UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ESPECIALIZAÇÃO EM ARQUITETURA E GESTÃO DE INFRAESTRUTURA DE TI

FÁBIO LUIZ BARBOSA

KUBERNETES EM AMBIENTES NÃO PERSISTENTES

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2021

FÁBIO LUIZ BARBOSA

KUBERNETES EM AMBIENTES NÃO PERSISTENTES

Monografia de Especialização, apresentada ao Curso de Especialização em Arquitetura e Gestão de Infraestrutura de TI, do Departamento Acadêmico de Eletrônica – DAELN, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Dr. Kleber Kendy Horikawa Nabas

CURITIBA
2021

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Curitiba

Diretoria de Pesquisa e Pós-Graduação
Departamento Acadêmico de Eletrônica
Curso de Especialização em Arquitetura e Gestão de Infraestrutura de TI

TERMO DE APROVAÇÃO

KUBERNETES EM AMBIENTES NÃO PERSISTENTES

por

FÁBIO LUIZ BARBOSA

Esta monografia foi apresentada em 21 de Dezembro de 2021 como requisito parcial para a obtenção do título de Especialista em em Arquitetura e Gestão de Infraestrutura de TI. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Kleber Kendy Horikawa Nabas
Orientador

_____
Prof. Dr. Joilson Alves Junior
Membro titular

_____
Prof. M. Sc. Omero Francisco Bertol
Membro titular

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

AGRADECIMENTOS

Agradeço aqueles que souberam, participaram ou incentivaram deste momento especial na adição dos conhecimentos adquiridos neste curso de especialização em tempos tão perturbadores como foram os anos de 2020 até a escrita deste documento.

Agradeço ao meu orientador Prof. Dr. Kleber Kendy Horikawa Nabas, pelas aulas descontraídas, a facilidade em repassar seu conhecimento e a orientação até aqui.

Ao Prof. Dr. Joilson Alves Junior se preocupou com tantos detalhes e possivelmente alguns que nunca saberemos.

Aos meus colegas de sala que permitiram um ambiente saudável com grande diversidade de opiniões sempre respeitando os outros mesmo quando discordando.

A Secretaria do Curso, pela cooperação e rápidas ações quando foram necessárias.

A Universidade principalmente pela oportunidade de abrir o curso, mas também por sediar e permitir a participação na DebConf19.

A minha esposa que teve compreensão e apoiou em momentos chave.

Por fim, deixar igualmente anotado, o meu reconhecimento à minha família, pois sem eles e seus ensinamentos de vida raramente somos algo principalmente em momentos de grandes desafios.

Teoria e prática algumas vezes se batem.
E quando isso acontece, a teoria perde.
Todas às vezes.

(TORVALDS, Linus)

RESUMO

BARBOSA, Fábio L. **Kubernetes em ambientes não persistentes**. 2021. 30 p. Monografia de Especialização em Arquitetura e Gestão de Infraestrutura de TI, Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

O presente documento aborda de forma introdutória o uso da ferramenta Kubernetes com o propósito para o qual este foi efetivamente criada sendo a efemeridade, a simplificada gestão, facilidade para de desenvolvimento e das migrações seja ela para outros clusters Kubernetes como serviços, *cloud*, *on premise* da mesma tecnologia. A metodologia adotada para análise do conteúdo usou trabalhos acadêmicos e teses publicados no Google Acadêmico e sites unificando e comparando os relatos para o tema proposto. O arquivo contempla como a necessidade no uso de recursos ou projetos de uma parte da estrutura persistente, ou seja, que não serão apagadas quando o pod for reiniciado ou escalado embora seja possível seu uso, criam uma camada a mais de preocupação além de reduzir consideravelmente a agilidade que a ferramenta veio trazer mostrando alguns como alguns processos seriam executados. Também é apresentado uma breve parte da história do Docker e Kubernetes para compreensão dos conceitos e que vem melhorando a qualidade e rapidez de como os projetos são desenvolvidos, testados e implementados.

Palavras-chave: Sistema de computação virtual. Linux. Kubernetes. Contêiner.

ABSTRACT

BARBOSA, Fábio L. **Kubernetes on non-persistent environments**. 2021. 30 p. Monography of Specialization in Archiqueture and Management of Infrastructure of IT, Academic Department of Electronics, Federal Technological University of Paraná. Curitiba, 2021.

This document addresses in an introductory way the use of the Kubernetes tool for the purpose for which it was effectively created, being the ephemerality, the simplified management, ease of development and migration, whether it is to other Kubernetes clusters as services, cloud, on premise of the same technology. The methodology adopted for content analysis used academic works and theses published on Google Scholar and websites unifying and comparing the reports for the proposed theme. The file contemplates how the need to use resources or projects of a part of the persistent structure, that is, that will not be erased when the pod is restarted or scaled although it is possible to use them, creates an extra layer of concern in addition to considerably reducing the agility that the tool brought, showing somehow some processes would be executed. A brief part of the history of Docker and Kubernetes is also presented to understand the concepts and that has been improving the quality and speed of how projects are developed, tested and implemented.

Keywords: Virtual computing system. Linux. Kubernetes. Container.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

HA	*High Availability*
HPA	*Horizontal Pod Auto scaling*
K8s	*Kubernetes*
PV	*Persistent Volume*
PVC	*Persistent Volume Claim*

LISTA DE SIGLAS

CDN	*Content Delivery Network*
GCP	*Google Console Platform*
SaaS	*Service as a Service*

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Um pouco antes de entrarmos no universo do Kubernetes é necessário assim como em qualquer assunto entender um pouco da história e conceitos até aqui e isto inclui saber um pouco também sobre Docker, não sendo todos que começam a usar sistemas de contêinerização que tem conhecimento que seu conceito principal, o isolamento de sistemas e recursos, que, para Osztertág (2020) e TLF (2017), foi iniciado com a chegada do comando *chroot* em 1979 no sistema operacional conhecido como *Unix Version 7* que não fazia mais do que mascarar o sistema de arquivos raiz do processo pai e seus processos filhos. Apenas muitos anos depois, em 2000, o FreeBSD implementou uma ferramenta chamada *jail* que permitirá que tenha sua própria rede e IP.

