do teste de *failover*.



Variable_name	Value
group_replication_primary_member	5bae8e6d-39e8-11ea-b851-fa5c762672fe

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	1e69e731-39e8-11ea-890a-2a985cbae710	my-group-0.my-group-gvr.data	3306	ONLINE
group_replication_applier	4077dcda-39e8-11ea-8c08-c6e0b1406b49	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	5bae8e6d-39e8-11ea-b851-fa5c762672fe	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	757a9eac-39e8-11ea-b657-f2065b93f641	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	90464e20-39e8-11ea-9354-a6cf5d00d3d3	my-group-4.my-group-gvr.data	3306	ONLINE

Figura 29 - Status do grupo de replicação (Cinco servidores - Dados com 47.2MB).

Com o grupo em operação foi realizado o *failover* da arquitetura, sendo representadas pelas Figuras 30 e 31 respectivamente, na primeira imagem é dado o tempo para recuperação no primeiro *failover*, já na segunda imagem é dado o tempo de recuperação para o segundo *failover*, ambos destacados em verde, sendo o tempo para recuperação de aproximadamente 10 e 15 segundos respectivamente.

✓	645	09:35:43	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✓	646	09:35:43	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✓	647	09:35:43	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✗	648	09:35:44	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	Error Code: 2013: Lost connection to MySQL server d...
✓	649	09:35:54	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✓	650	09:35:54	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✓	651	09:35:55	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected

Figura 30 - Primeiro *failover* (Cinco servidores - Dados com 47.2MB).

✓	692	09:36:06	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✓	693	09:36:06	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✗	694	09:36:07	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	Error Code: 2013: Lost connection to MySQL server d...
✓	695	09:36:22	delete from dataset.mydataset	400 row(s) affected

Figura 31 - Segundo *failover* (Cinco servidores - Dados com 47.2MB).

Na Figura 32, é representado o grupo de replicação após a ocorrência de falha e recuperação total da arquitetura.

```

row in set (0.00 sec)
mysql> show status like '%primary%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| group_replication_primary_member | 4077dcda-39e8-11ea-8c08-c6e0b1406b49 |
+-----+-----+
row in set (0.00 sec)

mysql> select * from performance_schema.replication_group_members;
+-----+-----+-----+-----+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STA |
+-----+-----+-----+-----+-----+
| group_replication_applier | 1e69e731-39e8-11ea-890a-2a985cbae710 | my-group-0.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 4077dcda-39e8-11ea-8c08-c6e0b1406b49 | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 5bae8e6d-39e8-11ea-b851-fa5c762672fe | my-group-2.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 757a9eac-39e8-11ea-b657-f2065b93f641 | my-group-3.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 90464e20-39e8-11ea-9354-a6cf5d00d3d3 | my-group-4.my-group-gvr.data | 3306 | ONLINE |
+-----+-----+-----+-----+-----+

```

**Primário**

**Grupo de replicação**

Figura 32 - Status do grupo após recuperação de falha (Cinco servidores - Dados com 47.2MB).

#### 4.1.2.3 Grupo de replicação com tamanho de 110.2MB

Na Figura 33 é ilustrado o grupo antes da carga de teste, listando o grupo antes da ocorrência das falhas.

```

@@hostname |
my-group-3 |
+-----+-----+
| Variable_name | Value |
+-----+-----+
| group_replication_primary_member | 02dc8d49-4aa0-11ea-b3d5-561689acc35a |
+-----+-----+

CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+-----+-----+-----+-----+
| group_replication_applier | 02dc8d49-4aa0-11ea-b3d5-561689acc35a | my-group-0.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 24519fb4-4aa0-11ea-9ba6-9a58e682da06 | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 3fca8cfc-4aa0-11ea-876f-26eb5df28eb4 | my-group-2.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 5d086e83-4aa0-11ea-8e12-ced7ad178a6a | my-group-3.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 791ad1c0-4aa0-11ea-83d8-161aa2682b85 | my-group-4.my-group-gvr.data | 3306 | ONLINE |
+-----+-----+-----+-----+-----+

count(emp_no) |
2844047 |

```

**Servidor Primário**

**Grupo de replicação**

**Total de dados armazenados em uma das tabelas do banco**

Figura 33 - Status da arquitetura antes da ocorrência das falhas (Cinco servidores - Dados com 110.2MB).

Nas Figuras 34 e 35 é ilustrado o status do grupo no momento da primeira falha e o tempo de failover para a eleição do mestre, sendo o tempo para recuperação de falhas, destacado em verde, de aproximadamente 16 segundos.

```

Ocorrência da falha
87 17:17:57 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,207 sec
88 17:17:57 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,392 sec
89 17:17:58 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... Error Code: 2013. Lost connection to MySQL server d... 0,384 sec
90 17:18:14 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,199 sec
91 17:18:15 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,204 sec

```

**Failover bem sucedido**

Figura 34 - Ocorrência da falha e failover bem sucedido (Cinco servidores - Dados com 110.2MB).

```

> mysql -u master -p -h35.193.164.135 -P6033 -e "Select @@hostname;
show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(emp_no) from employees.salaries;"

Enter password:
+-----+
| @@hostname |
+-----+
| my-group-2 |
+-----+

Variable_name      Value
+-----+-----+
| group_replication_primary_member | 24519fb4-4aa0-11ea-9ba6-9a58e682da06 |
+-----+-----+

CHANNEL_NAME      MEMBER_ID          MEMBER_HOST          MEMBER_PORT  MEMBER_STATE
+-----+-----+-----+-----+-----+
| group_replication_applier | 24519fb4-4aa0-11ea-9ba6-9a58e682da06 | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 3fca8cfc-4aa0-11ea-876f-26eb5df28eb4 | my-group-2.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 5d086e83-4aa0-11ea-8e12-ced7ad178a6a | my-group-3.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 791ad1c0-4aa0-11ea-83d8-161aa2682b85 | my-group-4.my-group-gvr.data | 3306 | ONLINE |
+-----+-----+-----+-----+-----+

count(emp_no)
+-----+
| 2844047 |
+-----+

~/Documentos/TCC/sql/test_db-master 15s 17:18:10

```

**Figura 35 - Status do grupo no momento da primeira falha (Cinco servidores - Dados com 110.2MB).**

Nas Figuras 36 e 37 são ilustrados o tempo para a ocorrência do *failover* e o *status* do grupo no momento da segunda falha da arquitetura respectivamente, sendo o tempo para o *failover*, destacado em verde entre o registro 114 e 116, de aproximadamente 12 segundos.

```

Início do failover
✓ 112 17:18:25 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,217 sec
✓ 113 17:18:25 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,171 sec
✗ 114 17:18:25 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... Error Code: 2013. Lost connection to MySQL server d... 0,269 sec
✓ 115 17:18:37 use dataset 0 row(s) affected 0,199 sec
✓ 116 17:18:37 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,167 sec
Failover bem sucedido

```

**Figura 36 - Ocorrência de falha e *failover* bem sucedido (Cinco servidores - Dados com 110.2MB).**



```

@@hostname |
my-group-3 |
Variable_name | Value |
group_replication_primary_member | 3fca8cfc-4aa0-11ea-876f-26eb5df28eb4 |
CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
group_replication_applier | 3fca8cfc-4aa0-11ea-876f-26eb5df28eb4 | my-group-2.my-group-gvr.data | 3306 | ONLINE |
group_replication_applier | 5d086e83-4aa0-11ea-8e12-ced7ad178a6a | my-group-3.my-group-gvr.data | 3306 | ONLINE |
group_replication_applier | 791ad1c0-4aa0-11ea-83d8-161aa2682b85 | my-group-4.my-group-gvr.data | 3306 | ONLINE |
count(emp_no) |
2844047 |

```

Servidor primário

Grupo de replicação

**Figura 37 - Status do servidor no momento da segunda falha (Cinco servidores - Dados com 110.2MB).**

Na Figura 38 é ilustrado o grupo totalmente recuperado, após a ocorrência de todos os testes na arquitetura.

```

@@hostname |
my-group-0 |
Variable_name | Value |
group_replication_primary_member | 3fca8cfc-4aa0-11ea-876f-26eb5df28eb4 |
CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
group_replication_applier | 02dc8d49-4aa0-11ea-b3d5-561689acc35a | my-group-0.my-group-gvr.data | 3306 | ONLINE |
group_replication_applier | 24519fb4-4aa0-11ea-9ba6-9a58e682da06 | my-group-1.my-group-gvr.data | 3306 | ONLINE |
group_replication_applier | 3fca8cfc-4aa0-11ea-876f-26eb5df28eb4 | my-group-2.my-group-gvr.data | 3306 | ONLINE |
group_replication_applier | 5d086e83-4aa0-11ea-8e12-ced7ad178a6a | my-group-3.my-group-gvr.data | 3306 | ONLINE |
group_replication_applier | 791ad1c0-4aa0-11ea-83d8-161aa2682b85 | my-group-4.my-group-gvr.data | 3306 | ONLINE |
count(emp_no) |
2844047 |

```

**Figura 38 - Grupo recuperado após testes de *failover* (Cinco servidores - Dados com 110.2MB).**

### 4.1.3 Arquitetura com sete servidores

Para a realização dos testes, foi realizado três *failovers* consecutivos nos servidores, respeitando o total de falhas suportadas para essa arquitetura, conforme Tabela 2.

