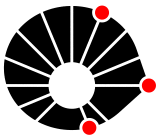

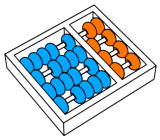

Benchmark de Desempenho de Infraestrutura Automatizada em CaaS

G. P. Pereira *J. P. Soubihe* *L. F. Bittencourt*

Relatório Técnico - IC-PFG-20-21
Projeto Final de Graduação
2020 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Benchmark de Desempenho de Infraestrutura Automatizada em CaaS

Gustavo Pereira João Paulo Soubihe Luiz Fernando Bittencourt*

Resumo

Esse projeto tem como objetivo desenvolver uma infraestrutura que possibilite a realização de um benchmark de utilização de recursos do provedor de serviços de infraestrutura na nuvem da Amazon Web Services, além de avaliar determinados cenários de instâncias com poder computacional variado. Através de uma arquitetura de containers, criamos todas a infraestrutura disponível para a realização de testes utilizando uma aplicação genérica que permite executar operações que provoquem uma carga em recursos de memória de CPU nos mais diversos tipos de instâncias, além de coletar todas as métricas necessárias e disponibilizá-las em uma aplicação de visualização de dados.

1 Introdução

Ao longo dos anos cloud computing vem se tornando o padrão para utilização de infraestruturas de computação nos mais diversos domínios de uso e vem substituindo a prática de possuir recursos computacionais próprios. No modelo IaaS (Infrastructure-as-a-Service), os recursos computacionais, como CPU, memória, disco de armazenamento e infraestrutura de rede, podem ser adquiridos na forma de serviços e possibilitam uma maior volatilidade da infraestrutura, uma vez que esses recursos podem ser incrementados ou dispensados de forma fácil através de APIs (Application Programming Interface) e provisionados como virtual machines (VMs). Essas VMs podem ser disponibilizadas com os mais diversos níveis de configuração, desde máquinas mais simples com pouca memória até máquinas muito potentes [1]. Como a gama de configurações para as máquinas virtuais é muito grande, selecionar uma configuração apropriada para uma aplicação acaba se tornando uma tarefa muito difícil. Para esse trabalho, propomos uma infraestrutura de clusters e containers que possibilita analisar o comportamento de diversos tipos de instâncias de maneira flexível, através da execução de testes de simulação que exigem o uso de recursos como memória e CPU. Essas métricas serão coletadas e expostas em um dashboard contido em nossa infraestrutura para análise do funcionamento e posterior análise do custo proporcionado pelo provedor CaaS (Container-as-a-Service). Outra motivação foi a de avaliar a decisão entre uma estratégia de escalabilidade vertical, adicionando recursos às máquinas existentes, e escalabilidade horizontal, a partir da réplica de instâncias de mesmo valor computacional

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP

no ambiente. Segundo a Gartner [8], em matéria de Julho de 2020 , o revenue de serviços baseados em cloud computing irá crescer em torno de 6.3%

2019	2020	2021	2022
45,212	43,438	46,287	49,509
37,512	43,498	57,337	72,022
102,064	43,948	57,337	72,022
102,064	104,672	120,990	140,629
12,836	14,663	16,089	18,387

Tabela 1: crescimento dos serviços cloud

Tecnologias novas como o 5G, e a nova rotina de trabalho usual, por efeito da pandemia, só aceleraram esse cenário. Por ser fruto de uma propriedade particular, muitos dos seus recursos são declarados sem que haja uma mostra real do seu potencial. Para esse efeito, os cloud benchmarks são instrumentos poderosos nesse tipo de situação, em que precisa-se obter uma informação mais precisa quanto à performance desses recursos.

2 Trabalhos Relacionados

Com o crescimento e a relevância que a área de cloud computing tomou, existem muitos trabalhos na academia sobre o assunto. Alguns deles buscam comparar os serviços equivalentes em cada cloud provider com base em análises a partir de benchmarks, que podem ser desenvolvidos especificamente para cada avaliação, com foco em determinado parâmetro, ou que já fazem parte de alguma aplicação terceira, como por exemplo o CloudWorkbench(CWB), ou outra ferramenta a disposição no mercado, [1] e [12]. Nosso foco aqui foi o de propor uma reflexão a respeito do uso de recursos em diferentes distribuições físicas, como o local onde cada máquina estará, a quantidade de recursos, e também prover uma aplicação capaz de gerar a estrutura em questão e permitir a sua avaliação, rápida e automatizada. Para isso, seguimos uma linha de arquitetura muito parecida com alguns outros artigos da bibliografia, em [**?**], [10], [11], existe todo um fluxo para que a realização e análise dos testes possa ser feita. Normalmente esse fluxo se dá pela criação da infraestrutura →Execução de testes →Análise dos resultados. A complexidade pode depender do tipo de serviço, seja ele um SaaS ou IaaS, ou até um PaaS. No nosso caso estaremos analisando um serviço derivado do SaaS, o CaaS. Containers têm sido cada vez mais utilizados no mercado e na academia, disseminando estudos ao seu respeito. Nosso foco aqui, não se dará propriamente nessa tecnologia, mas é inevitável não associá-la e/ou esbarrar em algumas questões referentes à mesma. De certa forma, pode-se considerar nosso projeto como uma ferramenta para service selection, através de benchmarks e a análise dos seus resultados. Com a aplicação da mesma, o cliente poderá visualizar com mais dados qual a melhor disposição das suas instâncias e equilibrá-la com o fator tamanho e/ou recurso.

3 Conceitos

3.1 CaaS

Containers como serviço (CaaS) é um modelo de computação em nuvem que permite aos usuários implantar e gerenciar aplicações por meio da abstração baseada em containers, seja em data centers on-premise ou na nuvem. O provedor oferece o framework ou a plataforma de orquestração na qual os containers serão implantados e gerenciados. É por meio dessa orquestração que as principais funções de TI serão automatizadas. Uma solução de CaaS é útil, principalmente, para que os desenvolvedores possam criar aplicações em containers mais seguras e escaláveis. Os usuários podem adquirir apenas os recursos que querem (funcionalidades de programação, balanceamento de carga etc.) para economizar e aumentar a eficiência. [2]

3.2 Containers

Os containers oferecem um mecanismo de empacotamento lógico em que os aplicativos podem ser abstraídos pelo ambiente em que são efetivamente executados. Esse desacoplamento permite que aplicativos baseados em contêiner sejam implantados de maneira fácil e consistente, independentemente de o ambiente de destino ser um data center particular, a nuvem pública ou até mesmo o laptop pessoal de um desenvolvedor. A conteinerização oferece uma separação clara de preocupações, porque os desenvolvedores focam na lógica e nas dependências do aplicativo, enquanto as equipes de operações de TI podem focar na implantação e no gerenciamento sem se preocupar com detalhes do aplicativo, como versões de software específicas e configurações específicas do app. Assim como as máquinas virtuais, os containers permitem empacotar o aplicativo com bibliotecas e outras dependências, oferecendo ambientes isolados para executar os serviços de software. Como você verá abaixo, são apenas essas as semelhanças. Isso porque os contêineres fornecem uma unidade muito mais leve para desenvolvedores e equipes de operações de TI trabalharem, oferecendo uma infinidade de vantagens. [3]

3.3 Terraform

Terraform é uma ferramenta de código aberto comumente utilizada para construir, alterar e versionar uma infraestrutura de forma segura e eficiente, através de uma linguagem declarativa. A ferramenta é escrita em GoLang e teve seu primeiro release em 2014 e atualmente suporta uma grande variedade de nuvens públicas, como Google Cloud, AWS, Azure, Digital Ocean e também nuvens privadas, como por exemplo o OpenStack. Com Terraform é possível gerenciar os componentes chamados de low-level, como por exemplo instâncias (máquinas virtuais), storage e redes, mas também serviços, como DNS e bancos de dados. [4] No nosso projeto, nos aproveitamos de um de seus maiores trunfos: a automação do provisionamento de toda uma infraestrutura em cloud

3.4 Grafana

O Grafana é uma plataforma para visualizar e analisar métricas por meio de gráficos. Ele tem suporte para diversos tipos de bancos de dados — tanto gratuitos quanto pagos —, e pode ser instalado em qualquer sistema operacional. Para facilitar a visualização dos gráficos, é possível criar dashboards dinâmicos que podem ser compartilhados com toda a equipe. Além disso, a ferramenta permite configurar alertas com base nas métricas, que são analisadas de forma contínua para notificar o usuário sempre que preciso, de acordo com as regras definidas por ele. É bastante utilizado por sistemas de monitoramento para gerar gráficos real-time. [5]