Para Osztertág (2020), em 2008 aparecia no Kernel do Linux o *cgroups*, um novo recurso que permitia o isolamento dos recursos do um dispositivo (Processador, memória, disco, rede, entrada, saída, etc.). Os *namespaces* vieram a aparecer apenas em 2013 permitindo que *namespaces* não pudessem acessar outros *namespaces*. No mesmo ano surgia o Docker que resolvia problemas como a falta de uso de imagens onde, até então, só era possível fazendo várias atividades manuais e com o crescimento das atividades para os times de tecnologia como diferencial competitivo das organizações não era fácil encontrar profissionais que soubessem desempenhar essa atividade.

De acordo com RisingStack (2021), o Google traz em 2014 o K8s, como uma versão *open source* do projeto Borg e começa a se popularizar em 2016 quando sai o Minikube que facilita o uso do orquestrador rodando local.

Com o K8s os deploys e as implementações ficaram muito mais rápidos ainda mais sabendo que existem recursos como CDN que mascaram uma atualização de código que possa vir a criar uma indisponibilidade no sistema durante a implementação.

Não obstante, como tantas outras tecnologias novas o Kubernetes teve seu uso deturpado sendo considerada a solução para todos os problemas possíveis em TI, o que não seria verdade dado que o foco sempre foi que a resiliência e escalabilidade (fosse ela automática ou não) seguindo o método efémero de

funcionamento, ou melhor, a estrutura não deveria depender de recursos estáticos como imagens ou arquivos de um sistema de bases de dados.

1.2 PROBLEMA

Está sendo considerado que uma organização esteja atualizando sua arquitetura e que a mesma não tenha processos ou procedimentos bem definidos de TI, mas que é uma exigência executiva que todos os recursos usados pela empresa sejam migrados para a tecnologia Kubernetes.

1.3 OBJETIVOS

O trabalho terá foco apenas em sistemas de base de dados independente de qual é usado, mas desde que suporte cluster de banco de dados.

Como membros de TI sabe-se que as bases de dados são as estruturas mais complexas, críticas e sensíveis de uma organização e a falha neste sistema pode trazer grandes prejuízos financeiros como o fechamento da organização.

1.3.1 Objetivo Geral

A contribuição neste cenário é auxiliar na tomada de decisão no que se refere a considerar que alguns sistemas precisam continuar em servidores monolíticos, máquinas virtuais ou poderiam ser usados como SaaS em clouds.

1.3.2 Objetivos Específicos

- Explanar estruturas monolíticas e de micro serviços;
- Explanar ferramenta Docker e Kubernetes;
- Comprovar que mesmo sendo possível ativar um sistema de banco de dados em K8s sua manutenção e cuidados podem ser contraproducentes;
- Apresentar soluções que possam suprir a necessidade de escalonamento da base de dados.

1.4 JUSTIFICATIVA

Tendo em vista o quão crítico e complexo pode ser para qualquer organização um sistema de banco de dados existe uma demanda para uma pessoa ou equipe imensamente especializada dando continuidade não apenas ao negócio, mas também as atividades que uma base de dados demanda como otimização de arquitetura, *queries*, índices, manutenção corretivas, preventivas, preditiva, de versão, migração, confiabilidade dos dados, etc.

Sabendo da criticidade, sistemas que dependem de estaticidade não deveriam ser considerados de forma tão fácil ou passional uma ferramenta que preza pelo foco em sistemas efêmeros e descentralização geral da estrutura sendo que um estudo mais aprofundado deve ser feito em cada caso visando sempre à garantia da continuidade do negócio e integridade dos dados.

A defesa no uso de orquestradores ou contêinerização para necessidades não efêmeras, atividade principal deste documento, deve ser feita preferencialmente antes considerando que se for mal implementado ou mal documentado a execução pode trazer consequências extremamente demoradas ou irreversíveis.

1.5 ESTRUTURA DO TRABALHO

Tal documento de especialização foi organizado em quatro capítulos. A primeira e já exibida foi o tema do trabalho tal qual a motivação e os objetivos gerais e específicos da pesquisa, a justificativa e a estrutura geral do trabalho.

Como segundo capítulo tem-se o referencial teórico que visa às essências e estratégias para melhor compreensão final.

Para o terceiro capítulo serão expostos estudos ligados à possibilidade de ter sistemas de base de dados em Kubernetes e o comportamento da ferramenta em alguns cenários.

O quarto capítulo apresenta algumas alternativas à contêinerização da base de dados.

Finalmente no quinto capítulo serão dissertadas as considerações finais com alguns apontamentos e recomendação de trabalhos futuros que poderiam ser realizados a partir deste estudo.

2 REFERENCIAL TEÓRICO

Nesta seção são apresentados os conceitos, técnicos e métodos para apoiar com base técnica para efetuar tal monografia.