Como complemento ao teste abordou-se nesta seção como a arquitetura se comporta em casos onde está ocorrendo uma demanda de inserções e ocorre uma falha no servidor.

### 4.1.3.1 Grupo de replicação sem dados

Na Figura 39 é ilustrado a estrutura inicial do grupo e a quantidade de dados iniciais na tabela que irá receber a carga de inserções.

```

mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;"
Enter password:

```

Variable_name	Value
group_replication_primary_member	25404485-39fa-11ea-9470-aaea0f05519c

**Servidor primário**

**Grupo de replicação**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	25404485-39fa-11ea-9470-aaea0f05519c	my-group-0.my-group-gvr.data	3306	ONLINE
group_replication_applier	46701c20-39fa-11ea-990b-86b88649644d	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

count(id)
0

**Quantidade de dados inseridos na tabela mydataset**

Figura 39 - Estrutura inicial com sete servidores (Sete servidores - Estrutura sem dados).

Nas Figuras 40 e 41 são ilustrados o primeiro *failover* e a quantidade de dados no momento da falha respectivamente, sendo o tempo de execução para o servidor eleger o novo mestre antes da segunda falha, destacado em verde, de aproximadamente 15 segundos.

```

Ocorrência da falha
Failover bem sucedido

```

✓	54	11:12:51	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected
✗	55	11:12:51	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... Error Code: 2013. Lost connection to MySQL server d...
✓	56	11:13:04	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected
✓	57	11:13:05	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected
✓	58	11:13:05	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected

Figura 40 - Primeiro *failover* na arquitetura (Sete servidores - Estrutura sem dados).

```

> mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;"
Enter password:

```

Variable_name	Value
group_replication_primary_member	46701c20-39fa-11ea-990b-86b88649644d

**Servidor primário**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	46701c20-39fa-11ea-990b-86b88649644d	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

**Grupo de replicação**

count(id)
50

**Quantidade total inserida antes do proximo failover.**

**50**

**Figura 41 - Quantidade de dados inseridos antes da segunda falha (Sete servidores - Estrutura sem dados).**

Nas Figuras 42 e 43 são ilustrados o grupo de replicação no momento da segunda falha, exibindo o tempo para recuperação e a quantidade de dados armazenados antes da terceira falha ocorrer respectivamente.

```

Ocorrência da falha

```

✓	108	11:13:21	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✗	109	11:13:21	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	Error Code: 2013. Lost connection to MySQL server d...
✓	110	11:13:33	use dataset	0 row(s) affected
✓	111	11:13:34	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✓	112	11:13:34	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected

**Failover bem sucedido**

**Figura 42 - Segunda falha e failover bem sucedido (Sete servidores - Estrutura sem dados).**

Na Figura 42, destacado de verde, ilustra a hora da ocorrência da falha no registro 109 e a execução de um comando de alteração no registro 111, sendo o tempo do failover a diferença entre as horas das execuções, que foi de aproximadamente 13 segundos.



```

mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;"
Enter password:

```

Variable_name	Value
group_replication_primary_member	61b47e42-39fa-11ea-bc5e-f6055af10854

Servidor primário

Grupo de replicação

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

Quantidade de dados inseridos ate o momento da terceira falha

count(id)
103

103

Figura 43 - Quantidade de dados armazenados antes da terceira falha ocorrer (Sete servidores - Estrutura sem dados).

Nas Figuras 44 e 45, são retratados o tempo e a quantidade armazenada na ocorrência do terceiro processo de *failover* respectivamente, onde o tempo gasto para eleger um novo mestre no failover, destacado de verde, foi de aproximadamente 12 segundos.

```

Ocorrência da falha
175 11:13:46 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected
176 11:13:46 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected
177 11:13:47 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... Error Code: 2013. Lost connection to MySQL server d...
178 11:13:59 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected
179 11:13:59 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected
180 11:13:59 INSERT INTO mydataset(id,track_name,size bytes,app_desc) V... 1 row(s) affected

```

Failover bem sucedido

Figura 44 - Terceira falha e *failover* bem sucedido (Sete servidores - Estrutura sem dados).

```

select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;"
Enter password:

```

Variable_name	Value
group_replication_primary_member	7b566992-39fa-11ea-a06b-3a6ba9e39f5e

Servidor primário

Grupo de replicação

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

Quantidade total de dados inseridas, mesmo na ocorrência de falha

count(id)
169

169

Figura 45 - Quantidade total de dados inseridos mesmo com o servidor em falha (Sete servidores - Estrutura sem dados).

Na Figura 45, é possível identificar que mesmo na ocorrência de três falhas consecutivas, a arquitetura conseguiu continuar as inserções de dados e manter a consistência e alta disponibilidade da arquitetura.

#### 4.1.3.2 Grupo de replicação com tamanho de 47.2MB

Na Figura 46 é ilustrada a arquitetura antes do início dos testes de *failover*.

```

mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;
select count(emp_no) from employees.salaries;"
Enter password:

```

Variable_name	Value
group_replication_primary_member	25404485-39fa-11ea-9470-aaea0f05519c

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	25404485-39fa-11ea-9470-aaea0f05519c	my-group-0.my-group-gvr.data	3306	ONLINE
group_replication_applier	46701c20-39fa-11ea-990b-86b88649644d	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

count(id)	0
count(emp_no)	967330

**Figura 46 - Grupo de replicação antes dos testes de *failover* (Sete servidores - Dados com 47.2MB).**

Nas Figuras 47 e 48, são ilustrados o *status* do grupo e o tempo para a execução do *failover* respectivamente, sendo o tempo, destacado em verde na Figura 48, de aproximadamente 10 segundos para a arquitetura voltar a operação.



```

mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;
select count(emp_no) from employees.salaries;"
Enter password:

```

Variable_name	Value
group_replication_primary_member	46701c20-39fa-11ea-990b-86b88649644d

**Servidor primário**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	46701c20-39fa-11ea-990b-86b88649644d	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

**Grupo de replicação**

count(id)
60

**Quantidade de dados inseridas no primeiro failover**

count(emp_no)
967330

**Quantidade de dados em uma das tabelas no primeiro failover**

Figura 47 - Status do grupo na primeira execução do *failover* (Sete servidores - Dados com 47.2MB).

```

Ocorrência da falha.
323 11:44:15 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected
324 11:44:15 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... Error Code: 2013. Lost connection to MySQL server d...
325 11:44:25 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected

```

**Failover bem sucedido.**

Figura 48 - Tempo para ocorrência do *failover* (Sete servidores - Dados com 47.2MB).

As Figuras 49 e 50 ilustram a segunda ocorrência de falha e status do grupo respectivamente, sendo o tempo para a execução do *failover*, destacado em verde, de aproximadamente 12 segundos.

```

Momento inicial de execução do failover
343 11:44:28 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected
344 11:44:28 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... Error Code: 2013. Lost connection to MySQL server d...
346 11:44:40 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected

```

**Servidor primário operacional**

Figura 49 - Momento da ocorrência da falha e início do *failover* (Sete servidores - Dados com 47.2MB).

```

mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;
select count(emp_no) from employees.salaries;"
Enter password:

```

**Servidor primário**

Variable_name	Value
group_replication_primary_member	61b47e42-39fa-11ea-bc5e-f6055af10854

**Grupo de replicação**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

```

count(id)
79
count(emp_no)
967330

```

**Quantidade de dados inseridas no momento da segunda falha**  
79

**Quantidade já existente em uma das tabelas do banco**  
967330

Figura 50 - Status do grupo no momento da ocorrência da segunda falha (Sete servidores - Dados com 47.2MB).