4 Infraestrutura

4.1 Ideia Geral

A Figura 1 exemplifica o conceito da interação proposta com os principais componentes e a sua ordem de execução

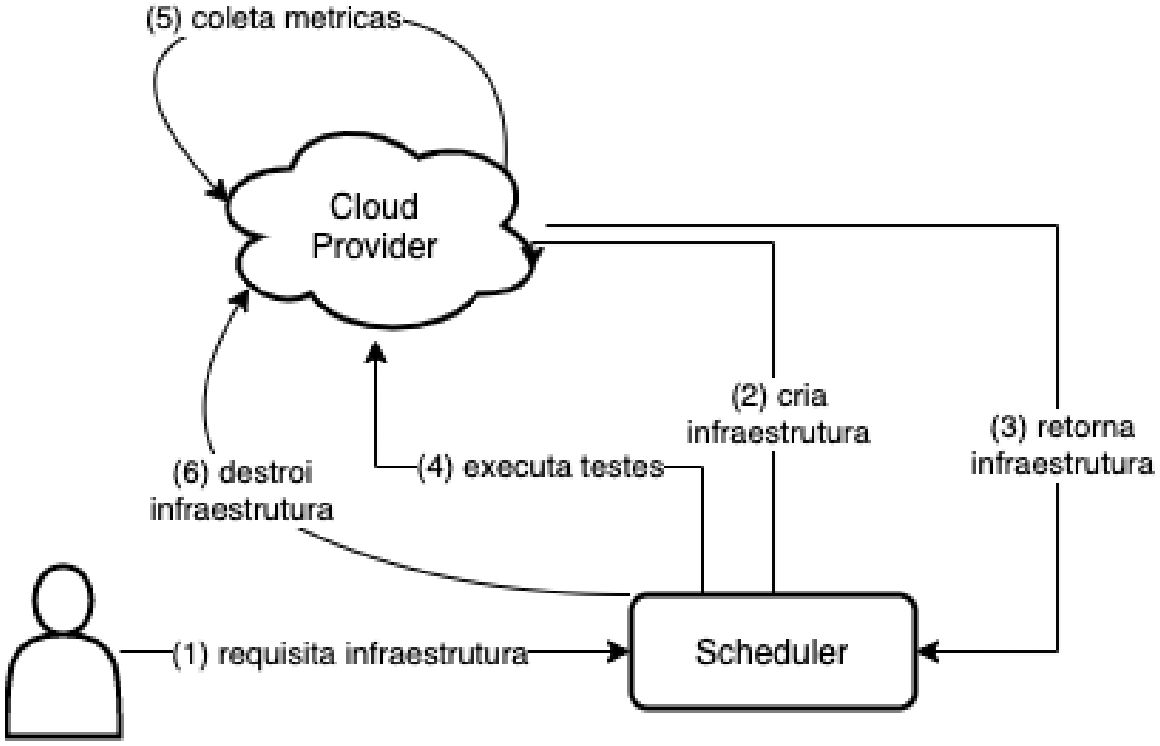

Figura 1: Modelo da arquitetura do projeto

Inicialmente o usuário realiza uma requisição para a aplicação do scheduler, responsável por criar toda a infraestrutura necessária para a realização do benchmark, determinando um conjunto de testes a ser avaliado. Após aplicar a infraestrutura conforme as regras estabelecidas nos arquivos de configuração do Terraform, o scheduler faz consecutivas verificações de saúde da aplicação genérica utilizada a fim de determinar o momento em que

poderá iniciar o conjunto de testes definidos pelo usuário. Após o scheduler determinar que a infraestrutura está saudável, ele começa a realizar uma sequência de requisições que irão desencadear comportamentos de consumo de recursos na aplicação genérica, como simular o uso de CPU, uso de memória, tempo de leitura e alocação de um arquivo estático na memória e simular um teste de carga apenas segurando a execução de uma thread. Durante esses testes, a aplicação irá coletar diversas métricas que serão disponibilizadas em um dashboard do Grafana. É através dessas métricas coletadas que iremos analisar como se comportam diferentes tipos de instâncias virtuais de acordo com um mesmo conjunto e parâmetros de teste. Ao fim dos testes e da coleta das métricas, o scheduler se encarrega de destruir a infraestrutura criada.

4.2 CaaS

Na Figura 2 está esquematizado a infraestrutura completa de clusters e containers utilizada na Amazon Web Services para a realização do benchmark.

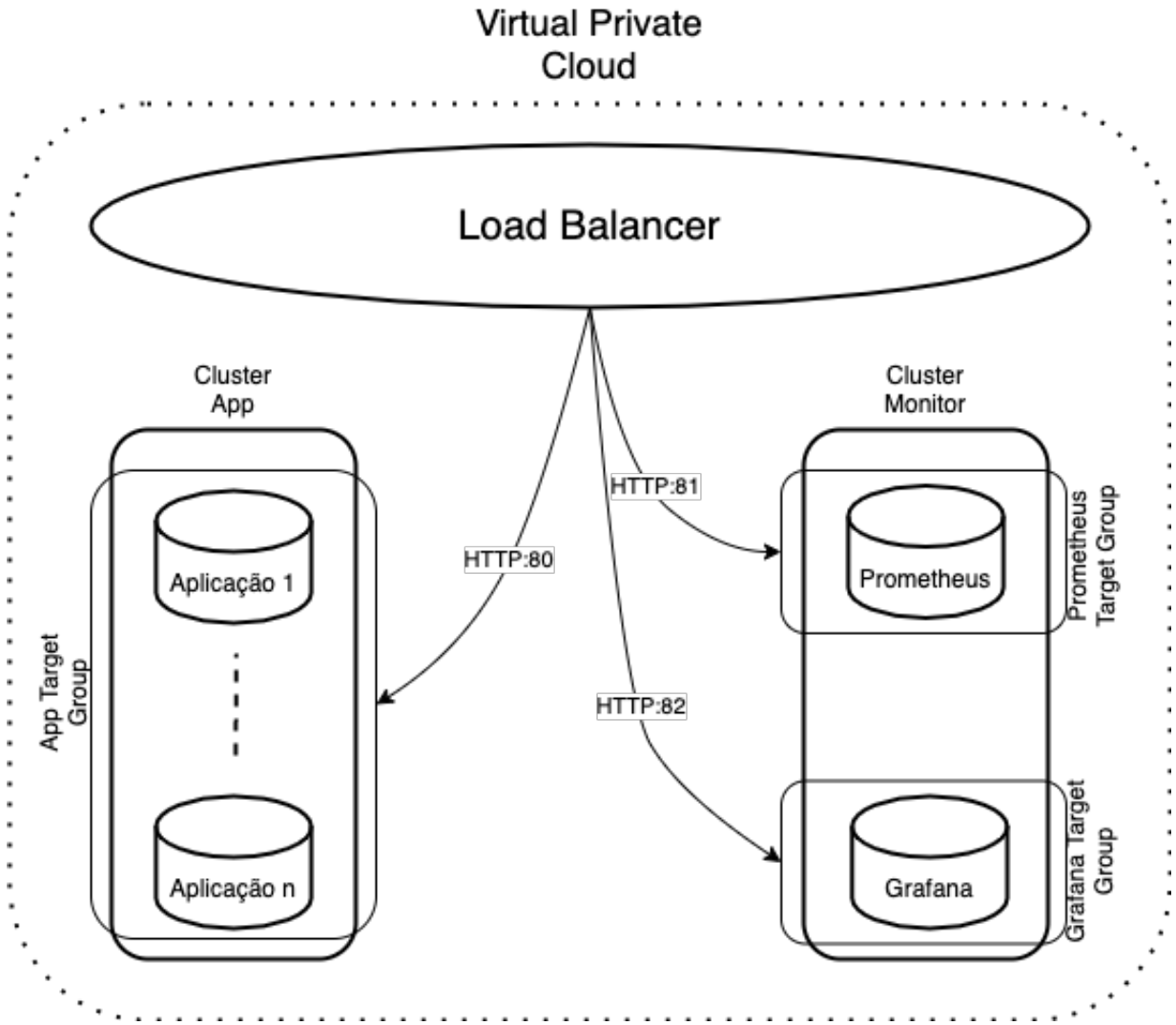

Figura 2: Modelo arquitetura da solução no AWS

Uma abstração de rede denominada Virtual Private Cloud agrega todos os recursos em uma mesma rede interna, o que possibilita a comunicação entre os componentes. Foram criados 2 clusters: um cluster que contém 'n' instâncias que podem variar com relação ao cenário proposto, e um cluster que incorpora os componentes necessários para fazer o scraping (coleta) das métricas geradas pela aplicação e visualização dessas métricas em um dashboard. O Load Balancer é o componente responsável por identificar o estado das instâncias da aplicação e rotear as requisições para determinada instância de acordo com a disponibilidade de cada uma delas dentro do seu grupo de destino. Requisições HTTP feitas na porta 80 são direcionadas para a aplicação, requisições feitas na porta 81 são direcionadas para o Prometheus e requisições feitas para a porta 82 são direcionadas para o Grafana. No diagrama da Figura 3 conseguimos visualiaxar o fluxo de dados que ocorre nessa infraestrutura remota.

