UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RAPHAEL HENRIQUE DE LUCENA AGRA

ANÁLISE COMPARATIVA ENTRE DOCKER SWARM E KUBERNETES EM AMBIENTES DE NUVEM

CAMPINA GRANDE - PB

2024

RAPHAEL HENRIQUE DE LUCENA AGRA

ANÁLISE COMPARATIVA ENTRE DOCKER SWARM E KUBERNETES EM AMBIENTES DE NUVEM

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador : Reinaldo Cezar de Morais Gomes

CAMPINA GRANDE - PB

2024

RAPHAEL HENRIQUE DE LUCENA AGRA

ANÁLISE COMPARATIVA ENTRE DOCKER SWARM E KUBERNETES EM AMBIENTES DE NUVEM

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Reinaldo Cezar de Morais Gomes
Orientador – UASC/CEEI/UFCG

Thiago Emmanuel Pereira da Cunha Silva
Examinador – UASC/CEEI/UFCG

Francisco Vilar Brasileiro
Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 15 de Maio de 2024.

CAMPINA GRANDE - PB

RESUMO

Os contêineres são uma solução amplamente adotada na indústria de tecnologia, graças à sua flexibilidade e escalabilidade. Nesse contexto, a gestão eficiente dos contêineres é essencial para garantir a disponibilidade e o desempenho do sistema. Por esse motivo, há uma grande variedade de ferramentas de gerenciamento disponíveis, cada uma adequada para diferentes casos de uso. O objetivo deste estudo é analisar e comparar o desempenho de duas das principais ferramentas de orquestração de containers, o Docker Swarm e o Kubernetes, em ambientes de nuvem.

Para realizar esta análise comparativa de desempenho, serão realizados testes de carga em ambas as plataformas e os resultados serão comparados. Os testes serão executados em um ambiente de nuvem pública, com diferentes tamanhos de cluster e cargas de trabalho.

Os critérios de avaliação serão seis: complexidade de configuração, tempo de implantação de containers, escalabilidade, uso de recursos de CPU e memória e disponibilidade.

Os resultados dessa análise comparativa de desempenho permitirão contribuir para a identificação de qual ferramenta é mais adequada para diferentes cenários e cargas de trabalho.

ANÁLISE COMPARATIVA ENTRE DOCKER SWARM E KUBERNETES EM AMBIENTES DE NUVEM

ABSTRACT

Containers are a widely adopted solution in the technology industry, thanks to their flexibility and scalability. In this context, efficient container management is essential to ensure system availability and performance. For this reason, there is a wide variety of management tools available, each suited to different use cases. The objective of this study is to analyze and compare the performance of two of the main container orchestration tools, Docker Swarm and Kubernetes, in cloud environments.

To conduct this comparative performance analysis, load tests will be performed on both platforms, and the results will be compared. The tests will be executed in a public cloud environment, with different cluster sizes and workloads.

The evaluation criteria will be sixfold: configuration complexity, container deployment time, scalability, CPU and memory resource usage, and availability.

The results of this comparative performance analysis will help identify which tool is more suitable for different scenarios and workloads.

Análise Comparativa Entre Docker Swarm e Kubernetes em Ambientes de Nuvem.

Raphael Henrique de Lucena Agra
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande, Paraíba - Brasil
raphael.agra@ccc.ufcg.edu.br

Reinaldo Cezar de Morais Gomes
Departamento de Sistemas e Computação
Universidade Federal de Campina Grande
Campina Grande, Paraíba - Brasil
reinaldo@computacao.ufcg.edu.br

Resumo

Os contêineres são uma solução amplamente adotada na indústria de tecnologia, graças à sua flexibilidade e escalabilidade. Nesse contexto, a gestão eficiente dos contêineres é essencial para garantir a disponibilidade e o desempenho do sistema. Por esse motivo, há uma grande variedade de ferramentas de gerenciamento disponíveis, cada uma adequada para diferentes casos de uso. O objetivo deste estudo é analisar e comparar o desempenho de duas das principais ferramentas de orquestração de containers, o Docker Swarm e o Kubernetes, em ambientes de nuvem.

Para realizar esta análise comparativa de desempenho, serão realizados testes de carga em ambas as plataformas e os resultados serão comparados. Os testes serão executados em um ambiente de nuvem pública, com diferentes tamanhos de cluster e cargas de trabalho.

Os critérios de avaliação serão seis: complexidade de configuração, tempo de implantação de containers, escalabilidade, uso de recursos de CPU e memória e disponibilidade.

Os resultados dessa análise comparativa de desempenho permitirão contribuir para a identificação de qual ferramenta é mais adequada para diferentes cenários e cargas de trabalho.