As Figuras 51 e 52 ilustram a terceira ocorrência de falha e o status do grupo no momento da execução do *failover* respectivamente, sendo o tempo total para recuperação dos dados, destacados de verde, de aproximadamente 11 segundos.

```

Momento inicial do failover

```

✓	355	11:44:42	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected
✗	356	11:44:42	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	Error Code: 2013: Lost connection to MySQL server d...
✓	357	11:44:53	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected

**Recuperação bem sucedida**

Figura 51 - Terceiro *failover* realizado com sucesso (Sete servidores - Dados com 47.2MB).

```

mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;
select count(emp_no) from employees.salaries;"
Enter password:

```

**Servidor primário**

Variable_name	Value
group_replication_primary_member	7b566992-39fa-11ea-a06b-3a6ba9e39f5e

**Grupo de replicação**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

```

count(id)
89
count(emp_no)
967330

```

**Quantidade de dados inseridas na ocorrência do terceiro failover**  
89

**Quantidade total em uma das tabelas da arquitetura**  
967330

Figura 52 - Status do grupo no momento da terceira execução do *failover* (Sete servidores - Dados com 47.2MB).



A Figura 53 ilustra o *status* do grupo após recuperação total dos servidores, retomando a arquitetura inicial com outro primário.

```

> mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;
select count(emp_no) from employees.salaries;"
Enter password:

```

Variable_name	Value
group_replication_primary_member	7b566992-39fa-11ea-a06b-3a6ba9e39f5e

**Servidor primário**

**Grupo totalmente recuperado**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	25404485-39fa-11ea-9470-aaea0f05519c	my-group-0.my-group-gvr.data	3306	ONLINE
group_replication_applier	46701c20-39fa-11ea-990b-86b88649644d	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

```

count(id)
-----
112

```

**Total de dados inseridos na arquitetura**

**112**

```

count(emp_no)
-----
967330

```

**Quantidade total de dados em uma das tabelas da arquitetura**

**967330**

Figura 53 - Arquitetura totalmente recuperada após ocorrência de falha (Sete servidores - Dados com 47.2MB).

#### 4.1.3.3 Grupo de replicação com tamanho de 110.2MB

Na Figura 54 é ilustrado a estrutura do grupo de alta disponibilidade antes dos testes de *failover*.

```

> mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;
select count(emp_no) from employees.salaries;"
Enter password:

```

Variable_name	Value
group_replication_primary_member	7b566992-39fa-11ea-a06b-3a6ba9e39f5e

**Servidor primário**

**Grupo de replicação**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	25404485-39fa-11ea-9470-aaea0f05519c	my-group-0.my-group-gvr.data	3306	ONLINE
group_replication_applier	46701c20-39fa-11ea-990b-86b88649644d	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	7b566992-39fa-11ea-a06b-3a6ba9e39f5e	my-group-3.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

```

count(id)
-----
0

```

**Quantidade de dados iniciais na tabela de carga de inserção**

**0**

```

count(emp_no)
-----
2844047

```

**Quantidade total em uma das tabelas do banco de dados**

**2844047**

Figura 54 - Grupo antes dos testes de *failover* (Sete servidores - Dados com 110.2MB).

Nas Figuras 55 e 56 são ilustrados o servidor em processo de *failover* e o total de inserções no momento do processo de recuperação, para a primeira ocorrência de falha.

Sendo o tempo para o processo de *failover*, destacado em verde na Figura 55, de aproximadamente 10 segundos.

**Momento inicial do processo de failover**

✓	468	12:40:19	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected	0,245 sec
✓	469	12:40:19	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected	0,212 sec
✗	470	12:40:19	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	Error Code: 2013. Lost connection to MySQL server d...	0,372 sec
✓	471	12:40:29	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected	0,199 sec
✓	472	12:40:29	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected	0,172 sec
✓	473	12:40:29	INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V...	1 row(s) affected	0,238 sec

**Failover bem sucedido**

Figura 55 - Servidor em processo de *failover* (Sete servidores - Dados com 110.2MB).

```

mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;
select count(emp_no) from employees.salaries;"
Enter password:

```

**Servidor primário**

Variable_name	Value
group_replication_primary_member	25404485-39fa-11ea-9470-aaea0f05519c

**Grupo de replicação**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	25404485-39fa-11ea-9470-aaea0f05519c	my-group-0.my-group-gvr.data	3306	ONLINE
group_replication_applier	46701c20-39fa-11ea-990b-86b88649644d	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

**Quantidade de dados inseridos no momento da primeira falha**  
85

**Quantidade total de dados em uma das tabelas do banco**  
2844047

Figura 56 - Status do servidor no momento da primeira falha (Sete servidores - Dados com 110.2MB).

As Figuras 57 e 58 retratam o momento do processo de *failover* e o *status* do grupo no segundo processo de falha da arquitetura, sendo o tempo para a realização bem sucedida do *failover*, destacado em verde, de aproximadamente 13 segundos.



**Servidor em processo de failover**

```

✓ 490 12:40:35 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,307 sec
✓ 491 12:40:35 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,407 sec
✓ 492 12:40:36 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,410 sec
✗ 493 12:40:36 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... Error Code: 2013. Lost connection to MySQL server d... 0,313 sec
✓ 494 12:40:49 use dataset 0 row(s) affected 0,191 sec
✓ 495 12:40:49 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,165 sec
✓ 496 12:40:49 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,270 sec

```

**Failover bem sucedido**

Figura 57 - Processo de *failover* em execução (Sete servidores - Dados com 110.2MB).

```

~
> mysql -u master -p -h 34.66.3.146 -P6033 -e "show status like '%primary%';
select * from performance_schema.replication_group_members;
select count(id) from dataset.mydataset;
select count(emp_no) from employees.salaries;"
Enter password:

```

**Servidor primário**

Variable_name	Value
group_replication_primary_member	46701c20-39fa-11ea-990b-86b88649644d

**Grupo de replicação**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	46701c20-39fa-11ea-990b-86b88649644d	my-group-1.my-group-gvr.data	3306	ONLINE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

```

+-----+
| count(id) |
+-----+
| 107 |
+-----+

```

**Quantidade total de inserções no momento da segunda falha**  
107

```

+-----+
| count(emp_no) |
+-----+
| 2844047 |
+-----+

```

**Quantidade total de dados em uma das tabelas do banco**  
2844047

Figura 58 - Status do grupo no momento da segunda falha (Sete servidores - Dados com 110.2MB).

Nas Figuras 59 e 60, são retratados o terceiro processo de *failover* e o status do grupo no momento de sua execução respectivamente, sendo o tempo para eleição de um novo mestre, destacado em verde, de aproximadamente 9 segundos.

**Momento de ocorrência da falha**

```

✓ 513 12:40:53 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,305 sec
✓ 514 12:40:54 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,281 sec
✓ 515 12:40:54 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,285 sec
✗ 516 12:40:54 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... Error Code: 2013. Lost connection to MySQL server d... 0,138 sec
✓ 517 12:41:03 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,510 sec
✓ 518 12:41:04 INSERT INTO mydataset(id,track_name,size_bytes,app_desc) V... 1 row(s) affected 0,404 sec

```

**Tempo para recuperação**

**Failover bem sucedido**

Figura 59 - Tempo para execução do processo de *failover* (Sete servidores - Dados com 110.2MB).

```

select count(id) from dataset.mydataset;
select count(emp_no) from employees.salaries;
Enter password:

```

Variable_name	Value
group_replication_primary_member	61b47e42-39fa-11ea-bc5e-f6055af10854

**Servidor primário**

CHANNEL_NAME	MEMBER_ID	MEMBER_HOST	MEMBER_PORT	MEMBER_STATE
group_replication_applier	61b47e42-39fa-11ea-bc5e-f6055af10854	my-group-2.my-group-gvr.data	3306	ONLINE
group_replication_applier	96451bf2-39fa-11ea-8d4f-825de89a58d8	my-group-4.my-group-gvr.data	3306	ONLINE
group_replication_applier	b13d3cdc-39fa-11ea-8801-ae2421e27854	my-group-5.my-group-gvr.data	3306	ONLINE
group_replication_applier	d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3	my-group-6.my-group-gvr.data	3306	ONLINE

**Grupo de replicação**

count(id)
128

**Quantidade de dados inseridas no momento da terceira falha**

count(emp_no)
2844047

**Quantidade total de dados em uma das tabelas do banco**

Figura 60 - Status do grupo no momento da terceira falha (Sete servidores - Dados com 110.2MB).