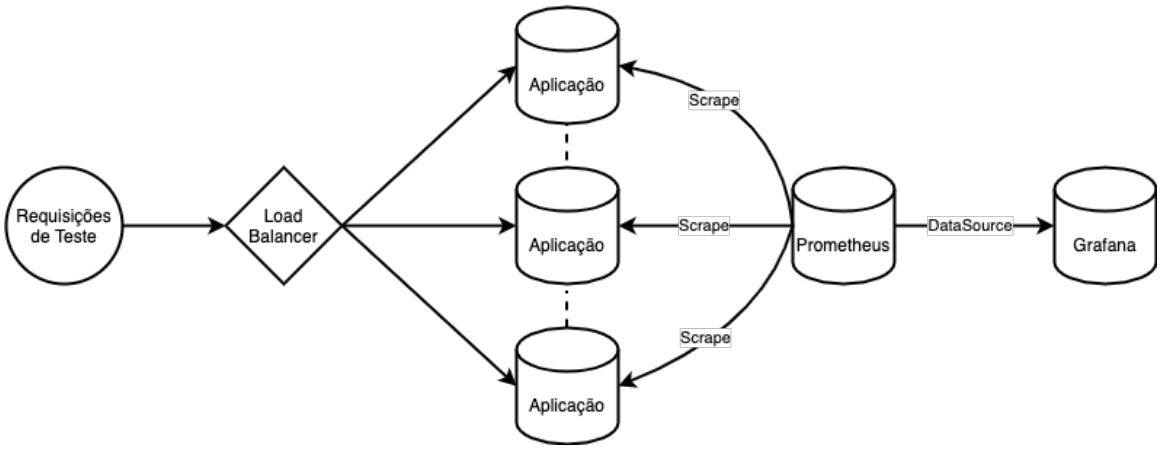

Figura 3: Fluxo de dados

O serviço do Prometheus fica responsável por descobrir quais instâncias da aplicação estão disponíveis e estão aptas a receber requisições. Então ele, de tempos em tempos, realiza o scraping das métricas dessas aplicações e armazena dentro do seu container. Assim, o Prometheus é disponibilizado como o data source dessas métricas que vão ser exibidas no Grafana. O Grafana por sua vez consulta essas métricas a cada 30 segundos e atualiza o dashboard automaticamente. Elaboramos um dashboard no Grafana para visualizar e analisar as métricas definidas como as mais relevantes para o estudo do benchmark. O código desse dashboard está disponível no repositório do projeto no GitHub [9].

5 Metodologia

No projeto proposto iremos analisar o comportamento do serviço de IaaS da Amazon Web Services, onde iremos disponibilizar uma infraestrutura

5.1 Uptime

Para a medição do tempo de Uptime do ambiente em cada cenário, realizamos duas contagens diferentes: tempo de criação da infraestrutura e tempo de health da aplicação (tempo com que a aplicação começa a rodar no container e está apta a receber requests). Para o tempo de criação da infraestrutura, iniciamos o contador a partir do início da aplicação do script do Terraform até o momento em que a aplicação retorna um status HTTP 200 para o endpoint de health check /actuator/health. Para o tempo de health, iniciamos o contador a partir do fim da execução do script do Terraform até o momento em que a aplicação retorna o status HTTP 200 no endpoint de health check /actuator/health

5.2 CPU

No cenário onde será avaliado o uso de CPU das instâncias, definimos um caso de uso onde a aplicação irá executar o carregamento de um arquivo contendo um vetor desordenado com 500.000 elementos inteiros de 0 a 500.000. Após o carregamento desse arquivo em

memória, a aplicação executa uma ordenação desse vetor utilizando o método sort() do pacote java.util.List. A ordenação original do vetor e seus elementos é mantida fixa para que o custo da operação seja o mesmo para todas as instâncias avaliadas. Na execução dos testes, o scheduler separa o conjunto de requisições para a aplicação em 5 pacotes de 100 requisições cada, onde cada pacote é executado com um intervalo de 3 minutos, totalizando 500 requisições, a fim de medir o comportamento e utilização da CPU nessa janela de tempo. Pelo fato de a ordenação de um vetor de muitos elementos se tratar de uma operação custosa, podemos comparar melhor a diferença entre diversos recursos com unidades menores de CPU versus um único recurso computacional com mais unidades de CPU.

5.3 Memória

Para a avaliação da performance da memória nas diferentes instâncias, foi elaborada uma execução onde é carregado um arquivo texto de 1Mb em memória, em seguida é copiado todos os bytes desse arquivo para um segundo vetor de bytes. Assim medimos o tempo de execução desse código e expomos em uma métrica denominada memory_read_operation_seconds. Essa métrica será analisada no Grafana como uma média entre todas as execuções dessa mesma operação. O scheduler então separou 6 conjuntos de requisições com 50 requisições em cada conjunto, onde cada conjunto é executado em um intervalo de 3 minutos, totalizando 300 requisições. Optamos por medir esse tempo de execução pois achamos que valeria a pena mensurar a performance através do cenário onde fossem realizadas operações de leitura e escrita em memória, pois dessa maneira teríamos uma análise mais assertiva a respeito da forma que cada Hardware de cada instância se comportaria com um número fixo de bytes.

5.4 Preço

Quando falamos de escalabilidade e uso prático de uma solução baseada em computação em nuvem há um fator muito relevante nesse planejamento: o preço. Uma das grandes vantagens do cloud computing é a cobrança on demand. Dessa maneira empresas que usufruem do serviço pagarão somente pelos recursos que escolherem usar, conforme a demanda desses clientes. É inegável, então, que o preço terá grande impacto na escolha das instâncias e na disposição das mesmas. A Tabela 2 mostra os valores extraídos de [13] dos recursos utilizados, sob demanda:

5.5 Cenários

Cada cenário de teste foi elaborado usando um tipo de instância diferente disponível no serviço de EC2 da Amazon Web Services [6]. A ideia é que para todos os cenários, o cluster da aplicação possua o mesmo poder computacional, totalizando sempre o mesmo número de unidades de CPU e tamanho de memória. Como mais um atributo para nossa análise, também elaboramos um cenário em específico para avaliar o desempenho dessa mesma quantidade de instâncias distribuída de uma maneira isolada entre as availability zones. A forma “spread” vai contra o nosso modelo default de organizar todas as instâncias

Tipo de instância	Preço
m4.large	0.10 USD por hora
m4.2xlarge	0.40 USD por hora
r5.large	0.252 USD por hora
r5.2xlarge	0.504 USD por hora
c5.large	0.085 USD por hora
c5.2xlarge	0.34 USD por hora

Tabela 2: Preços de instância

necessariamente em um mesmo espaço, compartilhando recursos entre si. Com base na teoria de Placement Groups [14], segundo o AWS, desenvolvemos mais uma alternativa à nossa solução.
As instâncias de cada cenário, assim como os seus recursos computacionais estão descritos na Tabela 3.

Tipo de instância	Preço	N	CPU	Memória	Cluster CPU	Cluster Memória
1	m4.large	4	2 vCPU	8 GiB	8 vCPU	32 GiB
2	m4.2xlarge	1	8 vCPU	32 GiB	8 vCPU	32 GiB
3	r5.large	4	2 vCPU	16 GiB	8 vCPU	32 GiB
4	r5.2xlarge	1	8 vCPU	64 GiB	8 vCPU	32 GiB
5	c5.large	4	2 vCPU	4 GiB	8 vCPU	16 GiB
6	c5.2xlarge	1	8 vCPU	16 GiB	8 vCPU	16 GiB
7	m4.large (spread)	4	2 vCPU	8 GiB	8 vCPU	32 GiB

Tabela 3: Instâncias utilizadas

6 Avaliação

- Uptime

  Com base na coleta dos tempos de Uptime em cada cenário, mostrada acima na Tabela 4, pudemos constatar que o tempo necessário para o ambiente estar saudável e pronto para receber novas requisições foi, em média, 12,31% menor em clusters com 4 instâncias do que em clusters com uma única instância com capacidade de processamento maior.