Palavras-chave

Contêineres; docker; kubernetes; orquestração

Repositório com os artefatos utilizados nos experimentos:
https://github.com/raphaelhla/TCC

1 Introdução

A computação em nuvem tem se estabelecido como uma abordagem essencial para a implantação e gerenciamento de aplicações modernas. Nesse contexto, o uso de containers tem ganhado destaque devido à sua capacidade de fornecer um ambiente isolado e portátil para a execução de aplicações, facilitando a escalabilidade e a implantação consistente em diferentes ambientes.

No entanto, à medida que as aplicações se tornam mais complexas e exigem uma infraestrutura de nuvem distribuída, surge a necessidade de ferramentas de orquestração de containers para gerenciar eficientemente os clusters e as cargas de trabalho. Entre as soluções mais populares atualmente, destacam-se o Docker Swarm e o Kubernetes, que oferecem recursos avançados de orquestração e gerenciamento de containers em ambientes de nuvem.

Diante desse cenário, surge a pergunta: qual dessas plataformas de orquestração de containers é a mais adequada em termos de desempenho e eficiência operacional? A resposta a essa pergunta é crucial para empresas e desenvolvedores que buscam adotar uma solução de orquestração de containers em seus ambientes de nuvem.

Este trabalho tem como objetivo realizar uma análise comparativa de desempenho entre o Docker Swarm e o Kubernetes em ambientes de nuvem. Serão considerados critérios como tempo de implantação de containers, escalabilidade, uso de recursos de CPU e memória, e disponibilidade. Além disso, aspectos relacionados à facilidade de configuração e gerenciamento de clusters também serão levados em consideração.

Através dessa análise comparativa, espera-se contribuir para o entendimento das características e diferenças entre as plataformas, auxiliando na tomada de decisão informada sobre a escolha da solução mais adequada para cada cenário de uso. Os resultados e conclusões obtidos poderão ajudar desenvolvedores e organizações que buscam otimizar o desempenho e a eficiência operacional de suas aplicações em ambientes de nuvem, por meio da escolha da plataforma de orquestração de containers mais adequada.

2 contextualização

2.1 Máquina Virtual

O termo "máquina virtual" refere-se a um ambiente de execução virtualizado que funciona como uma cópia isolada de um sistema operacional e dos recursos de hardware subjacentes. Ela permite a execução de múltiplos sistemas operacionais e aplicativos em um único computador físico, criando uma camada de abstração entre o hardware real e o software em execução. Em uma máquina virtual, um software chamado hipervisor é responsável por criar e gerenciar as máquinas virtuais. O hipervisor permite que cada máquina virtual tenha seu próprio conjunto de recursos virtuais, como CPU, memória, disco rígido e interfaces de rede. Cada máquina virtual é executada como se fosse um sistema operacional independente, com sua própria instância do sistema operacional e aplicativos instalados.

2.2 Container

Um contêiner é uma ambiente de execução virtualizado em nível de sistema operacional, onde o sistema operacional subjacente é compartilhado entre os contêineres, permitindo que eles sejam mais leves e eficientes do que outras formas de virtualização tradicional, como máquinas virtuais. Cada contêiner é isolado dos demais e do

sistema host, o que significa que ele possui seus próprios processos, sistema de arquivos e recursos de rede, mas todos compartilham o mesmo kernel do sistema operacional hospedeiro.

2.3 Docker

Docker é uma ferramenta que permite a criação, o gerenciamento e a execução de aplicativos em contêineres. Um contêiner é uma unidade isolada de software que empacota todo o ambiente necessário para que um aplicativo seja executado, incluindo o código, as dependências e as configurações. Essa abordagem de contêinerização permite que os aplicativos sejam executados de maneira consistente e confiável em qualquer ambiente, independentemente das diferenças de infraestrutura subjacente.

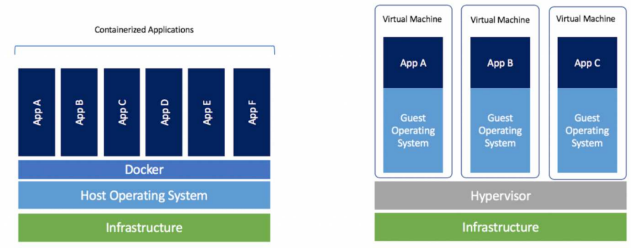

Figura 1: Contêiner vs Máquina Virtual.