Nas Figuras 61, 62 e 63 são ilustrados a integridade dos dados em cada servidor que ocorreu as falhas, sendo possível identificar que todos os servidores conseguiram ingressar novamente ao grupo e ficar totalmente atualizado.

```

mysql> Select @@hostname;
+-----+
| @@hostname |
+-----+
| my-group-3 |
+-----+
1 row in set (0.00 sec)

```

**Servidor acessado**

```

mysql> show status like '%primary%';
+-----+
| Variable_name | Value |
+-----+
| group_replication_primary_member | 61b47e42-39fa-11ea-bc5e-f6055af10854 |
+-----+
1 row in set (0.00 sec)

```

**Servidor primário**

```

mysql> select * from performance_schema.replication_group_members;
+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+
| group_replication_applier | 25404485-39fa-11ea-9470-aaea0f05519c | my-group-0.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 46701c20-39fa-11ea-990b-86b88649644d | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 61b47e42-39fa-11ea-bc5e-f6055af10854 | my-group-2.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 7b566992-39fa-11ea-a06b-3a6ba9e39f5e | my-group-3.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 96451bf2-39fa-11ea-8d4f-825de89a58d8 | my-group-4.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | b13d3cdc-39fa-11ea-8801-ae2421e27854 | my-group-5.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3 | my-group-6.my-group-gvr.data | 3306 | ONLINE |
+-----+
7 rows in set (0.00 sec)

```

**Grupo de replicação**

```

mysql> select count(id) from dataset.mydataset;
+-----+
| count(id) |
+-----+
| 168 |
+-----+
1 row in set (0.00 sec)

```

**Quantidade total de dados inseridas no teste**

```

mysql> select count(emp_no) from employees.salaries;
+-----+
| count(emp_no) |
+-----+
| 2844047 |
+-----+
1 row in set (0.62 sec)

```

**Quantidade total de dados em uma das tabelas do banco**

Figura 61 - Primeiro servidor em falha totalmente recuperado (Sete servidores - Dados com 110.2MB).



```

mysql> Select @@hostname;
+-----+
| @@hostname |
+-----+
| my-group-0 |
+-----+
1 row in set (0.00 sec)

mysql> show status like '%primary%';
+-----+
| Variable_name | Value |
+-----+
| group_replication_primary_member | 61b47e42-39fa-11ea-bc5e-f6055af10854 |
+-----+
1 row in set (0.00 sec)

mysql> select * from performance_schema.replication_group_members;
+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+
| group_replication_applier | 25404485-39fa-11ea-9470-aaea0f05519c | my-group-0.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 46701c20-39fa-11ea-990b-86b88649644d | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 61b47e42-39fa-11ea-bc5e-f6055af10854 | my-group-2.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 7b566992-39fa-11ea-a06b-3a6ba9e39f5e | my-group-3.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 96451bf2-39fa-11ea-8d4f-825de89a58d8 | my-group-4.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | b13d3cdc-39fa-11ea-8801-ae2421e27854 | my-group-5.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3 | my-group-6.my-group-gvr.data | 3306 | ONLINE |
+-----+
7 rows in set (0.00 sec)

mysql> select count(id) from dataset.mydataset;
+-----+
| count(id) |
+-----+
| 168 |
+-----+
1 row in set (0.00 sec)

mysql> select count(emp_no) from employees.salaries;
+-----+
| count(emp_no) |
+-----+
| 2844047 |
+-----+
1 row in set (4.10 sec)

```

Figura 62 - Segundo servidor em falha totalmente recuperado (Sete servidores - Dados com 110.2MB).

```

mysql> Select @@hostname;
+-----+
| @@hostname |
+-----+
| my-group-1 |
+-----+
1 row in set (0.00 sec)

mysql> show status like '%primary%';
+-----+
| Variable_name | Value |
+-----+
| group_replication_primary_member | 61b47e42-39fa-11ea-bc5e-f6055af10854 |
+-----+
1 row in set (0.00 sec)

mysql> select * from performance_schema.replication_group_members;
+-----+
| CHANNEL_NAME | MEMBER_ID | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE |
+-----+
| group_replication_applier | 25404485-39fa-11ea-9470-aaea0f05519c | my-group-0.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 46701c20-39fa-11ea-990b-86b88649644d | my-group-1.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 61b47e42-39fa-11ea-bc5e-f6055af10854 | my-group-2.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 7b566992-39fa-11ea-a06b-3a6ba9e39f5e | my-group-3.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | 96451bf2-39fa-11ea-8d4f-825de89a58d8 | my-group-4.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | b13d3cdc-39fa-11ea-8801-ae2421e27854 | my-group-5.my-group-gvr.data | 3306 | ONLINE |
| group_replication_applier | d7dae3eb-39fa-11ea-b48b-a6a3aa5706d3 | my-group-6.my-group-gvr.data | 3306 | ONLINE |
+-----+
7 rows in set (0.00 sec)

mysql> select count(id) from dataset.mydataset;
+-----+
| count(id) |
+-----+
| 168 |
+-----+
1 row in set (0.00 sec)

mysql> select count(emp_no) from employees.salaries;
+-----+
| count(emp_no) |
+-----+
| 2844047 |
+-----+
1 row in set (3.14 sec)

```

Figura 63 - Terceiro servidor em falha totalmente recuperado (Sete servidores - Dados com 110.2MB).

## 4.2 A ESCALABILIDADE NOS TESTES

Conforme discutido na seção de alta disponibilidade, a escalabilidade é um recurso de extrema importância em arquiteturas de alta disponibilidade, e pensando nesse fator, os testes realizados no decorrer do trabalho foram todos realizados na mesma arquitetura, utilizando os recursos do *Kubernetes* para expandir ou retrain o tamanho do *cluster*.

Para a realização desse processo, foi preparado o ambiente de testes com três arquivos de configuração diferentes para o *ProxySQL*, cada um deles contendo o tamanho correto da arquitetura de replicação.

À expansão da arquitetura de replicação foi realizada alterando o arquivo de configuração do banco, para a quantidade de replicação que seria utilizada.

Para esse trabalho de conclusão, foram alterados no arquivo, o valor de três para cinco, nos testes com cinco servidores e sete para os testes com sete servidores.

Com o arquivo alterado, bastou-se utilizar o comando **kubectl apply -f 'nomedodocumento'.yaml** para que a arquitetura *Kubernetes* atualizasse a lista de servidores, conforme Figura 64.