- Cenários 1 e 2

  - CPU

  Para a análise de CPU dos cenários 1 e 2, que compartilham o grupo de instâncias M4, um grupo de instâncias de uso geral, vamos utilizar os gráficos de load de CPU

Cenário	Tempo de criação da infraestrutura	Tempo de health da aplicação
1	319.933 s	24.101 s
2	323.351 s	27.837 s
3	309.189 s	24.334 s
4	353.297 s	25.412
5	321.267 s	23.426 s
6	380.414	29.021 s
7	313.951 s	25.074 s

Tabela 4: Uptime do ambiente

(Figuras 4 e 5) - para medir a quantidade de processos enfileirados no processador - e gráfico de uso de CPU em porcentagem (Figuras 6 e 7).

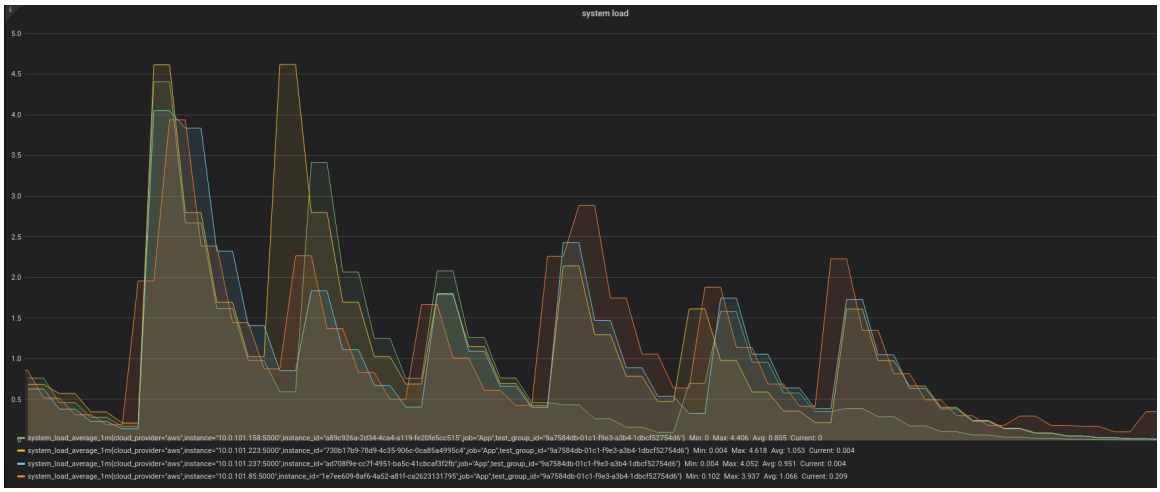

Figura 4: CPU Load do cenário 1

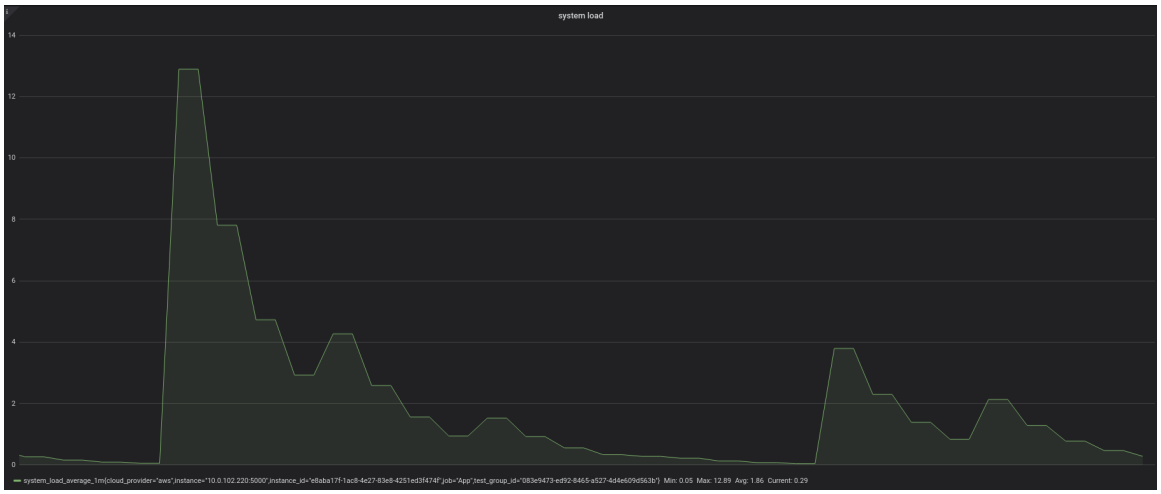

Figura 5: CPU Load do cenário 2

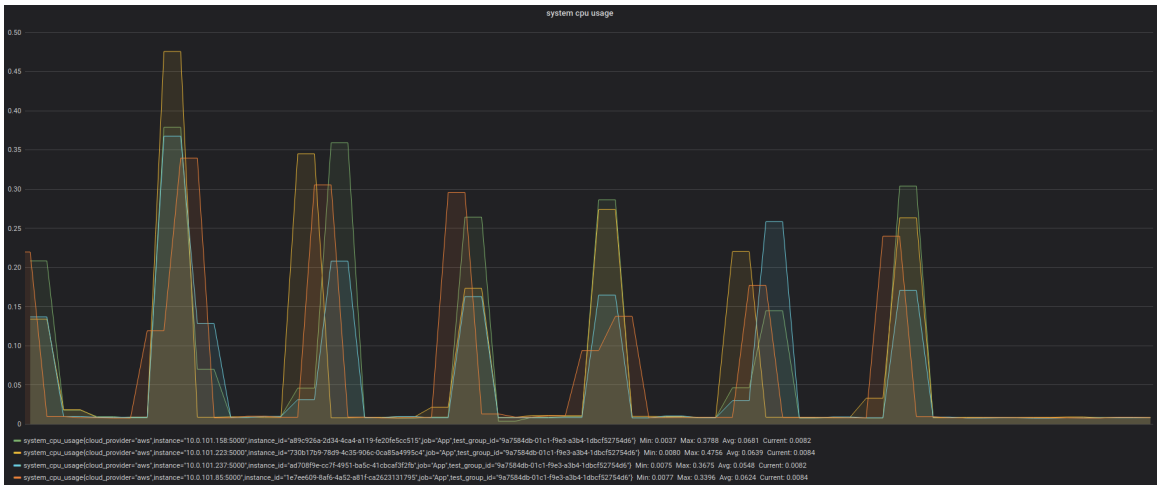

Figura 6: CPU Usage do cenário 1

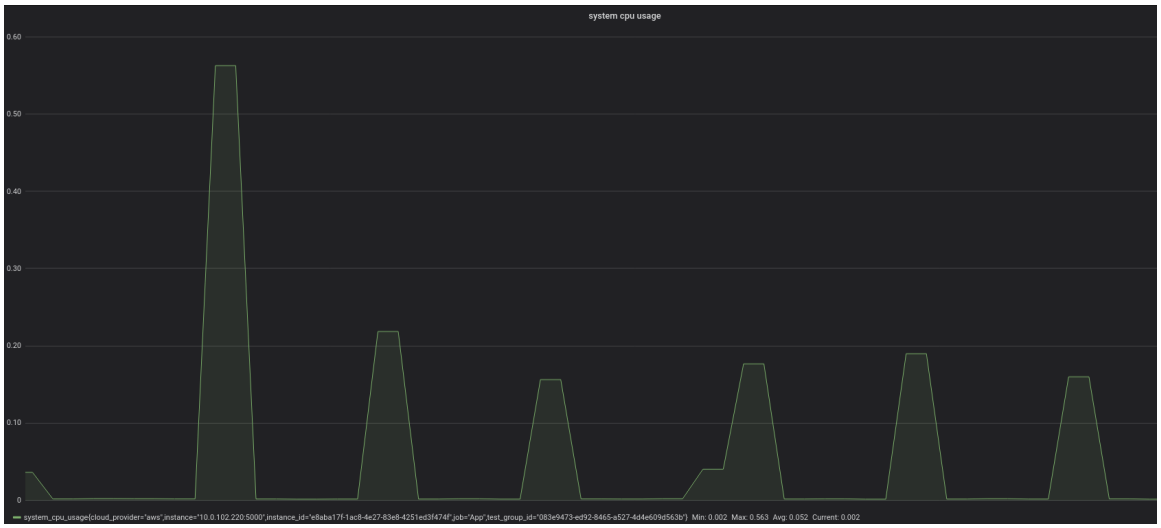

Figura 7: CPU Usage do cenário 2

Observamos que o cluster com 4 instâncias apresentou, durante os bursts dos requests, um uso de CPU acima do valor encontrado para o cluster de 1 única instância, entretanto o load de CPU se manteve abaixo de 1, enquanto o load do cluster de 1 instância esteve com o valor 1,86. Esse resultado mostra fator interessante para a argumentação do escalonamento vertical, para processos que exigem de um alto consumo de recursos de CPU. Depois de um alto consumo para inicializar a aplicação, a performance se estabilizou em níveis menores que os apresentados na solução horizontal. Apesar disso, encontramos uma situação de tradeoff, pois para situações onde há um grande número de requests em um curto espaço de tempo algumas dessas solicitações podem não ser atendidas, por um fator limitante da instância única. Já a solução com múltiplas instâncias trabalha com uma melhor distribuição da mesma, satisfazendo um número maior de requisições. Com relação a esses cenários, um cluster contendo várias instâncias acarreta em uma maior disponibilidade dos recursos, com um aumento muito insignificante do uso de suas CPUs se comparado com o cluster de 1 instância. Principalmente em serviços que receberão um alto volume de requisições em um curto espaço de tempo, situação onde uma arquitetura mais distribuída leva grande vantagem.