2.1 MONÓLITOS

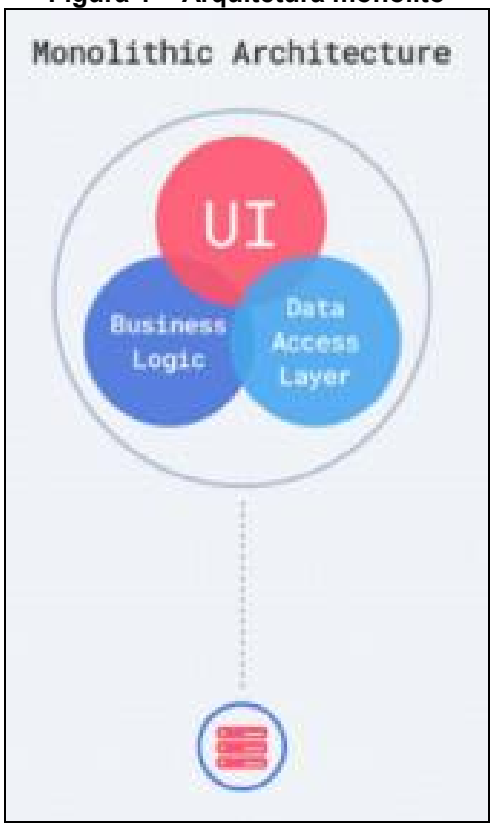
Na análise de MD (2019) para sistemas Web sugere que monólitos por padrão assumem algumas das seguintes características:

- Uma Linguagem;
- Um Servidor de Aplicação;
- Desenvolve-se aplicação orientada aquela plataforma;
- “Se ganha” recursos para armazenar sessão, pool de conexões com o banco de dados, filas, caches, entre muitos outros;
- Modelo *Stateful*.

Ainda em sistemas Web como apontado por Sequeira *et al.* (2020) usando o modelo da Figura 1 apresenta como desvantagens que atualizações mal feitas em módulos que pode derrubar o sistema e que monólitos tentem a ficar presos aos limites de tecnologia escolhidas para usar no código. E para MD (2019) existem outros problemas sendo:

- Aplicações muito grandes;
- Implantações problemáticas pra novas versões;
- Gestão de conflitos de código e dificuldade de desenvolvimentos em paralelo.
- Elasticidade limitada por ser extremamente caro e trabalhoso.

Figura 1 – Arquitetura monólito



Fonte: Sequeira *et al.* (2020).

Segundo Sequeira *et al.* (2020, p. 8), “Conforme cresce e fica mais complexa, a aplicação deixa de ser totalmente compreensível, o que faz com que a descoberta e posterior correção de erros ou a realização de alterações necessárias seja mais difícil de fazer rapidamente e de maneira correta”.

Não se pode dizer diferente uma vez que é o que acontece com aplicações, principalmente, Web pela falta de comunicação e tempo entre os departamentos do dia-a-dia gera este ruído (CORRÊA; SOARES, 2020). No entanto seria incorreto considerar pelo simples fato de serem grandes que tenham apenas desvantagens, para bases de dados que por padrão são aplicações monolíticas isto seria uma vantagem. Como apontado por Newman (2020), estas arquiteturas deveriam tender a um fluxo mais simples como ocorre com bancos de dados que são apenas uma aplicação e se desconsiderando a complexidade da modelagem do banco de dados.

2.2 MICROSERVIÇOS

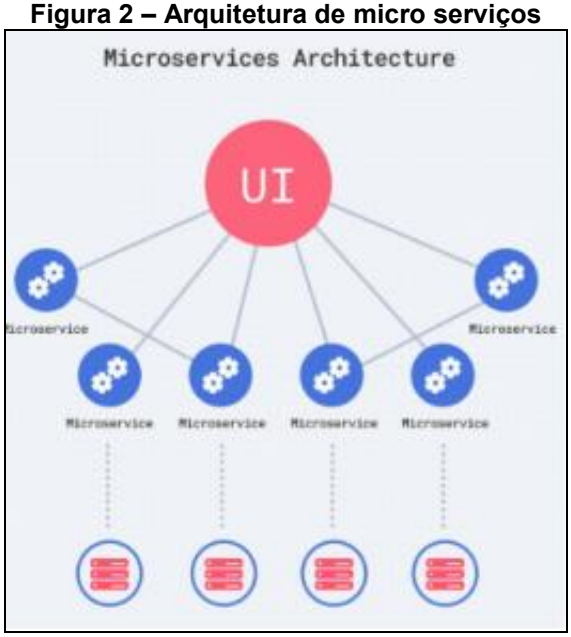
Não é um grande desafio entender que micro serviços referem-se a um sistema que deveriam estar separado em vários pedaços menores e por contexto.

Esta separação tem algumas características e benefícios específicas como indicado por MD (2019):

- Quantas linguagens ou tecnologias forem necessárias;
- Orientadas a API;
- Stateless;
- Elasticidade natural;
- Domínios bem definidos;
- Facilidade em gerenciar desenvolvimentos em paralelo.

Igualmente Sequeira *et al.* (2020) demonstra pela Figura 2 e informa que a modularidade permite uma liberdade maior sendo que caso uma parte da aplicação esteja com problema ou má formada, diferente do monólito, terá apenas uma parte do sistema afetada e justamente por ter que ser pequena permite que seja rapidamente identificado e corrigido o problema. O que não quer dizer que não teria um problema se pensarmos em algum componente principal ou primário como um *frontend* de *login*.

Figura 2 – Arquitetura de micro serviços



Fonte: Sequeira *et al.* (2020).

Assim como em monólitos não é de se esperar que micro serviços não tenham problemas. É apresentado por Sequeira *et al.* (2020) que o fato de ter um sistema baseado nesta arquitetura direciona para um sistema, ou com intenções que podem se tornar complexo com uma estrutura de testes um pouco mais intrincado

afinal cada micro serviço deve se comunicar com os outros mesmo usando linguagens totalmente distintas e desenvolvedores que não sabem como está o desenvolvimento da aplicação dos pares. Um exemplo de que micro serviço, não é sinônimo de simplicidade estrutural, são as estruturas da amazona.com e da Netflix apresentados na Figura 3.

Figura 3 – Representação da comunicação de micro serviços da Amazon e Netflix

Fonte: Autoria própria.