Com o banco de replicação expandido, utilizou-se novamente o comando para atualizar a estrutura do *ProxySQL*, conforme Figuras 65 e 66, tornando a arquitetura altamente escalável.

```
mateus@cloudshell:~$ kubectl apply -f database.yaml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
mysql.kubedb.com/my-group configured
mateus@cloudshell:~$ kubectl get pod,svc -n data
```

NAME	READY	STATUS	RESTARTS	AGE
pod/my-group-0	1/1	Running	1	7m29s
pod/my-group-1	1/1	Running	0	6m6s
pod/my-group-2	1/1	Running	0	121m
pod/my-group-3	1/1	Running	0	120m
pod/my-group-4	1/1	Running	0	119m
pod/my-group-5	0/1	Init:0/1	0	8s
pod/proxysql-0	1/1	Running	0	114m
pod/proxysql-1	1/1	Running	0	114m

Comando utilizado para atualizar a estrutura do grupo de replicação

Estrutura em processo de atualização

Figura 64 - Escalando estrutura de servidores.



```

mateus@cloudshell:~$ kubectl create configmap proxysql-configmap2 --from-file=proxysql.cnf -n data
configmap/proxysql-configmap2 created
mateus@cloudshell:~$ kubectl get pod,svc -n data

```

NAME	READY	STATUS	RESTARTS	AGE
pod/my-group-0	1/1	Running	1	6m40s
pod/my-group-1	1/1	Running	0	5m17s
pod/my-group-2	1/1	Running	0	120m
pod/my-group-3	1/1	Running	0	119m
pod/my-group-4	1/1	Running	0	118m
pod/proxysql-0	1/1	Running	0	113m
pod/proxysql-1	1/1	Running	0	113m

Comando utilizado para criar o novo arquivo de configuração para o ProxySQL

Arquitetura atual

Figura 65 - Criação do segundo arquivo de configuração do *ProxySQL*.

```

mateus@cloudshell:~$ kubectl apply -f proxysql.yaml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
statefulset.apps/proxysql configured
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
service/proxysql configured
mateus@cloudshell:~$ kubectl get pod,svc -n data

```

NAME	READY	STATUS	RESTARTS	AGE
pod/my-group-0	1/1	Running	1	10m
pod/my-group-1	1/1	Running	0	9m3s
pod/my-group-2	1/1	Running	0	124m
pod/my-group-3	1/1	Running	0	123m
pod/my-group-4	1/1	Running	0	122m
pod/my-group-5	1/1	Running	0	3m5s
pod/my-group-6	1/1	Running	0	2m21s
pod/proxysql-0	1/1	Running	0	117m
pod/proxysql-1	1/1	Terminating	0	117m

Comando utilizado para atualizar as configurações do ProxySQL

Estrutura em processo de atualização

Figura 66 - Atualização da arquitetura do *ProxySQL* para atender a nova arquitetura.

## 5. RESULTADOS

Conforme visto na elaboração dos testes, com a arquitetura montada, foi possível obter um ambiente de alta disponibilidade e altamente escalável para banco de dados, sendo a arquitetura de fácil implementação e manuseio.

Outro ponto a se destacar, é que a ferramenta *Kubernetes* provou ser grande aliada para a aplicação de alta disponibilidade em banco de dados *MySQL*, e que com as ferramentas certas é possível operar banco de dados relacionais containerizados em nuvem.

Nos testes realizados quanto a tolerância a falhas, provou-se que a arquitetura conseguiria passar por um processo de falha e se recuperar de forma automática, sendo o tempo médio, para a eleição de um novo líder do grupo de replicação, de aproximadamente 13 segundos, conforme Quadro 3.

**Quadro 3: Relação de tempo de *failover* por arquitetura.**

<b>Quantidade de servidores</b>	<b>Dados armazenados</b>	<b>Quantidade de execuções de falhas</b>	<b>Tempo para conclusão do <i>failover</i> em cada falha (em segundos)</b>
3	Sem dados	1	11
3	Dados intermediários	1	21
3	Dados expressivos	1	15
5	Sem dados	2	11
5	Dados intermediários	2	10 e 15 respectivamente
5	Dados expressivos	2	16 e 12 respectivamente
7	Sem dados	3	15, 13 e 12 respectivamente
7	Dados intermediários	3	10, 12, 9 respectivamente
7	Dados expressivos	3	10, 13 e 9 respectivamente

Com base nos resultados dos testes, podemos afirmar que foi possível alcançar tanto os objetivos gerais quanto os objetivos específicos deste trabalho, no Quadro 4, é ilustrado como a arquitetura poderia ter ajudado na redução de prejuízos para o estudo apresentado em E-COMMERCE (2018).

**Quadro 4: Relação de economia de tempo e valor monetário comparado com a arquitetura proposta.**

<b>Empresa</b>	<b>Minutos <i>offline</i></b>	<b>Segundos <i>offline</i></b>	<b>Prejuízo em Reais durante 4 horas de instabilidade</b>	<b>Valor do prejuízo de 1 segundo <i>offline</i></b>	<b>Prejuízo de 13 segundos <i>offline</i>, usando a arquitetura proposta</b>
E-COMMERCE BRASIL-2018 Black Friday- 4 horas de instabilidade	240	14400	R\$ 3.100.000,00	R\$ 215,28	R\$ 2.798,64

Conforme visto no Quadro 4, e considerando um ambiente idealizado, se o estudo apresentado em E-COMMERCE (2018) tivesse utilizado a arquitetura proposta neste trabalho, hipoteticamente, teriam deixado instáveis na *Internet* apenas 13 segundos ao invés de 4 horas, e considerando ainda o fator financeiro, teriam tido prejuízo de R\$ 2.798,64 ao invés de R\$ 3.100.000,00.

## 6. CONSIDERAÇÕES FINAIS

Conclui-se por meio deste trabalho que mesmo o *Kubernetes* sendo uma ferramenta relativamente nova e o banco de dados em *containers* ainda apresenta poucos estudos, a arquitetura de banco de dados em *containers* orquestradas pelo *Kubernetes*, pode sim ser um grande diferencial em um sistema que necessite de alta disponibilidade, fornecendo diversos recursos a fim de manter uma arquitetura íntegra e altamente escalável.

Observa-se cada vez mais o armazenamento de dados em nuvem, e que concomitante a essa realidade, está ocorrendo uma evolução tecnológica dos ambientes de alta disponibilidade para serviços de armazenamento, como banco de dados em *containers*. Muitas empresas como *Google* e *Oracle* já realizam pesquisas para o uso de banco de dados em *containers* e a tendência de ferramentas como *Kubernetes* será melhorar sua estrutura para comportar estes sistemas de alta disponibilidade.

Conforme os resultados obtidos nessa pesquisa, em caso de ocorrência de falhas e/ou momentos de instabilidade na *Internet*, a arquitetura apresentada obteve um tempo médio para a recuperação de tais falha(s) de aproximadamente 13 segundos. Realizando ainda uma analogia com os dados de uma das empresas citadas neste trabalho, é possível inferir que o usuário final ficaria sem os serviços disponíveis por intervalos de tempo bem menores, e que as empresas provedoras desses serviços teriam seus prejuízos minimizados significativamente.

Contudo, o uso do *Kubernetes* para os sistemas de alta disponibilidade, devem ser cuidadosamente analisados pela empresa, pois sua má configuração pode levar a severos transtornos.

Por essa razão, ao usar tal ferramenta, deve-se sempre realizar *backups* com frequência, podendo ser de duas a três vezes no dia, e ter um servidor replicado fora da estrutura *kubernetes*, para que em caso de perda total da estrutura, a empresa ainda possa continuar operante, minimizando quaisquer danos causados.

Como trabalhos futuros, propõe-se:

- Estudo da arquitetura para grupos de replicação multiprimário, avaliando como se comporta quanto à consistência dos dados em um grupo distribuído;
- Validação quanto a segurança da arquitetura, verificando e minimizando os possíveis pontos de vulnerabilidades;
- Realizar um estudo de *benchmark*, averiguando o quanto a estrutura pode comportar carga sem a ocorrência de falhas.

## REFERÊNCIAS

ALBUQUERQUE FILHO, A. C. (2016). **Estudo comparativo entre Docker Swarm e Kubernetes para orquestração de contêineres em arquiteturas de software com microsserviços**. Recife-PE, Universidade Federal de Pernambuco.

ALLES, G. R. (2018). **Análise da utilização de tecnologias de contêineres para aplicações de alto desempenho**. Porto Alegre-RS, Universidade do Rio Grande do Sul.

ANGONESE, C. **BALANCEAMENTO DE CARGA DE TRABALHO EM COMPUTAÇÃO EM NUVEM BASEADO EM REDES MAGNÉTICAS VIRTUAIS**. Dissertação (Pós-Graduação em Informática), Curitiba-PR, Pontifícia Universidade Católica do Paraná, 2012.

BORGES, H. P et al (2011). **COMPUTAÇÃO EM NUVEM**, Portal do livro aberto. Disponível em:  
<https://livroaberto.ibict.br/bitstream/1/861/1/COMPUTA%c3%87%c3%83O%20EM%20NUVEM.pdf>. Último acesso em: 06 de jan. de 2020.