- Memória

Para a análise do desempenho de memória dos clusters dos cenários 1 e 2, utilizamos o gráfico da métrica customizada da operação de memória, representado nas Figuras 8 e 9, memory_read_operation_seconds.

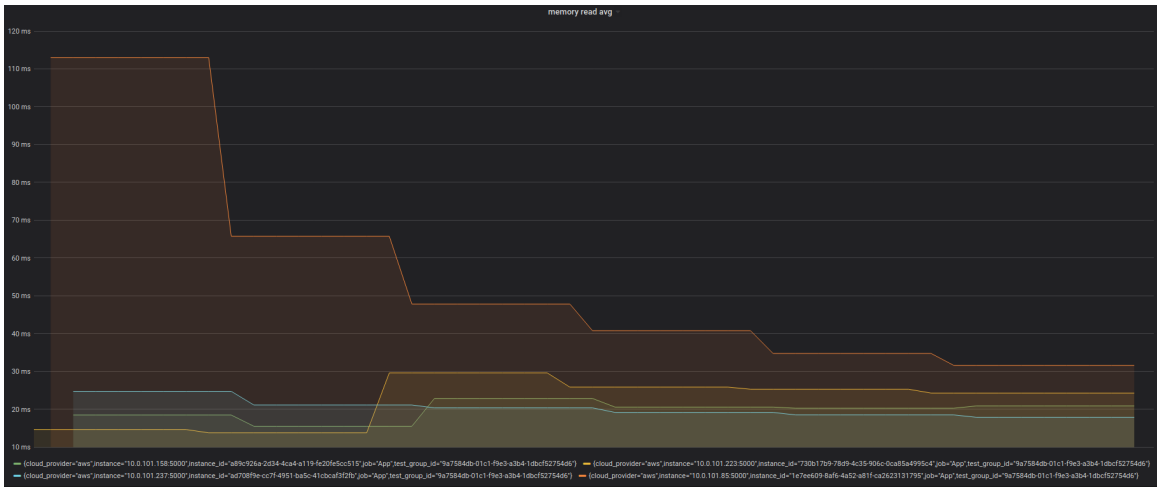

Figura 8: read memory time do cenário 1

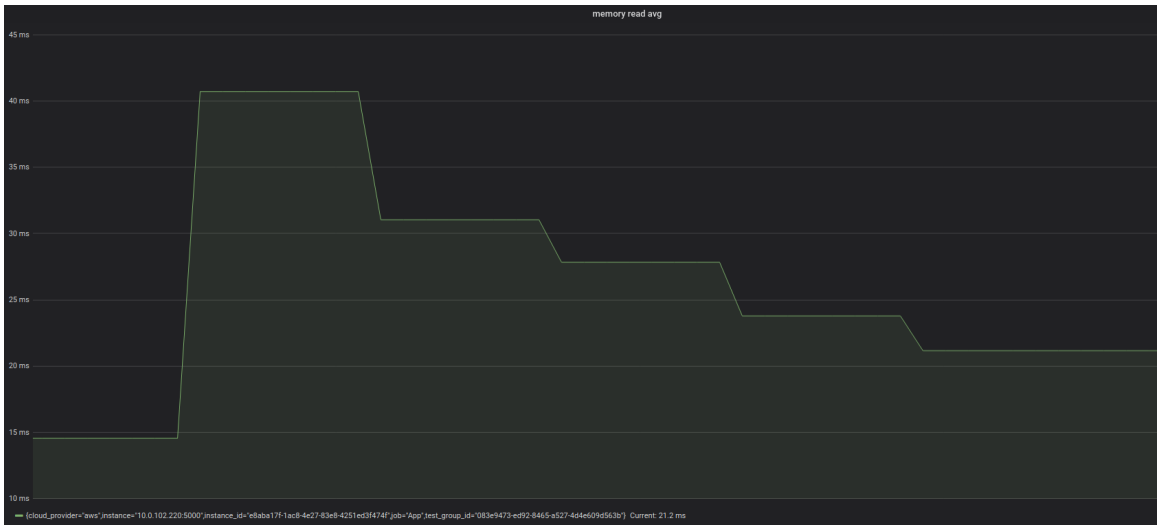

Figura 9: read memory time do cenário 2

Com relação ao desempenho de memória, percebe-se que ambos os clusters do cenário 1 e cenário 2 se comportaram da mesma forma, com valores médios da operação em torno de 20ms Logo, ambos os clusters apresentaram desempenho equivalente com relação ao recurso de memória, logo uma abordagem de escalonamento horizontal ou vertical para aplicações que utilizarem instâncias M4 será apropriado independente do caso de uso.

- Cenários 3 e 4
  Como elemento da nossa avaliação, escolhemos um tipo de instância otimizada para memória. No caso, as instâncias do tipo R5 foram projetadas para funcionar com aplicações que requerem um uso intensivo de memória. São um aperfeiçoamento da versão R4, também otimizada para memória.[6] Os cenários tem a mesma disposição que os demais pares estudados, para que tenhamos uma base de comparação entre eles e agregar ao nosso projeto. Depois de realizados os testes, obtivemos alguns gráficos que contribuiram com a formação

deste trabalho. Começando pelo load de CPU, Figuras 10 e 11, onde abaixo vemos que a quantidade de threads enfileiradas é significativa nas duas estratégias, o que seria normal por se tratarem de instâncias voltadas para operação na memória e não de alto teor computacional. Dito isso, vemos que, mais uma vez, a solução com múltiplas instâncias tem um desempenho um pouco melhor que a estratégia de comparação.