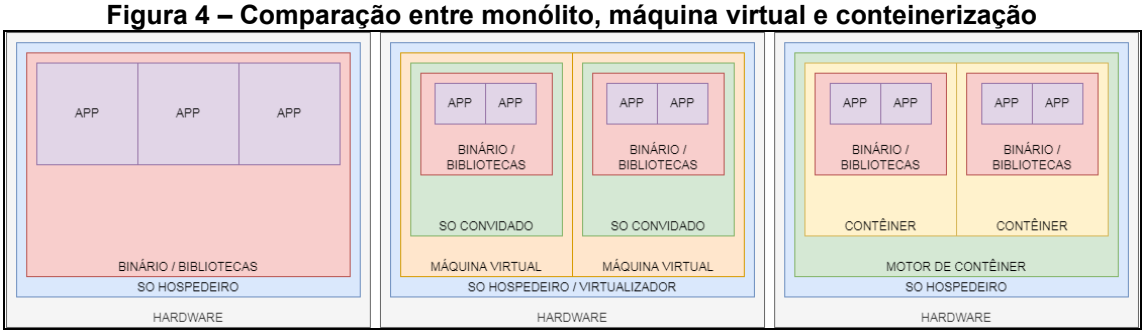
2.3 O PRECONCEITO

É bem comum o preconceito entre monólitos e micro serviços, mas como é dito por Newman (2020) que existem aqueles que acreditam que arquiteturas monolíticas são estruturas legadas e por tanto muitas vezes desconsideradas na arquitetura geral da aplicação e ignoradas ou por Sequeria (2020) que micro serviços são extremamente complexos e muito difícil se testar problemas que possam ocorrer na comunicação justamente pela singularidade do tipo de linguagem usada em cada micro serviço. Este preconceito é algo que na área de tecnologia já deveria ter sido superado afinal ambos tem vantagens, desvantagens e aplicabilidades como MD (2019) aponta.

2.4 DOCKER

Deixando um pouco de lado os monólitos, micro serviços foram possíveis apenas quando a tecnologia que permitiria este isolamento e modularidade de uma forma mais simplificada e acessível entrando a ferramenta *Docker*. Por Sequeira *et al.* (2020) um dos pontos de que permitem a velocidade de propagação do código além da velocidade nas operações é que diferente das máquinas virtuais que sobem o *kernel* e aloca *hardware* próprios os contêineres compartilham estes recursos do SO hospedeiro fazendo com que compartilhem o *kernel* e *hardware*. Outro ponto levantado é que e diferente do monólito é possível atualizar código, versão ou tipo de tecnologia sem implicar nas outras aplicações. Na Figura 4 é demonstrado de forma simplificada como são as estruturas monólitos, máquinas virtuais e contêinerizadas.

Figura 4 – Comparação entre monólito, máquina virtual e conteinerização



Fonte: Autoria própria.

2.5 KUBERNETES

Esta ferramenta como dito anteriormente tem se popularizado frente a alternativas e provavelmente considerando que várias clouds tem oferecido o K8s como PaaS (*platform-as-a-service*) justamente por ter uma lista um pouco maior de recursos disponíveis sendo que algumas serão abordadas.

Sequeira *et al.* (2020) traz alguns pontos mais aclamados da ferramenta:

- Um sistema de descoberta de serviços e balanceamento de carga;
- Orquestramento de armazenamento;
- Automação de *rollout* e *rollback*;
- Permite especificação de limitar o uso de *CPU* e memória por contêiner;
- Correção automática de contêineres que apresentem alguma condição;
- Mascaramento de configurações e dados sensíveis.

Um ponto não levantado, mas que pode ser encontrado nas documentações oficiais é a possibilidade de auto escalabilidade horizontal com base nos limites de recursos ou HPA (*Horizontal Pod Auto scaling*) em KA (2021a).

Da mesma forma que Sequeira *et al.* (2020) será apresentado brevemente os quatro componentes principais e que sem eles uma estrutura em Kubernetes seria dificultada: 1) *Node*; 2) *Deployment*; 3) *Service*; e 4) *Pod*.

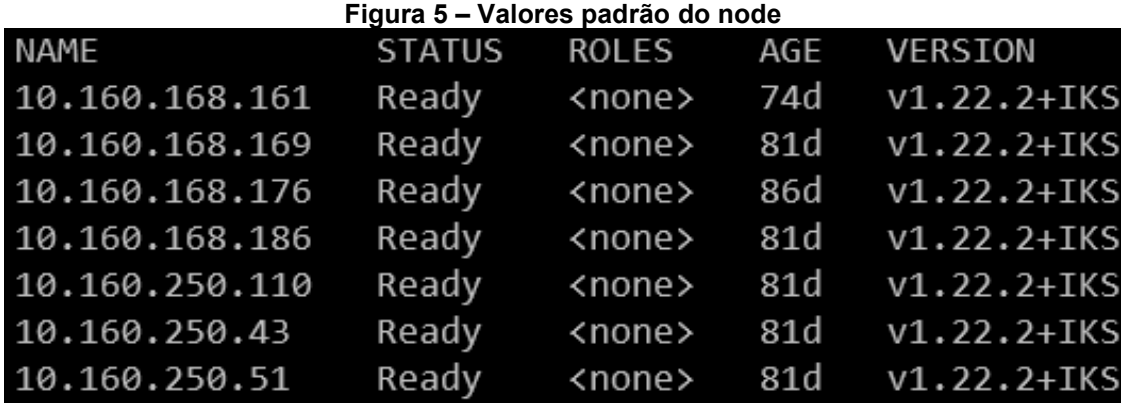
2.5.1 Node

De uma forma mais simplificada que a apresentada por Sequeira *et al.* (2020) pode-se dizer que é outro nome para servidor, seja ele físico ou virtual, sendo onde os contêineres gerados serão convertidos em *pods* e executados no *node* como se explica em KA (2021b) sendo que cada *node* é gerenciado pelo *control plane*.