BRUSCHI, G. C. et al (2014). **SISTEMA DE ALTA DISPONIBILIDADE EM BANCO DE DADOS MYSQL UTILIZANDO LINUX LVS**. Bauru-SP, Faculdade de Tecnologia de Bauru.

CANALI, G, P. et al (2015). **Alta disponibilidade em Banco de Dados Oracle usando Real Application Cluster (RAC)**. Curso de Tecnologia em Banco de Dados. Bauru-SP, Faculdade de Tecnologia de Bauru.

CANALTECH.COM. (2013). **Indisponibilidade no e-commerce pode gerar prejuízo de até US\$ 8 mil por minuto**. Publicado em: 13 de dez. de 2013. Disponível em:  
<https://canaltech.com.br/e-commerce/Indisponibilidade-no-e-commerce-pode-gerar-prejuizo-de-ate-US-8-mil-por-minuto/>. Acessado em: 15 de jan. de 2020.

CIRCUITOMT.COM. (2015). **Sites fora do ar na Black Friday causam prejuízo de R\$ 1,5 milhão**. Publicado em: 30 de out. de 2015. Disponível em:

<http://circuitomt.com.br/editorias/mundo-tecnologico/76489-sites-fora-do-ar-na-black-friday-causam-prejuizo-de-r-15-milhao.html>. Acessado em: 15 de jan. de 2020.

CORRÊA, J. N. G. F. (2016). **LIFTER - DISPONIBILIZAÇÃO DE APLICAÇÕES VIA CONTAINERS DE SOFTWARE EM UM CLUSTER DE ALTO DESEMPENHO**. Florianópolis-SC, Universidade Federal de Santa Catarina.

COSTA, F. M. et al. (2018). **Solução de Alta Disponibilidade para Banco de Dados MySQL**. Coordenadoria de Tecnologia da Informação. Foz do Iguaçu-PR, Universidade Federal da Integração Latino-Americana.

COSTA, H, L. A. **ALTA DISPONIBILIDADE E BALANCEAMENTO DE CARGA PARA MELHORIA DE SISTEMAS COMPUTACIONAIS CRÍTICOS USANDO SOFTWARE LIVRE: ESTUDO DE CASO**. Dissertação (Pós-Graduação em Ciência da Computação), Faculdade de Ciência da Computação, Viçosa-MG, Universidade federal de viçosa, 2009.

DOCS.MICROSOFT, (2020). **Switchovers and Failovers**. Disponível em: <https://docs.microsoft.com/en-us/exchange/switchovers-and-failovers-exchange-2013-help?redirectedfrom=MSDN>. Última revisão em: 01 de jan. de 2020. Último acesso em: 05 de jan. de 2020.

DOCKER.COM - Docs. (2019a). **Docker architecture**. Última revisão do documento: 28 de dez. de 2019. Disponível em: <https://docs.docker.com/engine/docker-overview/#docker-architecture>. Último acesso em: 29 de dez. de 2019.

DOCKER.COM - Docs. (2019b). **Docker overview**. Última revisão do documento: 28 de dez. de 2019. Disponível em: <https://docs.docker.com/engine/docker-overview/>. Último acesso em: 29 de dez. de 2019.

DOCKER.COM - Docs. (2019c). **Features**. Última revisão do documento: 28 de dez. de 2019. Disponível em: <https://docs.docker.com/compose/#features/>. Último acesso em: 29 de dez. de 2019.

E-COMMERCEBRASIL. (2018). **Lojas virtuais perderam ao menos R\$ 3,1 milhões em quatro horas de Black Friday**. Última revisão em: 28 de nov. de 2018. Disponível em: <https://www.ecommercebrasil.com.br/noticias/e-commerce-black-friday-2018/>. Acessado em: 01 de jan. de 2020.

EMER, B. (2016). **IMPLEMENTAÇÃO DE ALTA DISPONIBILIDADE EM UMA EMPRESA PRESTADORA DE SERVIÇOS PARA INTERNET**. Centro de ciências exatas e da tecnologia. Ciência da computação. Caxias do Sul-RS, Universidade Federal de Caxias do Sul.

ELMASRI, R; NAVATHE, S. B. **BANCO DE DADOS E OS USUÁRIOS DE BANCO DE DADOS**. IN: \_\_\_\_\_, \_\_\_\_\_ **Sistemas de banco de Dados**. 6 ed. Pearson, 2011, 804 p.

ESHEL, Marc M.; GOMEZ, Juan C.; NAIK, Manoj P. **System and method for preserving state for a cluster of data servers in the presence of load-balancing, failover, and fail-back events**. U.S. Patent No. 7,962,915. 14 jun. 2011.

ETCD. (2020). **A distributed, reliable key-value store for the most critical data of a distributed system**. Disponível em: <https://etcd.io/>. Último acesso em: 12 de jan. de 2020.

FEDERICI, M.; GAIBISSO C.; MARTINO, L. B. (2014). **HAVmS: Highly Available Virtual Machine Computer System Fault Tolerant with Automatic Failback and Close to Zero Downtime**. Acta Polytechnica CTU Proceedings. 1. 278-282. 10.14311/APP.2014.01.0278.

FREITAS, E. L. S. X. **KNOWING: UM MODELO PARA GARANTIA DE CONSISTÊNCIA DOS DADOS EM SISTEMAS DE BANCO DE DADOS RELACIONAIS EM NUVEM**. Dissertação (Mestrado em ciência da computação) Centro de informática. Recife-PE, Universidade de Pernambuco, 2014.

G1.COM. (2008). **Fora do ar, Amazon sofre prejuízo estimado em US\$ 31 mil por minuto**. Última revisão em: 09 de jun. de 2008. Disponível em: <http://g1.globo.com/Noticias/Tecnologia/0,,MUL592388-6174,00-FORA+DO+AR+AMAZON+SOFRE+PREJUIZO+ESTIMADO+EM+US+MIL+POR+MINUTO.html>. Acessado em: 10 de jan. de 2020.

GITHUB.COM, Gitub - Sysown. (2019a). **ProxySQL**. Última revisão do documento: 28 de dez. de 2019. Disponível em: <https://github.com/sysown/proxysql>. Último acesso em: 11 de jan. de 2020.

GREGOL, R. E. W. (2011). **RECURSOS DE ESCALABILIDADE E ALTA DISPONIBILIDADE PARA APLICAÇÕES WEB**. Curso Superior de Tecnologia em



Análise e Desenvolvimento de Sistemas. Medianeira-PR, UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ.

HANWHA-SECURITY.COM, Hanwha Techwin, (2017). **Failover Technology**.

Disponível em:

<https://www.hanwha-security.com/data/gen/attrbt/1511165564392.pdf>. último acesso em: 12 de jan. de 2020.

HASHIMOTO, G. T. **UMA PROPOSTA DE EXTENSÃO PARA UM PROTOCOLO PARA ARQUITETURA DE ALTA DISPONIBILIDADE**. Dissertação (Pós-graduação em ciência da computação). Uberlândia-MG, Universidade Federal de Uberlândia, 2009.

HIRST, Michael D.; GALE, Alan A.; CUMMINGS, Gene A. **Method and system for fault-tolerant network connection switchover**. U.S. Patent No. 6,173,411. 9 Jan. 2001.

KNOB, R. R. (2018). **Análise e benchmarking das soluções NewSQL CockroachDB, MemSQL, NuoDB e VoltDB**. Graduação em Sistemas de Informação. Florianópolis-SC, Universidade Federal de Santa Catarina.

KUBEDB. (2020). **Run production-grade databases easily on Kubernetes**. Disponível em: <https://kubedb.com/>. Último acesso em: 12 de janeiro de 2020.

KUBERNETES.IO - Docs. (2019a). **Pod Overview**. Última revisão do documento: 26 de dez. de 2019. Disponível em: <https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/>. Último acesso em: 29 de jan. de 2020.

KUBERNETES.IO - Docs. (2019b). **StatefulSets**. Última revisão do documento: 01 de dez. de 2019. Disponível em: <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>. Último acesso em: 01 de jan. de 2020.

KUBERNETES.IO - Docs. (2019c). **Deployments**. Última revisão do documento: 22 de nov. de 2019. Disponível em: <https://kubernetes.io/docs/concepts/workloads/controllers/deployment/>. Último acesso em: 29 de dez. de 2019.