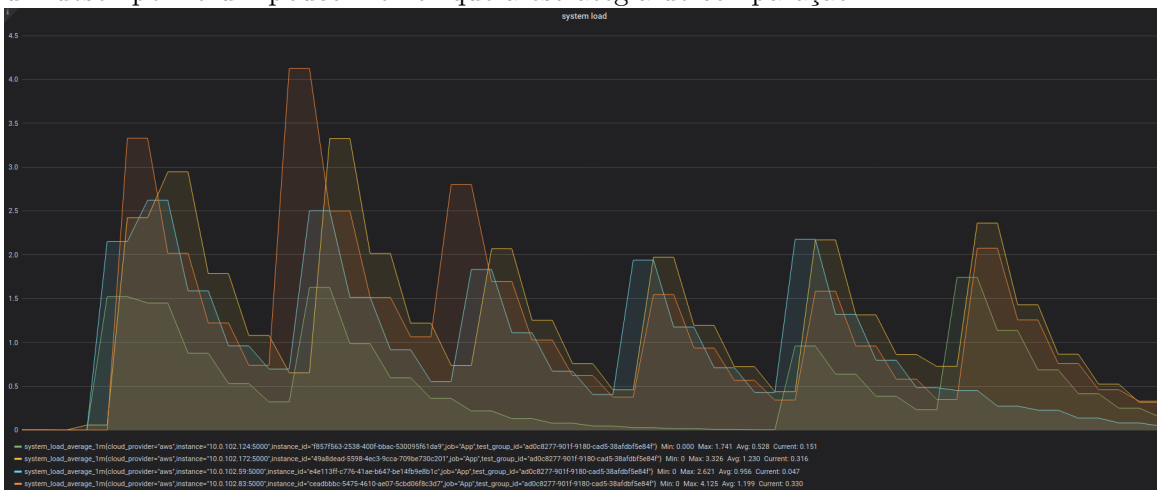

Figura 10: CPU Load do cenário 3

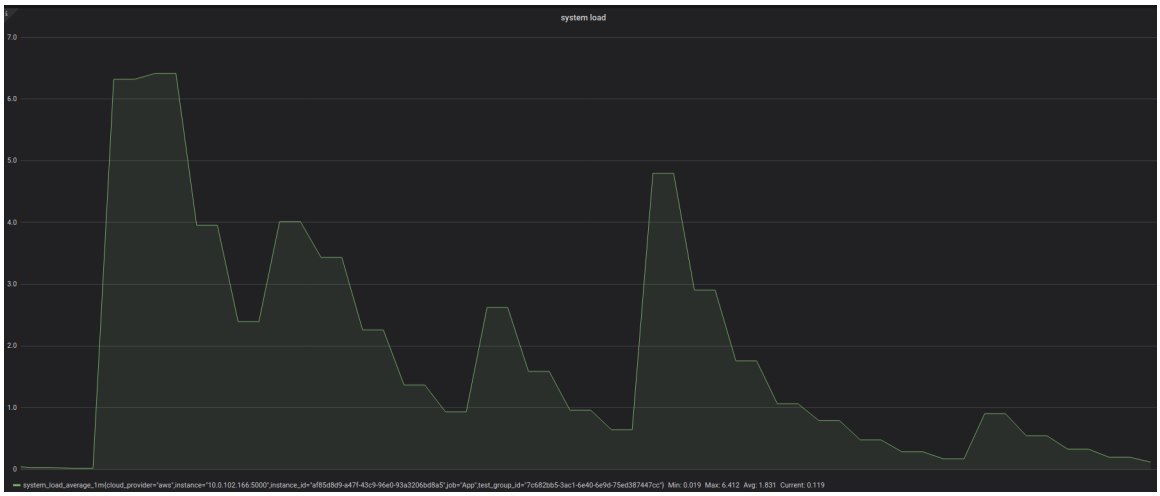

Figura 11: CPU Load do cenário 4

Já em termos de poder de processamento, vemos que depois de um período de estabilização, as medidas giraram por faixas bem próximas uma da outra. Na solução com 4 instâncias o processamento ficou por volta dos 15% da capacidade total, similar a fase mais estável da infraestrutura com uma única máquina atuante.