Sequeira *et al.* (2020) mostra algumas das principais informações oferecidas quando se analisa os nodes, como demonstrado na Figura 5, por meio do comando *kubectl get node* são:

- *NAME*: que é o mesmo do IP e *hostname*;
- *STATUS*: que pode variar entre *Ready* (node sem problemas), *OutOfDisk* (disco sem espaço para aceitar novos pods) e Unknown (quando o node demora mais do que 40 segundos para levantar, isto pode ocorrer durante o provisionamento);
- *ROLES*: normalmente entre *Master*, quando é o *node* principal do *cluster*, e não por *KS (AKS, IKS, etc.), e <*none*> para os outros;
- *AGE*: sendo a data de adição do *node* ao cluster
- *VERSION*: a versão que está sendo executada.

Figura 5 – Valores padrão do node



NAME	STATUS	ROLES	AGE	VERSION
10.160.168.161	Ready	<none>	74d	v1.22.2+IKS
10.160.168.169	Ready	<none>	81d	v1.22.2+IKS
10.160.168.176	Ready	<none>	86d	v1.22.2+IKS
10.160.168.186	Ready	<none>	81d	v1.22.2+IKS
10.160.250.110	Ready	<none>	81d	v1.22.2+IKS
10.160.250.43	Ready	<none>	81d	v1.22.2+IKS
10.160.250.51	Ready	<none>	81d	v1.22.2+IKS

Fonte: Autoria própria.

2.5.2 Deployment

Os recursos de *deployment* são em essência os responsáveis pela gerência total dos pods e Sequeira *et al.* (2020) descreve que podem ser usados para criar novos pods, mudar versão do contêiner tal como voltar o estado anterior em caso de problemas. Um dos apontamentos do autor é que podem ser escolhidos à quantidade de réplicas, cópias dos pods, e o valor que for definido será respeitado até que o valor seja atualizado na implementação de novas configurações para aquele deployment.

A Figura 6 mostra o resultado do comando *kubectl get deploy* sendo:

- *NAME*: sendo o nome de identificação do *deployment*;
- *READY*: comparação entre quanto *pods* estão prontos e o total desejado pelo *deployment*;
- *UP-TO-DATE*: a quantidade de *pods* com a versão mais recente de contêiner;
- *AVAILABLE*: total de *pods* atendendo as chamadas

Figura 6 – Valores padrão do deployment

```
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
strapi-prod-api                     1/1     1            1           83d
```

Fonte: Autoria própria.

2.5.3 Services

Sequeira *et al.* (2020) descreve que o recurso *services* fica responsável por controlar a efemeridade dos pods, como podem ser encontrados e quais portas devem ser usadas na comunicação dos pods. As informações padrão são vistas na Figura 7 como resultado do comando *kubectl get svc* onde:

- *NAME*: que representa a identificação do *service*;
- *TYPE*: que define como será o funcionamento do acesso, LoadBalance é o mais comum para divisão de carga entre os pods;
- *CLUSTER-IP*: representando o IP interno exposto para o serviço;
- *EXTERNAL-IP*: quando tem a necessidade de ser usado o um IP público, normalmente usado em aplicações que precisam ser acessadas externamente;

- *PORT(S)*: mostra na ordem qual porta está sendo exposta internamente para o Kubernetes e a porta está sendo vista externamente pelo IP da master embora não seja uma obrigação.

Figura 7 – Valores padrão do services

Fonte: Autoria própria.

2.5.4 Pods

Finalmente os *pods* e aqui já se começa a faltar da não persistência onde Sequeira *et al.* (2020) informa que “não devem ser considerados como entidades persistentes”, ou seja, em caso de problemas o K8s tentará resolver o problema sozinho com base nas informações passadas pelo *deployment* tentando evitar alguma interação humana sendo que um dos passos é apagar a existência daquele *pod* e criar outro em seu lugar o mesmo conceito se aplica quando uma réplica é adicionada. Para a Figura 8 foi usado o comando *kubectl get pod* você pode ver:

- *NAME* sendo o nome do pod consiste sempre na estrutura nome_da_imagem_no_deployment-hash_da_imagem-hash_do_pod;
- *READY* deve estar sempre 1/1, no caso de 0/1 o Kubernetes ainda não considera o pod saudável para estar atendendo chamadas;
- *STATUS* indica o nível de saúde do pod, sendo que *Running* é sempre o estado desejado;
- *RESTARTS* indica quantas vezes o K8s teve que agir naquele pod sem destruí-lo.

Figura 8 – Valores padrão do pods

Fonte: Autoria própria.

Um ponto de atenção é que o nome destinado que aprece no campo *NAME* do *deployment*, *services* e *pod* não precisa ser o mesmo embora seja altamente recomendado que sejam para agilizar a manutenção.

3 BANCO DE DADOS EM KUBERNETES

O principal estudo identificado que teve tal sucesso durante o experimento foi de Pereira *et al.* (2020) que utilizou a estrutura do *Google Cloud Platform* (GCP) para seu experimento através dos recursos de Kubernetes *Persistent Volume* (PV) e *Persistent Volume Claim* (PVC) que em faz um mapeamento de um disco, podendo ser o local do *node* criando uma área onde os dados serão mantidos mesmo que todos os contêineres sejam apagados gerando a persistência dos dados e fugindo do conceito de efemeridade referida por Sequeira *et al.* (2020). No experimento de Pereira *et al.* (2020) usou três arquiteturas de armazenamento que iriam mostrar em algumas situações como a replicação e o processo de recuperação dos dados se comportaria, aqui vamos nos atentar ao modelo com sete “servidores” por ser a maior.