KUBERNETES.IO - Docs. (2019d). **Understanding Kubernetes Objects**. Última revisão do documento: 08 de out. de 2019. Disponível em: <https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/>. Último acesso em: 29 de nov. de 2019.

KUBERNETES.IO - Docs. (2019e) **Extend the Kubernetes API with CustomResourceDefinitions**. Última revisão do documento: 20 de dez. de 2019. Disponível em: <https://kubernetes.io/docs/tasks/access-kubernetes-api/custom-resources/custom-resource-definitions/>. Último acesso em: 15 de jan. de 2020.

KUBERNETES.IO - Docs (2019f). **Extending the Kubernetes API with the aggregation layer**. Última revisão do documento: 16 de set. de 2019. Disponível em: <https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/apiserver-aggregation/>. Último acesso em: 11 de dez. de 2019.

KUBERNETES.IO. (2020a). **Production-Grade Container Orchestration**. Disponível em: <https://kubernetes.io/>. Último acesso em: 29 de jan. de 2020.

KUBERNETES.IO - Docs. (2020b). **Kubernetes Components**. Última revisão do documento: 16 de jan. de 2020. Disponível em: <https://kubernetes.io/docs/concepts/overview/components/>. Último acesso em: 29 de jan. de 2020.

KUBERNETES.IO - Docs. (2020c). **Node**. Última revisão do documento: 06 de jan. de 2020. Disponível em: <https://kubernetes.io/docs/concepts/architecture/nodes/>. Último acesso em: 29 de jan. de 2020.

KUBERNETES.IO - Docs. (2020d). **ReplicaSet**. Última revisão do documento: 10 de jan. de 2020. Disponível em: <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>. Último acesso em: 29 de jan. de 2020.

KUBERNETES.IO - Docs. (2020e). **Persistent Volumes**. Última revisão do documento: 16 de jan. de 2020. Disponível em: <https://kubernetes.io/docs/concepts/storage/persistent-volumes/>. Último acesso em: 29 de jan. de 2020.

KUBERNETES.IO - Docs. (2020f). **Configure a Pod to Use a ConfigMap**. Última revisão do documento: 08 de jan. de 2020. Disponível em:

<https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-configmap/>.  
Último acesso em: 27 de jan. de 2020.

KUBERNETES.IO - Docs. (2020g). **Service**. Última revisão do documento: 15 de jan. de 2020. Disponível em:  
<https://kubernetes.io/docs/concepts/services-networking/service/>. Último acesso em:  
29 de jan. de 2020.

LARSSON, O. **RUNNING DATABASES IN A KUBERNETES CLUSTER**. Master Thesis (Civilingenjörprogrammet i teknisk datavetenskap). Umea University, 2019.

LETTERMAN, D. **20 Key High Availability Design Principles** in: MARCUS, E; STERN H. **BLUEPRINTS FOR HIGH AVAILABILITY**. 2 ed. Indianapolis, Wiley Publishing, 2003. p. 75-104.

MCKEAN, Brian D.; OTTERNESS, Noel S.; SKAZINSKI, Joseph G. **System and method for efficient failover/failback techniques for fault-tolerant data storage system**. U.S. Patent No. 6,681,339. 20 jan. 2004.

MESSINA D. (2018). **5 years later, Docker has come a long way**. Disponível em:  
<https://www.docker.com/blog/5-years-later-docker-come-long-way/>. Último acesso em:  
em: 21 jan. 2020.

MEDEIROS, W. R. (2018). **Tolerância a Falhas em Sistemas Embarcados Baseados em Microcontroladores**. Natal-RN, Universidade Federal do Rio Grande do Sul.

MYSQL.COM, Mysql - Documentation. (2019a). **Group Replication**. Disponível em:  
<https://dev.mysql.com/doc/refman/5.7/en/group-replication-summary.html>. Último acesso em: 29 de dez. de 2019.

MYSQL.COM, Mysql - Documentation. (2019b). **Single-Primary Mode**. Disponível em:  
<https://dev.mysql.com/doc/refman/8.0/en/group-replication-single-primary-mode.html>.  
Último acesso em: 28 de dez. de 2019.

MYSQL.COM, Mysql - Documentation. (2019c). **Multi-Primary Mode**. Disponível em:  
<https://dev.mysql.com/doc/refman/8.0/en/group-replication-multi-primary-mode.html>.  
Último acesso em: 28 de dez. de 2019.

MYSQL.COM, Mysql - Documentation. (2019d). **Fault-tolerance**. Disponível em: <https://dev.mysql.com/doc/refman/5.7/en/group-replication-fault-tolerance.html>. Último acesso em: 07 de jan. de 2019.

NEWS.COMSCHOOL. (2015). **Impacto de um Aplicativo Fora do Ar**. Disponível em: <https://news.comschool.com.br/impacto-de-um-servico-fora-do-ar/>. Último acesso em: 21 de fev. de 2020.

NOVAES, R. (2014). **Falha no Facebook causa prejuízo de US\$ 500 milhões**. Publicado em: 02 de jul. de 2014. Disponível em: <https://www.psafe.com/blog/falha-causa-prejuizo-facebook/>. Acessado em: 05 de jan. de 2020.

ORACLE.COM. (2020) Technetwork, VENKATESH, P., **MySQL Replication and Scalability**, Disponível em: <https://www.oracle.com/technetwork/community/developer-day/mysql-replication-scalability-403030.pdf>. Último acesso em: 15 de jan. de 2020.

PAULA SILVA, A. L. (2016). **Avaliação Comparativa de Escalabilidade de Aplicações de Alto Desempenho em Nuvem Pública e Privada**. Jataí-GO, Universidade Federal de Goiás.

PITANGA, M. **Construindo Super Computadores com Linux**. 3. ed. Rio de Janeiro-RJ: Brasport, 400 p., 2008.

POSSOBOM, C. C. (2010). **ESTUDO DE CASO: CLOUD COMPUTING - COMPUTAÇÃO EM NUVEM**. Ijuí-RS, UNIVERSIDADE REGIONAL DO NOROESTE DO ESTADO DO RIO GRANDE DO SUL.

RUBENS, P. (2017). **What are containers and why do you need them?**. Cio.com Disponível em: <https://www.cio.com/article/2924995/what-are-containers-and-why-do-you-need-them.html>. Acessado em: 27 de dez. 2019.

SÁ, D.; NEVES. V. H; (2012). **CLUSTER DE ALTA DISPONIBILIDADE EM LINUX**. Tecnólogo em Redes de Computadores. Santa Bárbara d'Oeste-SP, Faculdade Politec.

SANTOS, E. V. (2015), **MECANISMO DE TOLERÂNCIA A FALHAS ATRAVÉS DE ESCALONAMENTO PARA UMA ARQUITETURA RECONFIGURÁVEL DE GRÃO GROSSO**, Natal-RN, Universidade Federal do Rio Grande do Norte.

SANTOS, J. B. (2016). **CONTAINERS DOCKER COMO ESTRATÉGIA DE VIRTUALIZAÇÃO DE CLOUD**. Curso de especialização em tecnologia da informação bancária. São Paulo-SP, Escola politécnica da Universidade de São Paulo.

SEVERALNINES. (2019). **Database Load Balancing for MySQL and MariaDB with ProxySQL - Tutorial**. Severalnines. Disponível em: <https://severalnines.com/resources/tutorials/proxysql-tutorial-mysql-mariadb>. Acessado em: 01 de jan. 2019.

SILVA, F. H. R. (2017). **Avaliação de Desempenho de Contêineres Docker para Aplicações do Supremo Tribunal Federal**. Brasília-DF, Universidade de Brasília.

SOUSA, F. R.; MOREIRA, L. O.; MACHADO, J.C. (2009) **Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios**. In: OLIVEIRA A. C.; MOURA, R. S.; SOUZA, F. V. (Org.). III Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI). 1 ed. Teresina: SBC, 2009, v. 1, p. 150-175 apud BORGES, H. P et al (2011). **COMPUTAÇÃO EM NUVEM, Portal do livro aberto**. Disponível em: <https://livroaberto.ibict.br/bitstream/1/861/1/COMPUTA%c3%87%c3%83O%20EM%20NUVEM.pdf>. Último acesso em: 06 de jan. de 2020.