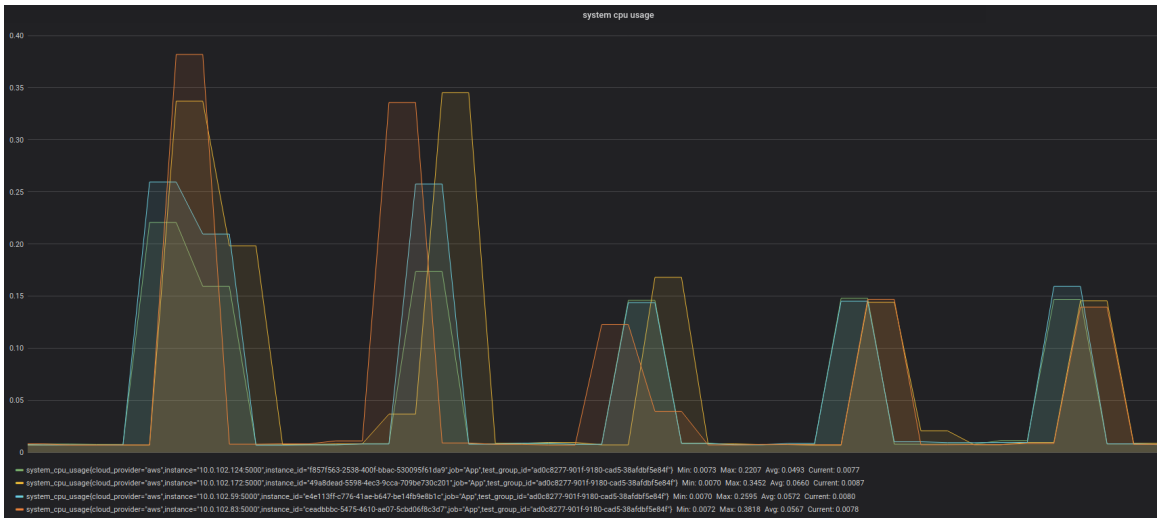

Figura 12: CPU Usage do cenário 3

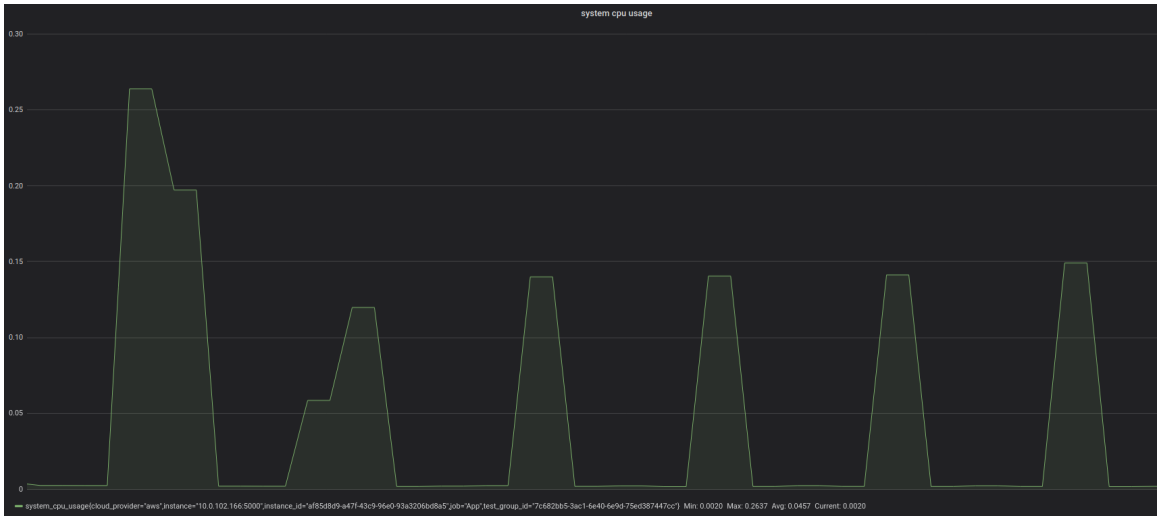

Figura 13: CPU Usage do cenário 4

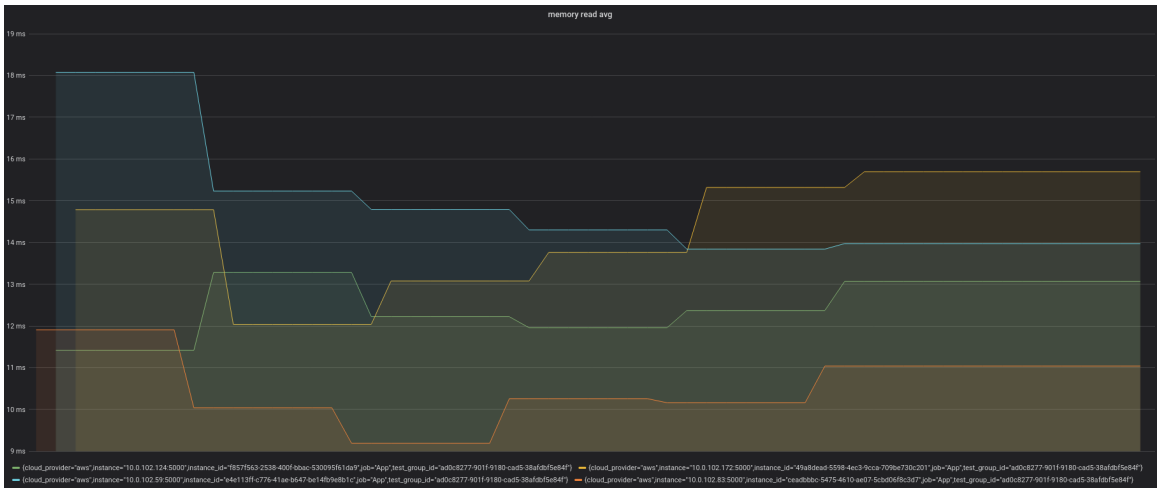

Figura 14: read memory time do cenário 3

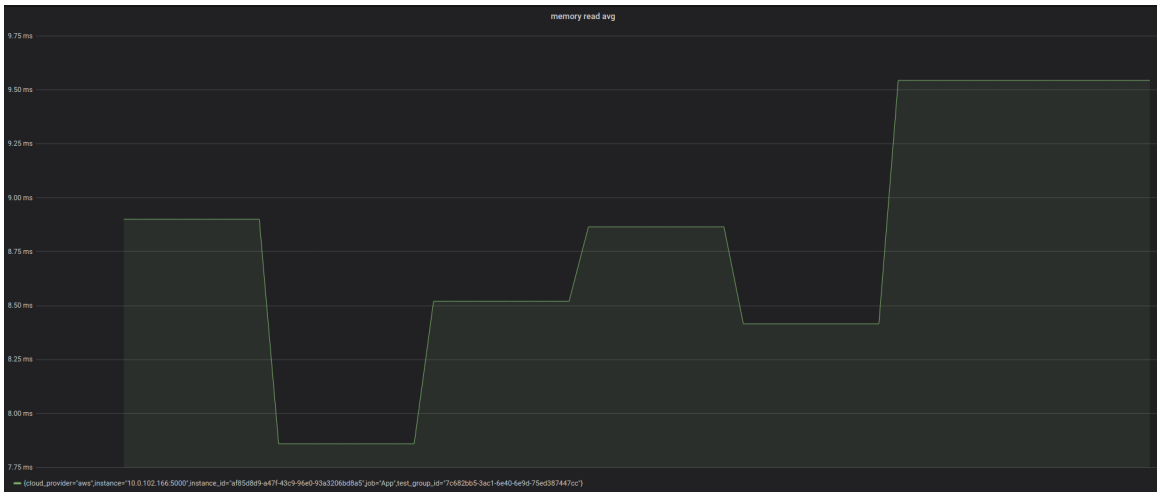

Figura 15: read memory time do cenário 4

Um resultado bem significativo nesse segmento da avaliação, foi a medida de tempo com que o arquivo de amostra foi lido da memória. A maior instância, r5.2xlarge, ao fim do teste, acumulava uma média de 9.50ms. Medida que as instâncias presentes na solução do cenário 3 não conseguiram atingir em nenhum momento. O resultado é uma prova que o modelo mais robusto realmente oferece um maior rendimento dos recursos disponibilizados pela solução.

- Cenários 5 e 6
  - CPU

  Utilizamos novamente as métricas de uso de CPU e load de CPU para avaliar o grupo de instâncias C5, que corresponde a uma família de instâncias otimizadas para computação, ou seja, são adequadas para cargas de trabalho de processamento em lote, transcodificação de mídia, servidores da Web de alto desempenho, computação de alto desempenho (HPC), modelagem científica, servidores de jogos dedicados e mecanismos de servidor de anúncios,

inferência de machine learning e outros aplicativos de uso intensivo de computação.são recomendadas para aplicações.