2.4 Amazon Web Services (AWS)

A Amazon Web Services, comumente conhecida como AWS, é uma plataforma de computação em nuvem oferecida pela Amazon. Ela fornece uma ampla gama de serviços de computação, armazenamento, banco de dados, análise, redes, aprendizado de máquina, segurança e muito mais, que permitem às empresas e indivíduos implantar aplicativos e recursos de TI de maneira escalável e flexível na nuvem. A AWS oferece uma infraestrutura global que abrange várias regiões geográficas, permitindo que os usuários implantem seus aplicativos em data centers localizados em todo o mundo. Isso possibilita alta disponibilidade, escalabilidade e redundância para aplicativos críticos. Além disso, a AWS fornece uma série de ferramentas e serviços para gerenciar e otimizar recursos na nuvem, permitindo que os usuários paguem apenas pelo que utilizam, o que torna a plataforma econômica para empresas de todos os tamanhos.

2.5 Cluster

Um cluster é um conjunto de computadores interconectados que trabalham juntos como se fossem um único sistema coeso. Esses computadores, chamados de "nós", são conectados entre si através de uma rede e são configurados para colaborar em tarefas específicas. A ideia por trás de um cluster é distribuir a carga de trabalho e melhorar o desempenho, a confiabilidade e a disponibilidade do sistema.

Em um cluster, os nós podem ser computadores físicos ou máquinas virtuais, e cada nó geralmente executa uma cópia do mesmo software. O software de gerenciamento de cluster coordena as operações entre os nós, distribui tarefas de processamento e garante que o sistema funcione de forma eficiente e resiliente.

3 Orquestração de Contêineres

A orquestração de contêineres é o processo de automatizar a implantação, o gerenciamento, a escala e a rede dos containers em ambientes distribuídos e de grande escala.

A orquestração de conteineres é especialmente útil em ambientes de computação em nuvem e microserviços, onde os aplicativos são compostos por vários serviços independentes que precisam ser escalados e gerenciados de forma dinâmica e resiliente. A orquestração de contêineres simplifica a implantação e operação desses aplicativos, fornecendo recursos avançados de automação e gerenciamento de ciclo de vida.

3.1 Benefícios da Orquestração de Contêineres

O uso de ferramentas de orquestração de contêineres oferece uma série de benefícios para o desenvolvimento e a implantação de aplicativos. A seguir, estão alguns dos principais benefícios da orquestração de contêiners:

Gerenciamento eficiente de recursos

As ferramentas de orquestração de contêineres permitem o gerenciamento eficiente de recursos de infraestrutura. Elas automatizam tarefas como provisionamento, dimensionamento e distribuição de contêineres em vários nós ou hosts, garantindo o uso otimizado dos recursos disponíveis.

Escalabilidade horizontal

Com a orquestração de contêineres, é possível escalar horizontalmente os aplicativos, adicionando ou removendo contêineres de acordo com a demanda. Isso permite que os aplicativos sejam dimensionados de forma dinâmica para lidar com picos de tráfego e garantir um desempenho consistente.

Alta disponibilidade

As ferramentas de orquestração de contêineres oferecem recursos avançados para garantir a alta disponibilidade dos aplicativos. Elas monitoram constantemente o estado dos contêineres e, em caso de falhas, automaticamente os reiniciam ou redirecionam o tráfego para instâncias saudáveis. Isso contribui para a resiliência dos aplicativos e reduz o tempo de inatividade.

Implantação simplificada

As ferramentas de orquestração de contêineres simplificam o processo de implantação de aplicativos. Elas permitem a definição de configurações e dependências em arquivos de manifesto, facilitando a replicação consistente do ambiente de execução. Além disso, elas automatizam tarefas de implantação, como a criação de contêineres, a configuração de redes e o balanceamento de carga.

Monitoramento e diagnóstico

As ferramentas de orquestração de contêineres fornecem recursos avançados de monitoramento e diagnóstico. Elas oferecem métricas e registros detalhados sobre o desempenho dos aplicativos, o uso de recursos e o estado dos contêineres. Isso permite a detecção de problemas e a tomada de medidas corretivas de forma proativa.

3.2 Ferramentas de orquestração

Existem várias ferramentas de orquestração de contêineres disponíveis, porém o foco deste trabalho será nas duas mais famosas: Kubernetes e Docker Swarm. A seguir, é dado uma definição geral sobre cada uma delas.

3.2.1 Kubernetes

Kubernetes é uma plataforma de orquestração de contêineres de código aberto amplamente utilizada para automatizar, dimensionar e gerenciar aplicativos em contêineres. Ele fornece um ambiente para implantar, dimensionar e gerenciar aplicativos em contêineres de forma eficiente, oferecendo recursos avançados, como balanceamento de carga, escalonamento automático, auto recuperação e distribuição de tráfego.

No Kubernetes, os contêineres são agrupados em unidades lógicas chamadas "pods". Um pod é a menor unidade implantável em Kubernetes e representa um ambiente de execução para um ou mais contêineres relacionados, compartilhando recursos como rede e armazenamento. Os pods são criados para suportar aplicações compostas por múltiplos contêineres que precisam ser executados juntos e se comunicar entre si.