3.1 CLUSTER DE BANCO DE DADOS COM SETE PODS

O experimento de Pereira *et al.* (2020) consistiu em criar um cluster MySQL no modelo de replicação *master-slave*, único disponível para a ferramenta usada naquele momento, com sete *pods*. De mesmo modo, aponta-se a necessidade de recursos adicionais dado que a construção da replicação manual não é simples sendo eles o *KubeDB*, versão 0.12.0, e o *ProxySQL* para balanceamento de carga sendo que este poderia ser considerado como outro motivo para evitar a arquitetura persistente dado que o próprio sistema de Kubernetes tem autonomia para controle de carga, porém seria necessário dado que o sistemas de base de dados tem um entendimento diferenciado em relação a balanceamento de carga quando se trata de queries.

3.1.1 Teste com Base > 100MB

Na monografia de Pereira *et al.* (2020) tiveram algumas imagens representando o comportamento do servidor antes, durante e depois dos testes de *failover*. Os testes consistiam em demonstrar como o cluster conseguiria manter a integridade das queries.

A Figura 9 inicia mostrando o estado do cluster antes dos testes para fins de comparação com as imagens finais da estrutura. Alguns pontos foram destacados para melhor identificação da estrutura inicial.

Figura 9 – Estrutura dos servidores antes dos testes de failover

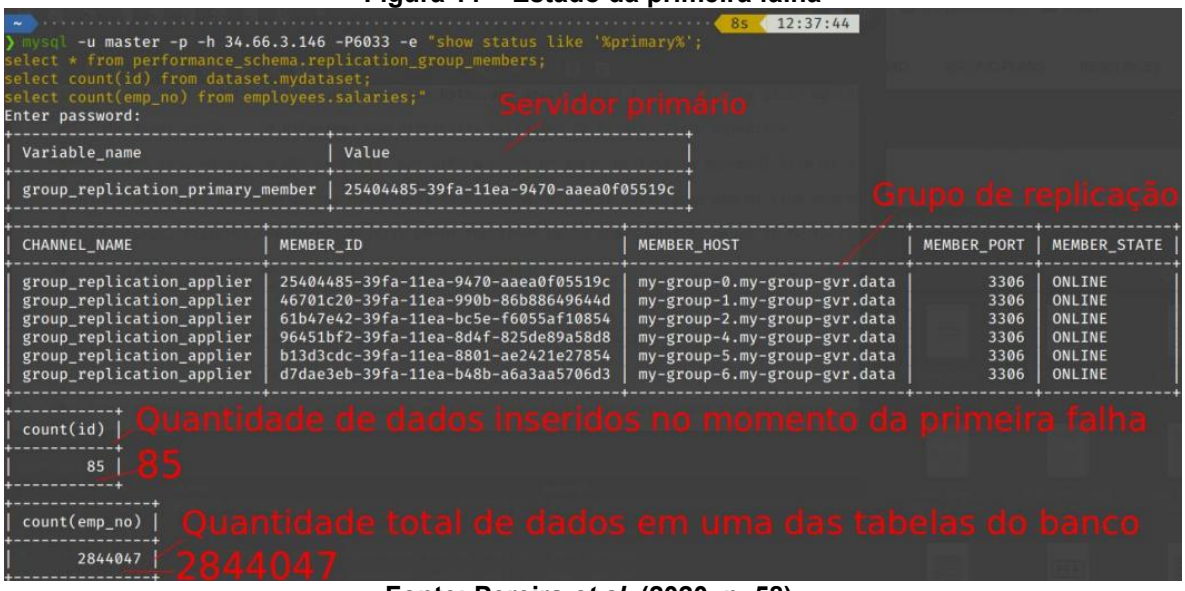
Fonte: Pereira *et al.* (2020, p. 75).

As próximas figuras representam três eventos de falhas simulando indisponibilidade. As figuras respectivamente indicam o momento da falha no cluster seguido da recuperação da falha sem necessidade de interação humana e como essa instabilidade conseguiu ser evitada impedindo que dados fossem perdidos na estrutura em caso de problema no cluster de banco de dados sendo: A) Figuras 10 e 11 primeiro evento; b) Figuras 12 e 13 segundo evento; c) Figuras 14 e 15 terceiro evento.

Figura 10 – Servidor durante primeiro failover

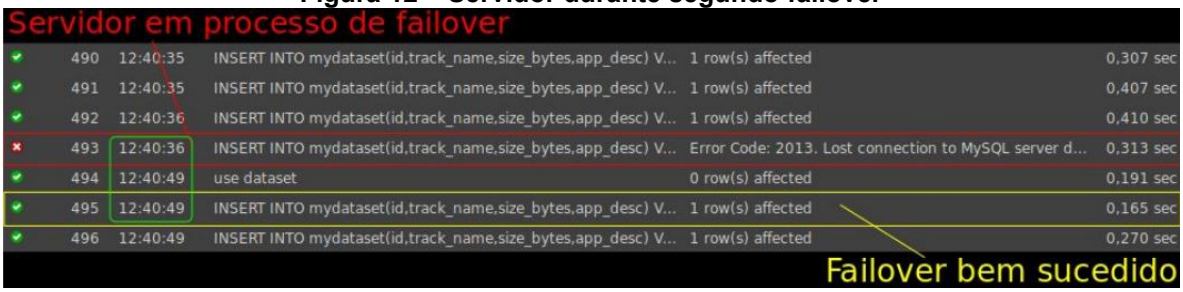
Fonte: Pereira *et al.* (2020, p. 58).