SOUZA, C; CAMPOS, P. (2008) **Alta disponibilidade em Banco de Dados utilizando tecnologia Oracle 147f**. Brasília-DF, Associação educativa do Brasil faculdades integradas ICESP apud CANALI, G, P. et al (2015). **Alta disponibilidade em Banco de Dados Oracle usando Real Application Cluster (RAC)**, Bauru-SP, Faculdade de Tecnologia de Bauru.

SYSBASE.COM, Adaptive Server® Enterprise, **Using Sybase Failover in a High Availability System**. Disponível em: [http://infocenter.sybase.com/help/topic/com.sybase.dc31651\\_1251/pdf/ha\\_avail.pdf](http://infocenter.sybase.com/help/topic/com.sybase.dc31651_1251/pdf/ha_avail.pdf). Último acesso em: 03 de jan. de 2020.

VAZ, D. D; MONKS, E. M. (2014). **Ferramentas Open Source e Redundância de Backups**. Tecnologia em Redes de Computadores. Pelotas-RS, Faculdade de Tecnologia Senac.

YAML. (2020). **YAML is a human friendly data serialization standard for all programming languages**. Disponível em: <https://yaml.org/>. Último acesso em: 12 de jan. de 2020.

## APÊNDICES A - PROXYSQL

O *ProxySQL* segundo sua documentação (GITHUB.COM, 2019a), “[...] é um *proxy* de alto desempenho, alta disponibilidade e com reconhecimento de protocolo para *MySQL* [...]”, onde atua na camada de aplicação, redirecionando a carga de trabalho com base nas *queries* solicitadas pela aplicação, tendo como principal vantagem ser totalmente *open source* e testado em ambiente de produção.

Por atuar na camada de aplicação, se torna extremamente flexível, podendo criar roteamento de *queries* específicas para cada servidor.

De acordo com sua documentação, seus benefícios vão além de balanceamento de carga, possuindo *cache* de consulta, suporte a *failover*, suporte a diversos tipos de topologias de replicação, *firewall* e principalmente configuração dinâmica, não sendo necessário parar o serviço para aplicar novas atualizações de configuração.

Toda a configuração é feita através da sintaxe SQL e é armazenada em um banco de dados *SQLite*, para sua configuração ser dinâmica são utilizados três camadas de configuração, sendo elas:

- **RUNTIME:** É a estrutura de dados em memória do *ProxySQL* usadas pelos pedidos de tratamento de *threads* (tarefas a serem executadas em segundo plano);
- **MEMORY:** Representa o banco de dados na memória que é exposto através da interface compatível com o *MySQL*;
- **DISK:** É o banco de dados *SQLite* contendo as configurações definidas em disco.

A Figura 67 ilustra de forma clara o funcionamento do *ProxySQL*:

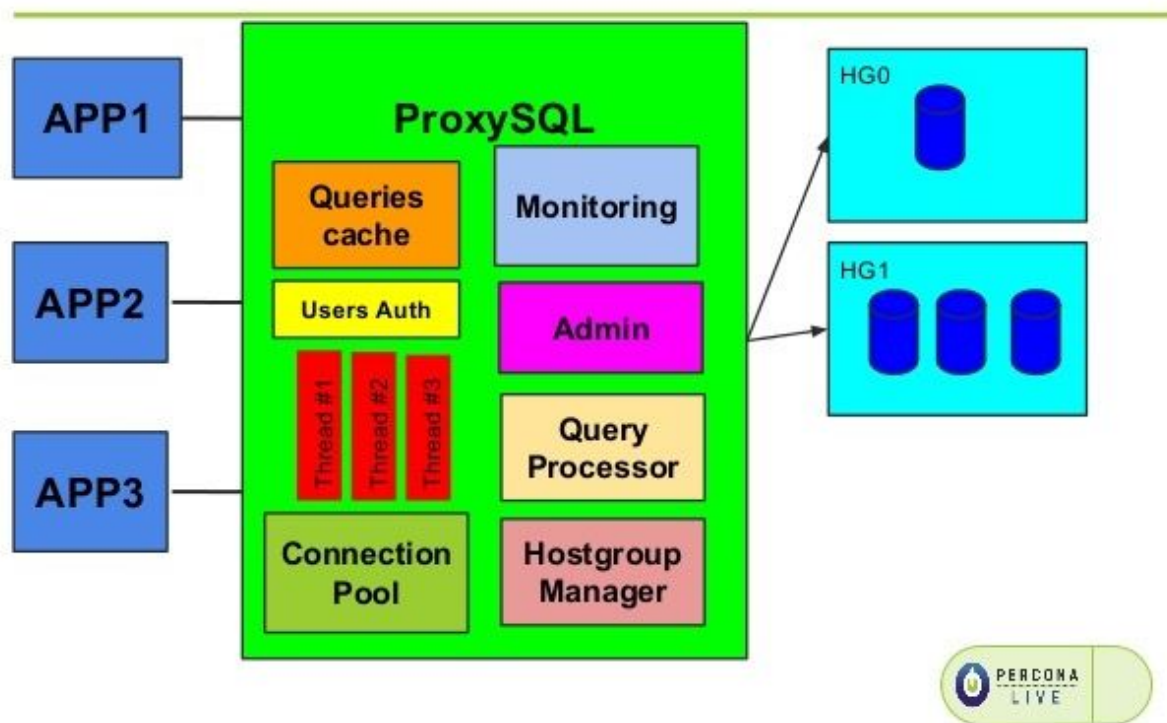


Figura 67 - Estrutura *ProxySQL*.  
Fonte: Severalnines.com (2019).

Por possuir uma estrutura descentralizada é possível manter mais de uma instância em vários locais distintos, a fim de manter a aplicação sem nenhum ponto único de falha.



## APÊNDICES B - *MYSQL NO KUBEDB*

*KubeDB* é uma ferramenta de código aberto, especializada em executar banco de dados em nível de produção de forma fácil e simples.

A execução do banco de dados a nível de produção em uma estrutura *Kubernetes* é complicada, e o *KubeDB* é um operador que facilita a implementação, realizando todo o processo de configuração, provisionamento, aplicação de atualizações, *backup*, recuperação, detecção de falhas e reparo de forma rápida e prática (KUBEDB, 2020).

Para utilizá-lo basta descrever a configuração de banco de dados desejada em um arquivo *YAML*, e o operador *KubeDB* criará objetos *Kubernetes* no estado desejado.

Atualmente a ferramenta conta com suporte a diversos banco de dados, dentre eles o *MySQL*, foco deste trabalho, dentre os recursos suportados do *MySQL* no *KubeDB* destacam-se:

- Os *backups* instantâneos;
- *Backups* agendados;
- Inicialização de banco de dados utilizando *backups*;
- Inicialização utilizando *scripts*.

O operador *MySQL* no *kubeDB* é um *Kubernetes* CRD, ele fornece configuração declarativa para o *MySQL* de maneira nativa do *Kubernetes*, descrevendo através de um objeto *MySQL* a configuração do banco de dados desejada.

Para utilizá-lo é necessário instalar operador *Kubernetes KubeDB* no *cluster*, após sua instalação, é possível instalar os objetos *Kubernetes*.

Com a instalação realizada, é necessário informar um arquivo de configuração, contendo a configuração desejada, obrigatoriamente dentro do arquivo é necessário possuir uma *apiVersion*, *kind*, *metadata*, e um *spec* assim como os demais objetos de configuração dentro do *Kubernetes*.

O Quadro 5, ilustra como é formado um arquivo de configuração para uma replicação de grupo na versão 5.7.25 do *MySQL*.

```
apiVersion: kubedb.com/v1alpha1
kind: MySQL
metadata:
  name: my-group
  namespace: demo
spec:
  version: "5.7.25"
  replicas: 3
  topology:
    mode: GroupReplication
    group:
      name: "dc002fc3-c412-4d18-b1d4-66c1fbfbbc9b"
      baseServerID: 100
  storageType: Durable
  storage:
    storageClassName: "standard"
    accessModes:
      - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  terminationPolicy: WipeOut
```

**Quadro 5 - Arquivo de configuração para replicação de grupo *MySQL*.**  
**Fonte: *KubeDB* (2020).**