Os nós no cluster Kubernetes são as instâncias de máquinas físicas ou virtuais que compõem o cluster. Cada nó executa o software Kubernetes (o kubelet) e é responsável por hospedar e executar contêineres. Existem dois tipos principais de nós em um cluster Kubernetes: Nós de Controle (Control Plane Node) e Nós de Trabalho (Worker Node).

O nó de controle é responsável por gerenciar e coordenar as operações do cluster Kubernetes. Ele inclui vários componentes principais, como o API server, o scheduler, o controller-manager e o etcd. O API server é a interface central para interagir com o cluster, enquanto o scheduler aloca contêineres em nós de trabalho com base nos requisitos de recursos e políticas de agendamento. O controller-manager monitora o estado do cluster e faz ajustes para garantir que o estado desejado seja mantido. Por fim, o etcd é um armazenamento de dados distribuído que mantém o estado do cluster. Um cluster Kubernetes deve ter pelo menos um nó de controle, mas é comum ter vários para garantir alta disponibilidade.

O nó de trabalho é onde os contêineres são realmente executados. Ele executa o kubelet, que é um agente que se comunica com o nó de controle e gerencia os contêineres em execução no nó. Os nós de trabalho são onde a carga de trabalho real é processada e são escalados horizontalmente para lidar com a demanda de recursos. Um cluster Kubernetes pode ter vários nós de trabalho para distribuir a carga de trabalho e aumentar a capacidade de processamento.

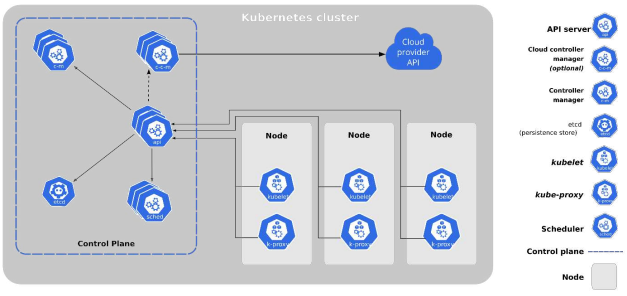

Figura 2: Arquitetura do cluster Kubernetes.

3.2.2 Docker Swarm

Docker Swarm é uma solução de orquestração nativa do Docker que permite criar, gerenciar e escalar clusters de contêineres de forma fácil e eficiente. O Docker Swarm possibilita a distribuição e a execução de contêineres em um ambiente distribuído, facilitando o gerenciamento de aplicações compostas por múltiplos contêineres em diversos nós de um cluster.

o Docker Swarm é conhecido por sua simplicidade de configuração e operação. Isso se deve ao fato de que muitos dos seus recursos de orquestração essenciais estão integrados e prontos para uso imediato, sem a necessidade de configurações complexas adicionais.

Os nós no cluster Docker Swarm são as instâncias de máquinas físicas ou virtuais que compõem o cluster. Cada nó executa uma instância do Docker Engine, que é o software responsável por executar e gerenciar os contêineres. Existem dois tipos principais de nós em um Docker Swarm: Nós de Gerenciamento (Manager Node) e Nós de Trabalho (Worker Node).

Um nó de gerenciamento é responsável por coordenar as atividades do cluster, como a distribuição de contêineres e o balanceamento de carga. Ele mantém o estado atual do cluster e coordena a comunicação entre os nós de trabalho e os clientes do Docker. Um cluster Docker Swarm deve ter pelo menos um nó de gerenciamento, mas é comum ter vários para garantir alta disponibilidade.

Um nó de trabalho é onde os contêineres são realmente executados. Eles executam as tarefas atribuídas pelo nó de gerenciamento e hospedam os contêineres que compõem as aplicações implantadas no cluster. Um cluster Docker Swarm pode ter vários nós de trabalho para distribuir a carga de trabalho e aumentar a capacidade de processamento.

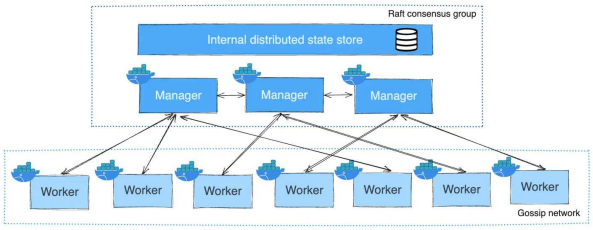

Figura 3: Arquitetura do cluster Docker Swarm.

4 METODOLOGIA

Nesse trabalho, será feita uma análise comparativa entre o Docker Swarm e o Kubernetes. A seguir será detalhado como essa análise será feita.

4.1 Configuração

Para realizar a comparação, utilizaremos recursos computacionais provenientes da infraestrutura de computação em nuvem da AWS (Amazon Web Services). Na AWS, serão criados dois clusters, um executando Kubernetes e o outro Docker Swarm.