Figura 11 – Estado da primeira falha



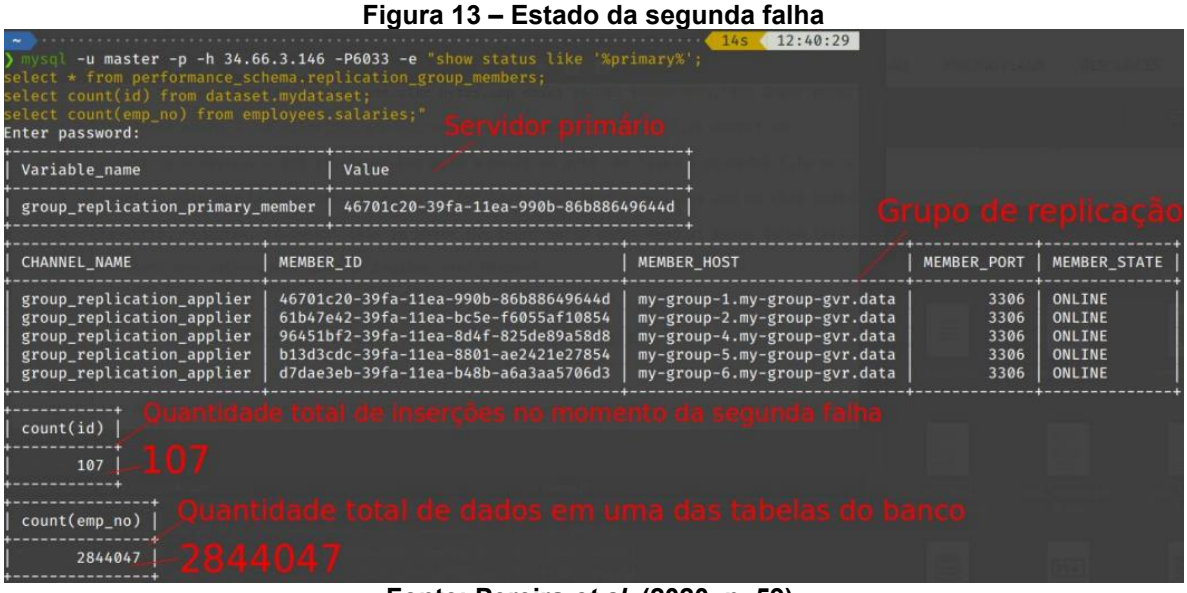
Fonte: Pereira *et al.* (2020, p. 58).

Figura 12 – Servidor durante segundo failover



Fonte: Pereira *et al.* (2020, p. 59).

Figura 13 – Estado da segunda falha



Fonte: Pereira *et al.* (2020, p. 59).

Figura 14 – Servidor durante terceiro failover

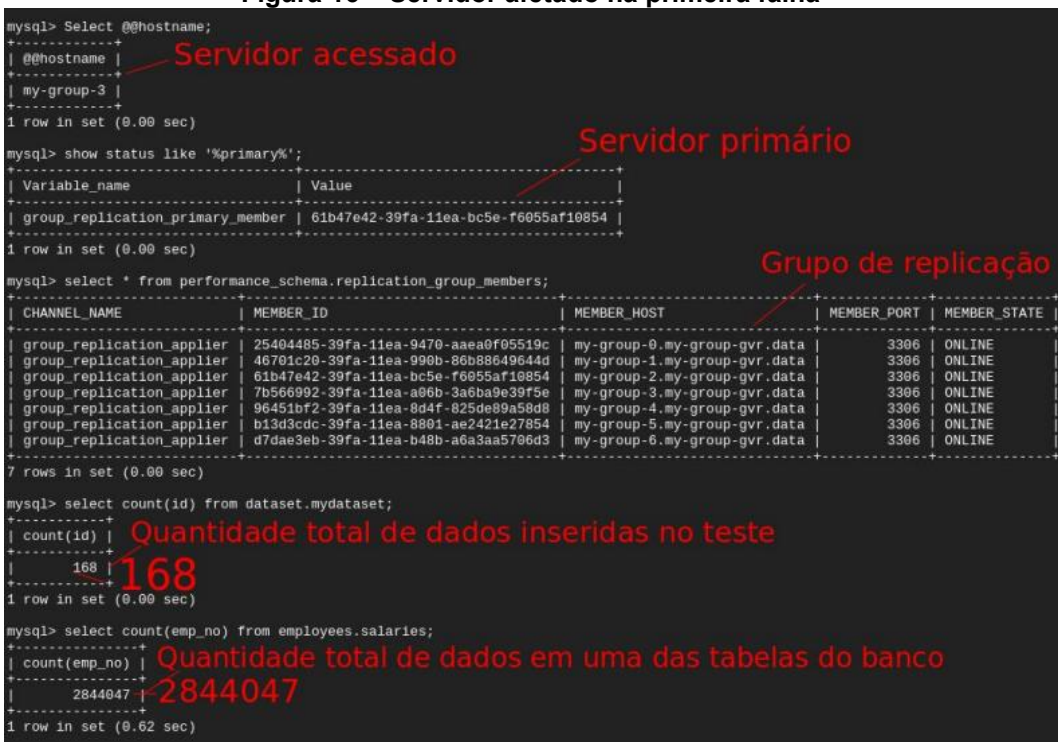
Fonte: Pereira *et al.* (2020, p. 59).

Figura 15 – Estado da terceira falha

Fonte: Pereira *et al.* (2020, p. 60).

Quanto às próximas Figuras 16, 17 e 18, são representações de como cada servidor afetado durante as falhas não tiveram problemas em continuarem íntegros após terem sido reintegrados no cluster.

Figura 16 – Servidor afetado na primeira falha



Fonte: Pereira *et al.* (2020, p. 60).

Figura 17 – Servidor afetado na segunda falha

Fonte: Pereira *et al.* (2020, p. 61).