Para a criação dos clusters, será utilizado o serviço Amazon EC2 com instâncias do tipo `t2.small` ou `t2.xlarge`. A tabela 1 mostra as configurações de cada tipo de instância utilizada.

Tipo	Memória	vCPU	Armazenamento
t2.small	2 GB	1	8 GB
t2.xlarge	16 GB	4	8 GB

Tabela 1: Configurações das Instâncias EC2 utilizadas

Cada instância EC2 utilizará a imagem Ubuntu Server 22.04, e apenas os softwares necessários para a execução de cada orquestrador serão instalados.

O Kubernetes utilizado será o mais recente, na versão 1.29.4. O cluster Kubernetes será configurado da seguinte forma:

- 1 Nó de controle (Control Plane Node)
- 2 Nós de trabalho (Worker Node)

O Docker Swarm utilizado será o mais recente, na versão 26.1.1. O cluster Docker Swarm será configurado da seguinte forma:

- 1 Nó de gerenciamento (Manager Node)
- 2 Nós de trabalho (Worker Node)

Cada cluster será inicialmente configurado com nós de trabalho do tipo `t2.small`. Em seguida, esses nós serão substituídos por novos nós do tipo `t2.xlarge`. Dessa forma, será possível avaliar o desempenho e escalabilidade de cada ferramenta de orquestração de containers em diferentes cenários de recursos computacionais.

Para realizar a comparação entre as ferramentas, foi desenvolvido uma aplicação de teste utilizando a linguagem Node.js em conjunto com o framework Express.js. A aplicação é chamada de Ping Pod e possui um servidor web em execução. O código da aplicação está disponível no github desse artigo.

Cada contêiner orquestrado estará utilizando uma imagem da aplicação Ping Pod.

4.2 Técnicas e métricas

Talvez fazer a divisão da comparação em testes e análises

4.2.1 Complexidade de configuração

Comparar a complexidade de configuração das duas plataformas.

4.2.2 Tempo de inicialização

Medir tempo necessário para iniciar contêineres e implantar a aplicação em cada plataforma. Isso inclui o tempo necessário para baixar imagens e iniciar contêineres.

4.2.3 Utilização de recursos

Monitorar o uso de recursos (CPU, memória, armazenamento) em cada nó do cluster e avaliar como cada plataforma distribui e otimiza o uso desses recursos.

4.2.4 Escalabilidade da aplicação

Verificar como cada plataforma lida com a escalabilidade da aplicação. Isso envolve aumentar o número de réplicas da aplicação.

4.2.5 Recuperação de falhas

Avaliar como cada plataforma lida com a recuperação de falhas em containeres e falhas em nós.

4.2.6 Upgrade de versão

Comparação da eficiência processo para realizar a implantação de uma nova versão da aplicação em cada plataforma

5 RESULTADOS

5.1 Complexidade de configuração

A configuração inicial dos nós do cluster Kubernetes possui vários componentes, como kubeadm, kubelet, kube-proxy, kubectl e containerd, que precisam ser instalados em cada nó do cluster. Isso envolveu a instalação de pacotes específicos, configuração de repositórios de pacotes e configurações adicionais do sistema operacional. Além disso, no cluster Kuberntes também foi necessário configurar o Container Network Interface para permitir a comunicação de rede entre os pods do cluster

Para fazer a configuração inicial nos nós do cluster Docker Swarm, foi necessário apenas instalar o Docker Engine em todos os nós do cluster.

Após a configuração inicial dos nós, em ambas as ferramentas, basta inicializar um nó como líder e adicionar nós de trabalho conforme necessário.

Portanto, no quesito de complexidade de configuração, o Docker Swarm apresentou uma configuração mais fácil e simples do que o Kubernetes.

5.2 Tempo de inicialização

A seguir será exibido o resultado coletado dos testes do tempo de inicialização dos contêineres com as duas ferramentas em clusters com nós do tipo t2.small e t2.xlarge.

	t2.small		t2.xlarge	
Contêineres	Swarm	Kubernetes	Swarm	Kubernetes
1	1,08s	1,40s	0,80s	1,27s
10	4,23s	3,51s	1,63s	2,71s
25	11,38s	8,06s	3,79s	4,45s
50	26,9s	15,33s	8,11s	7,97s
100	-	-	17,14s	17,10s
150	-	-	26,60s	25,14s
200	-	-	35,78s	34,47s

Tabela 2: Comparação do tempo de inicialização de contêineres em instâncias t2.small e t2.xlarge

Para instâncias menores, como as t2.small, observamos que ambos Docker Swarm e Kubernetes apresentam um tempo de inicialização bem parecido. No entanto, à medida que o número de contêineres aumenta, o Kubernetes mostra maior eficiência. Por exemplo, com 50 contêineres, o Kubernetes inicializa em 15,33 segundos, bastante à frente dos 26,9 segundos do Docker Swarm. Esta diferença sugere que o Kubernetes pode ser mais adequado para lidar com maior carga em máquinas de menor capacidade.

Quando analisamos as instâncias t2.xlarge, notamos uma melhoria significativa nos tempos de inicialização para ambos os sistemas, refletindo a vantagem de ter mais recursos disponíveis. Interessantemente, as diferenças de tempo entre Docker Swarm e Kubernetes tornam-se menos pronunciadas. Com 100 contêineres, por exemplo, ambos os sistemas têm tempos quase idênticos de cerca de 17 segundos. Isso demonstra que, com recursos adequados, ambos os sistemas podem escalar eficientemente.

5.3 Consumo de Memória

A seguir será exibido o resultado coletado dos teste do consumo de memória dos contêineres com as duas ferramentas em clusters com nós do tipo t2.small e t2.xlarge.

	t2.small		t2.xlarge	
Contêineres	Swarm	Kubernetes	Swarm	Kubernetes
1	12 MB	12 MB	12 MB	13 MB
10	164 MB	169 MB	156 MB	156 MB
25	415 MB	473 MB	395 MB	392 MB
50	870 MB	943 MB	778 MB	818 MB
100	-	-	1568 MB	1687 MB
150	-	-	2379 MB	2606 MB
200	-	-	3166 MB	3526 MB

Tabela 3: Comparação do uso de memória de contêineres em instâncias t2.small e t2.xlarge

Para instâncias menores, como as t2.small, observamos que ambos Docker Swarm e Kubernetes apresentam um consumo de memória relativamente baixo para um único contêiner, com ambos consumindo cerca de 12 MB. No entanto, à medida que o número de contêineres aumenta, o Kubernetes tende a consumir um pouco mais de memória do que o Docker Swarm. Por exemplo, com 50 contêineres, o Kubernetes consome 943 MB, enquanto o Docker Swarm consome 870 MB. Isso pode sugerir que o Kubernetes tem uma sobrecarga adicional de gerenciamento em relação ao Docker Swarm em ambientes com recursos limitados.

Ao passar para instâncias maiores, como as t2.xlarge, ambos os sistemas mostram um aumento significativo no consumo de memória à medida que o número de contêineres aumenta. No entanto, o Kubernetes demonstra um consumo de memória geralmente maior em comparação com o Docker Swarm. Por exemplo, com 100 contêineres, o Kubernetes consome 1687 MB, enquanto o Docker Swarm consome 1568 MB. Esta diferença torna-se mais pronunciada com um maior número de contêineres, com o Kubernetes consumindo consistentemente mais memória do que o Docker Swarm em todas as contagens de contêineres analisadas.

5.4 Consumo de CPU

A seguir será exibido o resultado coletado dos testes do consumo de CPU dos contêineres com as duas ferramentas em clusters com nós do tipo t2.small e t2.xlarge.

	t2.small		t2.xlarge	
Contêineres	Swarm	Kubernetes	Swarm	Kubernetes
1	0,7%	5,0%	0,1%	1,3%
10	1,4%	5,1%	0,2%	1,7%
25	2,0%	6,0%	0,4%	2,8%
50	2,2%	6,0%	0,8%	4,2%
100	-	-	1,4%	4,2%
150	-	-	2,4%	4,7%
200	-	-	3,3%	5,2%

Tabela 4: Comparação do consumo de CPU de contêineres em instâncias t2.small e t2.xlarge

Para instâncias menores, como as t2.small, observamos uma diferença notável no consumo de CPU entre Docker Swarm e Kubernetes. Notavelmente, o Kubernetes mostra consistentemente um consumo de CPU mais alto em comparação com o Docker Swarm em todos os cenários de carga. Por exemplo, com 50 contêineres, o Kubernetes consome 6,0% da CPU, enquanto o Docker Swarm consome apenas 2,2%.

Ao passar para instâncias maiores, como as t2.xlarge, ambos Docker Swarm e Kubernetes mostram um aumento proporcional no consumo de CPU à medida que o número de contêineres aumenta. No entanto, o padrão observado anteriormente persiste, com o Kubernetes consumindo mais CPU do que o Docker Swarm em todos os cenários analisados.

Observou-se que o uso de CPU dos workers se manteve constante conforme o número de contêineres aumentou. Os workers demandaram muita CPU apenas enquanto o cluster estava subindo os contêineres, mas quando os contêineres ficavam todos os estado 'Running', o uso de CPU estabilizava, porém isso pode variar de acordo com o tipo de aplicação que está sendo orquestrada.