Figura 18 – Servidor afetado na terceira falha

Fonte: Pereira *et al.* (2020, p. 61).

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

É visível que a proposta de Pereira *et al.* (2020) teve sucesso no que se diz respeito à integridade dos dados e como isto poderia ser uma solução viável. No entanto alguns pontos precisam ser levantados considerando o que já foi dito referente a uma configuração não bem planejada. O arquivo proposto na Figura 19 não existe referência à criação do PV e PVC que são os responsáveis pela armazenagem os dados de forma persistente no K8s.

Figura 19 – Dockerfile para replicação em grupo no 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
```

Fonte: Pereira *et al.* (2020, p. 80).

Um dos motivos para que não tenha sido um problema no experimento de Pereira *et al.* (2020) foi que o cluster estava se encarregando da integridade dos dados e que na estrutura com sete servidores poderia ocorrer problema simultaneamente em apenas três pods. No cenário de uma nova versão com problema ser implementada os pods seriam removidos antes que a estrutura pudesse replicar de forma saudável os dados e com o comportamento padrão do pod de ser efêmero, como é mencionado por Sequeira *et al.* (2020), os dados seriam perdidos sendo necessário uma restauração por backup do banco após a estabilidade confirmada.

Saidel-Keesing (2018), assim como Pereira *et al.* (2020) que recomenda inclusive um servidor fora da estrutura Kubernetes em caso de desastre no cluster contêinerizado, alerta no final do seu artigo que por mais que Kubernetes seja uma ferramenta incrível com vários benefícios e uma infinidade de aplicabilidade não pode ser considerado uma solução para todos os nossos problemas porque assim como os monólitos tem seus pontos fortes e fracos.

Outro artigo ligado ao movimento DevOps que tente a ser mais a favor de soluções em K8s e micro serviços, MD (2019), expõe que monólitos não deve ser desconsiderado justamente porque ainda tem utilidade em cenários como acesso controlado e restrito.

5 CONCLUSÃO

Não é complicado detectar que por mais que o Kubernetes seja uma ferramenta incrível e extremamente poderosa para aplicações e novas soluções de mercado não é algo milagroso que vai resolver todos os nossos problemas como desempenho, escalabilidade, integridade, segurança, boas práticas e etc. como aconteceu com a virtualização e a Cloud.

No que compete a esta monografia, acredita-se que na melhor das circunstâncias aplicações que demandam rápida escalabilidade sem preocupação com a integridade dos dados, como aplicações Web ou bancos de dados não relacionais para cache de resultados complexos de bancos de dados relacionais, Kubernetes seria a ferramenta mais indicada. Enquanto sistemas onde a integridade dos dados é uma preocupação ou em companhias que não possuem pessoas com conhecimento ou familiaridade em sistemas contêinerizados o mais indicado continua sendo o clássico monólito, mesmo que em máquinas virtuais seja *on premise* ou em *cloud*.

Ainda referente à monografia admite-se que são possíveis alguns estudos futuros como:

- Reanálise considerando modelo híbrido, Kubernetes e Máquina virtual ou monólito, como uma solução mais robusta;
- Análise mais profunda quanto ao tempo de restauração de um serviço monolítico em contêinerização;
- Cenário Web favorável a monólito.

REFERÊNCIAS

CORRÊA, Vagner Marcondes; SOARES, Nathalia Maria. **O papel da comunicação dentro do ambiente organizacional.** Revista Interface Tecnológica, v. 17, n. 1, p. 699-707, 2020.

KA. **Horizontal pod auto scaling**. Copyright© The Kubernetes Authors (KA), última atualização: 09 dez. 2021a. Disponível em: <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>. Acesso em: 03 dez. 2021.

KA. **Nodes**. Última atualização: Copyright©The Kubernetes Authors (KA), última atualização: 09 dez. 2021b. Disponível em: <https://kubernetes.io/docs/concepts/architecture/nodes/>. Acesso em: 03 dez. 2021.

MD. **Missão 03**: Monólitos & microserviços. Copyright© 2019 Missão DevOps (MD), 2019. Disponível em: <http://missaodevops.com.br/docs/warrior_monolito_microservicos.html>. Acesso em: 20 dez. 2021.

NEWMAN, Sam. **Migrando sistemas monolíticos para microserviços**: Padrões evolutivos para transformar seu sistema monolítico. Novatec Editora, 2020.

OSZTERTÁG, Dávid. **Docker**: Life before and after. 2020. Disponível em: <https://avatao.com/blog-life-before-docker-and-beyond/>. Acesso em: 02 dez. 2021.

PEREIRA, Mateus Augusto; *et al*. **Ambiente de alta disponibilidade para banco de dados utilizando MySQL em containers orquestrados pelo kubernetes**. 2020.

RISINGSTACK. **A brief history of multicluster kubernetes**. Copyright© RisingStack, publicado em: 11 out. 2021. Disponível em: <https://blog.risingstack.com/the-history-of-kubernetes/>. Acesso em: 03 dez. 2021.

SAIDEL-KEESING, Maish. **Kubernetes is not the silver bullet**. Copyright© IOD Cloud Technologies Research Ltd., publicado em: 26 jan. 2018. Disponível em: <https://iamondemand.com/blog/kubernetes-not-silver-bullet/>. Acesso em: 24 dez. 2021.

SEQUEIRA, António Miguel Guedes de Campos dos Santos; *et al*. **Estender a plataforma de microserviços elásticos**. 2020. Tese de Doutorado. Universidade de Coimbra.

TLN. **A Brief Look at the Roots of Linux Containers**. Copyright© The Linux Foundation (TLN), publicado em: 27 mai. 2017. Disponível em: <https://www.linuxfoundation.org/blog/a-brief-look-at-the-roots-of-linux-containers/>. Acesso em: 02 dez. 2021.