5.5 Escalabilidade da aplicação

Tanto o Docker Swarm quanto o Kubernetes oferecem suporte à escalabilidade horizontal, um componente fundamental para lidar com cargas de trabalho variáveis e garantir a disponibilidade e desempenho das aplicações em contêineres. Ambos os sistemas permitem que os usuários dimensionem suas aplicações para cima ou para baixo, adicionando ou removendo instâncias conforme necessário. No entanto, há diferenças significativas em como cada plataforma aborda e implementa essa funcionalidade.

No Docker Swarm, a escalabilidade horizontal é alcançada por meio da replicação de serviços. Os usuários podem definir o número desejado de réplicas para cada serviço, e o Docker Swarm se encarregará de distribuir essas réplicas entre os nós disponíveis no cluster. Embora isso permita uma escalabilidade eficaz, o Docker Swarm não oferece recursos nativos avançados de autoescala. Para implementar autoescala no Docker Swarm, os usuários geralmente precisam configurar scripts externos de monitoramento e automação para ajustar dinamicamente o número de réplicas com base em métricas de desempenho ou carga de trabalho.

Por outro lado, o Kubernetes oferece recursos avançados de autoescala, como o Horizontal Pod Autoscaler (HPA). O HPA monitora a utilização de recursos dos Pods em execução, como CPU ou memória, e ajusta automaticamente o número de réplicas com base nessas métricas. Isso permite que as aplicações escalonem dinamicamente de acordo com a demanda, garantindo que haja sempre recursos suficientes para lidar com picos de tráfego ou carga de trabalho.

5.6 Recuperação de falhas

5.6.1 Falha em um dos nós de trabalho no cluster:

No Docker Swarm, a recuperação de falhas de nós é tratada de forma relativamente simples. Quando um nó falha, o Docker Swarm detecta automaticamente a falha e redistribui os serviços que estavam em execução no nó falho para outros nós disponíveis no cluster. Isso é possível devido à arquitetura de cluster distribuído do Docker

Swarm, que permite que os serviços sejam escalonados e distribuídos entre vários nós de forma transparente.

No Kubernetes, a recuperação de falhas de nós é tratada de maneira semelhante. Quando um nó falha, o Kubernetes detecta a falha e inicia a reprogramação dos pods que estavam em execução no nó falho para outros nós disponíveis no cluster. O Kubernetes também oferece recursos avançados, como tolerância a falhas e replicação de pods, para garantir que as aplicações permaneçam disponíveis mesmo em caso de falha de nós.

5.6.2 Falha em containers no cluster

No Docker Swarm, a recuperação de falhas de contêineres é tratada automaticamente pelo próprio Docker Engine. Se um contêiner falhar devido a um erro de aplicação ou falha de sistema, o Docker Swarm reinicia automaticamente o contêiner em outro nó do cluster. Isso garante que a aplicação continue funcionando normalmente, mesmo em caso de falha de um contêiner individual.

No Kubernetes, a recuperação de falhas de pods é gerenciada pelo próprio Kubernetes. Se um pod falhar devido a um erro de aplicação, falha de sistema ou qualquer outro motivo, o Kubernetes inicia a criação de um novo pod para substituir o pod falho. Além disso, o Kubernetes oferece recursos de sondagem para detectar e lidar automaticamente com falhas de aplicação e garantir a disponibilidade contínua dos pods.

5.7 Upgrade de versão

No Docker Swarm, o processo de upgrade de versão de uma aplicação é relativamente simples e direto. O processo envolve a substituição dos contêineres em execução pela nova versão da imagem do contêiner. Isso é feito de forma que a aplicação permaneça disponível durante o processo de atualização, minimizando o tempo de inatividade.

Uma das vantagens do Docker Swarm é a facilidade de uso e simplicidade de sua abordagem para o upgrade de versão. O processo é geralmente rápido e transparente, garantindo que a aplicação permaneça disponível durante todo o processo de atualização. No entanto, em termos de flexibilidade, o Docker Swarm pode ser um pouco limitado em comparação com o Kubernetes.

No Kubernetes, o upgrade de versão de uma aplicação é um processo mais flexível e granular. O Kubernetes permite atualizar não apenas a imagem do contêiner, mas também outras partes da configuração da aplicação, como variáveis de ambiente, volumes e portas expostas. Isso é feito usando recursos como Deployment e StatefulSet, que gerenciam a implantação e atualização de pods de forma automatizada e controlada.

Uma das vantagens do Kubernetes é sua flexibilidade e extensibilidade no processo de upgrade de versão. Você pode implementar estratégias avançadas, como rollbacks automáticos em caso de falha, canários de implementação para testar novas versões com uma porcentagem pequena de tráfego e rollouts graduais para garantir uma transição suave entre versões, oferecendo mais controle e segurança durante o processo de upgrade, especialmente em ambientes de produção críticos.

6 CONCLUSÃO

O Docker Swarm é conhecido por sua simplicidade e facilidade de uso. Foi projetado para ser uma solução simples e direta para orquestração de contêineres, especialmente para equipes que já estão familiarizadas com o ecossistema Docker. Configurar um cluster do Docker Swarm geralmente envolve apenas alguns passos, como iniciar os nós, inicializar o Swarm em um nó e adicionar mais nós conforme necessário. O Docker Swarm possui muitos dos seus recursos de orquestração essenciais integrados e prontos para serem utilizados, sem a necessidade de configurações complexas. A interface de linha de comando do Docker Swarm é intuitiva e fácil de entender, tornando a configuração e o gerenciamento do cluster uma tarefa relativamente simples, mesmo para iniciantes.

O Kubernetes é uma plataforma mais robusta e complexa, projetada para lidar com casos de uso mais avançados e ambientes de produção em escala. A configuração do Kubernetes envolve uma série de conceitos e componentes, como Pods, Services, Deployments, e ConfigMaps. Configurar um cluster do Kubernetes geralmente requer mais etapas e conhecimento técnico, como a configuração de um plano de rede, a escolha de um provedor de armazenamento persistente e a configuração de políticas de segurança.

No entanto, a simplicidade do Docker Swarm também pode ser sua limitação. Embora seja fácil configurar um cluster inicial, ele pode não ser tão flexível ou escalável quanto o Kubernetes em ambientes complexos ou de grande escala. O Docker Swarm carece de recursos avançados de orquestração e gerenciamento, como balanceamento de carga avançado, escalonamento automático e monitoramento detalhado.

Um ponto crítico é observado quando ambos os sistemas chegam ao limite de 100 contêineres ou mais quando foi utilizado instâncias do tipo t2.small. Neste ponto, nenhum dos sistemas foi capaz de prosseguir com a inicialização devido à insuficiência de recursos nas instâncias t2.small. Ambos Docker Swarm e Kubernetes possuem limitações claras de escalabilidade em máquinas com recursos limitados. Isso destaca a importância de uma planejamento cuidadoso da infraestrutura, especialmente ao se escolher o tipo e o tamanho das instâncias em ambientes de produção, para garantir que haja recursos suficientes para suportar a escala desejada.

Além disso, é importante considerar o ecossistema e a comunidade em torno de cada plataforma. O Kubernetes possui uma comunidade extremamente ativa e diversificada, com contribuições de grandes empresas de tecnologia e uma vasta gama de ferramentas e recursos adicionais disponíveis. Isso significa que há uma ampla gama de soluções de escalabilidade, desde métricas e monitoramento até políticas avançadas de escalonamento. Embora o Docker Swarm também tenha uma comunidade sólida, ela pode ser menor em comparação com a do Kubernetes, o que pode limitar a disponibilidade de soluções e recursos adicionais em algumas áreas.

A escolha entre as duas plataformas depende das necessidades específicas de escalabilidade e dos recursos disponíveis para operar e manter o ambiente de contêineres.

Para ambientes de produção altamente escaláveis e complexos, o Kubernetes pode ser a melhor escolha devido à sua flexibilidade e recursos avançados de gerenciamento de cluster. No entanto, para casos de uso mais simples ou ambientes com recursos limitados, o Docker Swarm pode oferecer uma solução mais fácil de configurar e manter.

REFERÊNCIAS

(1) What is a Virtual Machine? | VMware Glossary.
Link: https://www.vmware.com/topics/glossary/content/virtual-machine.html

(2) Use containers to Build, Share and Run your applications.
Link: https://www.docker.com/resources/what-container/

(3) What is AWS? - Cloud Computing with AWS.
Link: https://aws.amazon.com/pt/what-is-aws/

(4) Kubernetes Documentation.
Link: https://kubernetes.io/docs/home/

(5) Swarm mode overview.
Link: https://docs.docker.com/engine/swarm/

(6) Docker Swarm vs Kubernetes: A Practical Comparison.
Link: https://betterstack.com/community/guides/scaling-docker/docker-swarm-kubernetes/

(7) The Power of Kubernetes Auto-Scaling: Scaling Your Applications with Ease.
Link: https://medium.com/@extio/the-power-of-kubernetes-auto-scaling-scaling-your-applications-with-ease-cb232391400c

(8) Docker Swarm: A Tale of Vertical and Horizontal Scaling.
Link: https://medium.com/@thakuravnish2313/docker-swarm-a-tale-of-vertical-and-horizontal-scaling-b8a8e98f323b

(9) Horizontal Pod Autoscaling.
Link: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/