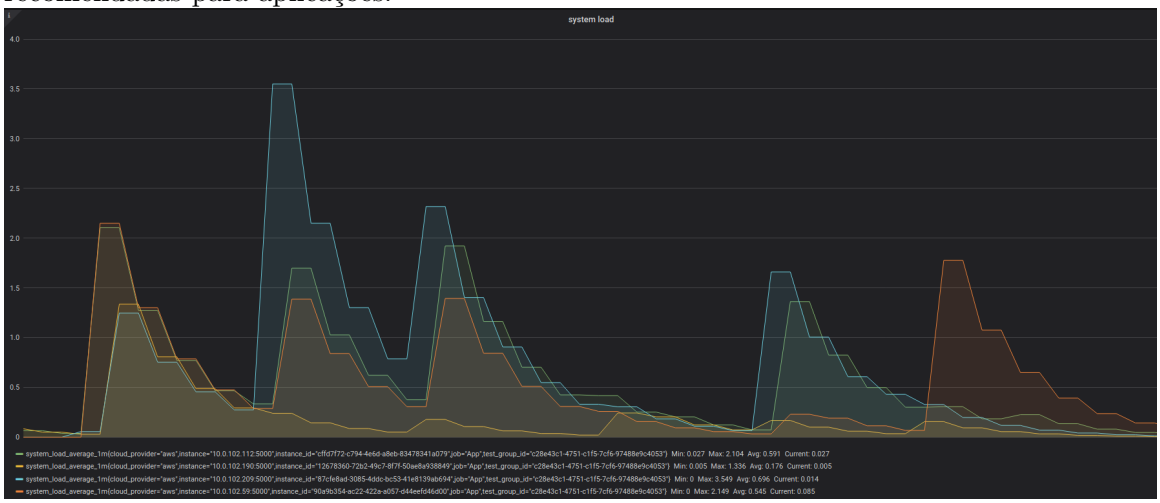

Figura 16: CPU Load do cenário 5

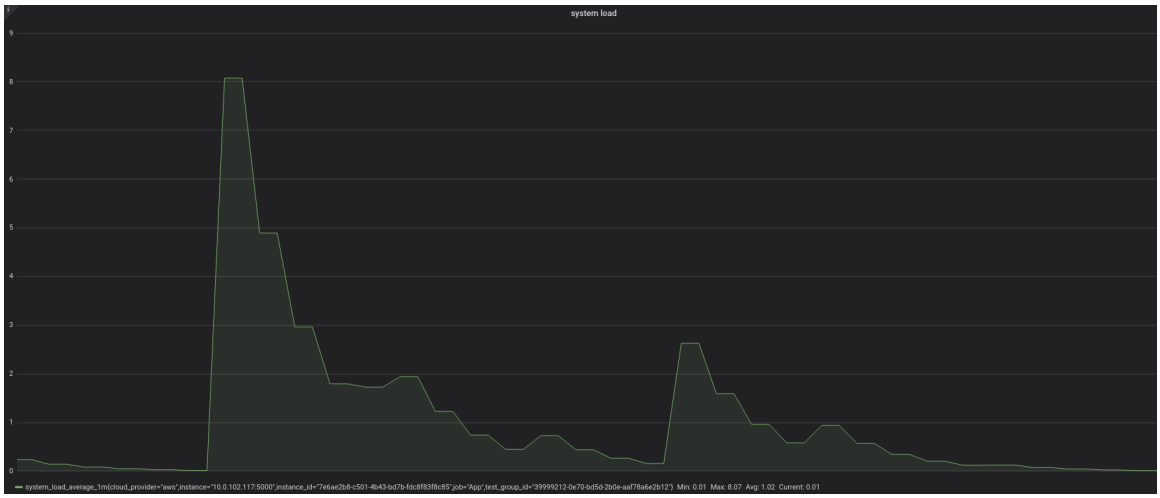

Figura 17: CPU Load do cenário 6

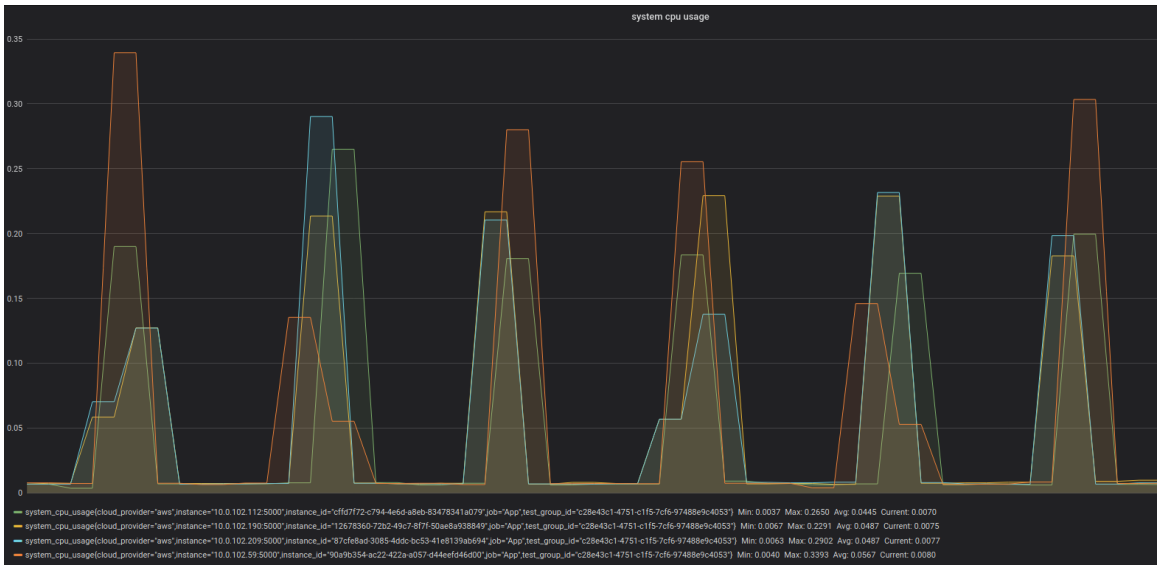

Figura 18: CPU Load do cenário 5

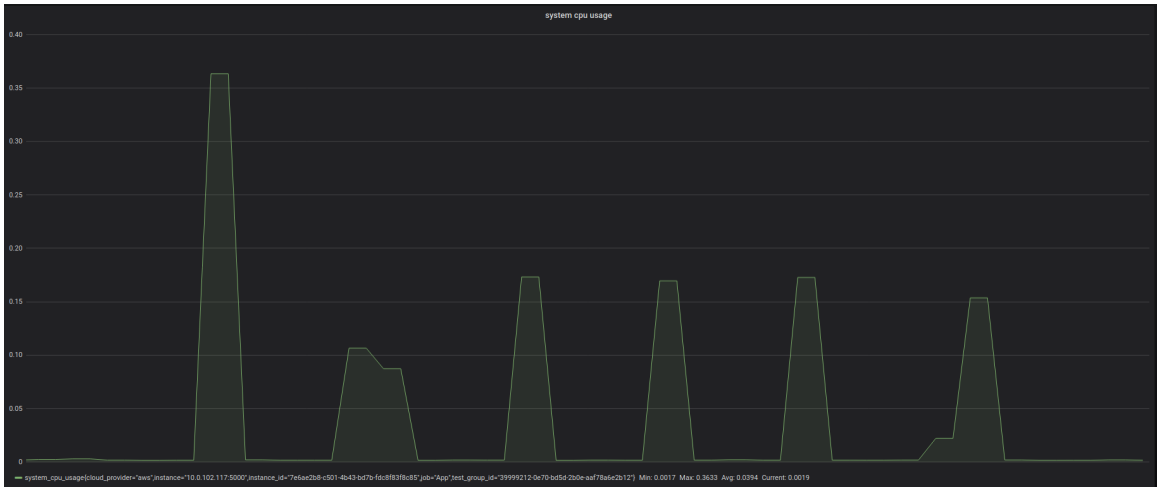

Figura 19: CPU Load do cenário 6

Ao comparar o desempenho do uso médio de CPU do cluster de 4 instâncias, Figura 18, contra o cluster de 1 instância, Figura 19, constatamos que o cluster de 4 instâncias apresentou uma média de uso superior ao do cluster de 1 instância. A especialização no processamento favorece a eficiência no uso desse recurso e acaba compensando, em termos gerais de performance, a maior disponibilidade da solução com múltiplas instâncias.

O load de CPU para o cluster de 4 instâncias ficou abaixo de 1, enquanto o load do cluster de 1 instância ficou um pouco acima de 1, o que nos leva a crer que em ambos os clusters não houve praticamente enfileiramento de processos e a disponibilidade de recursos se manteve alta durante toda a fase de testes. Um dos motivos que pode ter levado a esse acontecimento é o melhor aproveitamento da computação envolvida, processando as requisições mais rapidamente.

- Memória
Utilizamos novamente a métrica customizada de operação de memória para avaliar o desempenho de memória dos cenários 5 e 6.

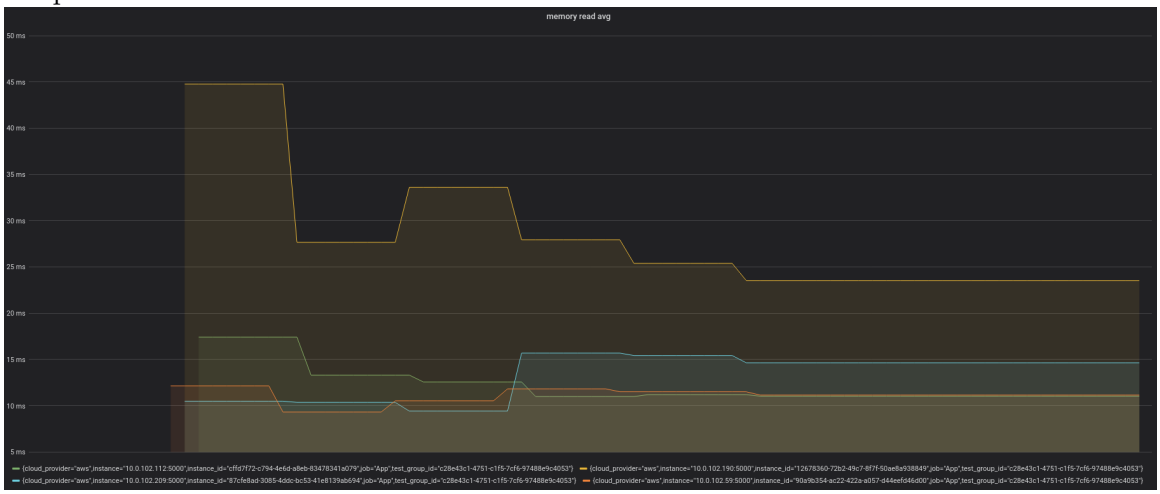

Figura 20: read memory time do cenário 5

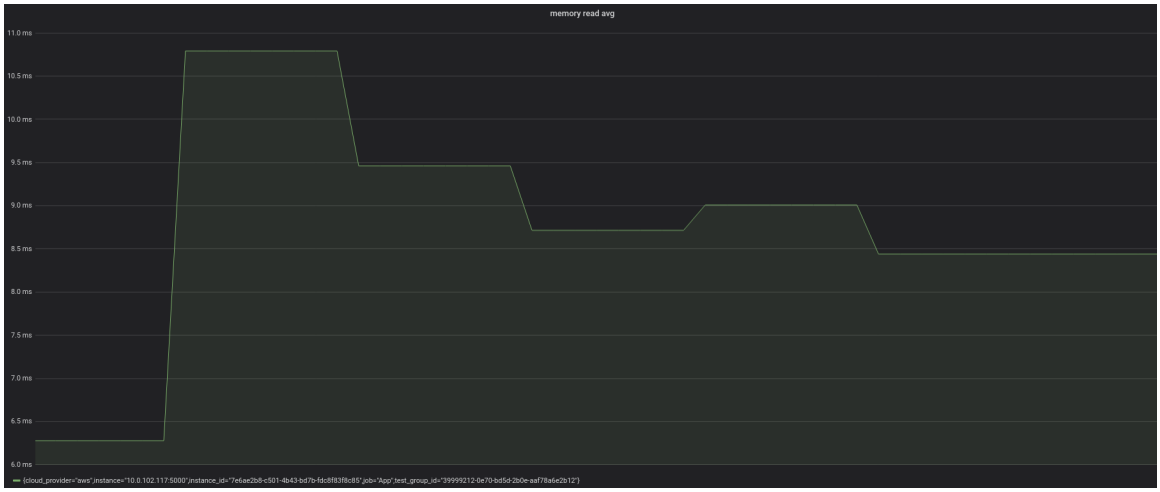

Figura 21: read memory time do cenário 6

Analisando os gráficos de média de tempo da operação de memória, Figura 20 e Figura 21, percebemos que o cluster de 1 instância teve um desempenho superior se comparado ao cluster de 4 instâncias, pois o cluster de 1 instância apresentou uma média em torno de 8.5ms, enquanto o cluster de 4 instâncias apresentou um valor médio em torno de 10ms, com uma das instâncias beirando o valor de 25ms. Logo, para aplicações que irão necessitar de recurso computacional para operações que envolvam muitas leituras e escritas em memória, recomenda-se utilizar um escalonamento vertical.

- Cenário 7
O cenário 7 corresponde apenas a uma validação com relação a um conceito presente no serviço de CaaS da AWS, que é o grupo de posicionamento. Basicamente consiste na

estratégia de posicionamento de instâncias EC2 de forma que todas as suas instâncias estejam distribuídas pelo hardware subjacente para minimizar falhas correlacionadas [13] Para efeitos comparativos da eficácia dessa estratégia, coletamos as mesmas métricas que todos os cenários e iremos compará-lo com relação ao cenário 1, onde o tipo de instâncias e número de instâncias é exatamente o mesmo, entretanto, o cenário 1 está usando a estratégia de grupo de posicionamento denominada cluster enquanto o cenário 7 usa a estratégia denominada spread. A diferença é que a estratégia de cluster agrupa as instâncias dentro de uma zona de disponibilidade enquanto a estratégia de spread posiciona estritamente um pequeno grupo de instâncias por hardware subjacente distinto a fim de reduzir falhas motivadas pelo compartilhamento dos recursos e, assim, ser mais um mecanismo a favor da disponibilidade do serviço.

- CPU

Utilizamos novamente os gráficos de uso de CPU (Figura 24 e Figura 25) e load de CPU (Figura 22 e Figura 23) para verificar diferenças de comportamento relevantes para nossa análise.

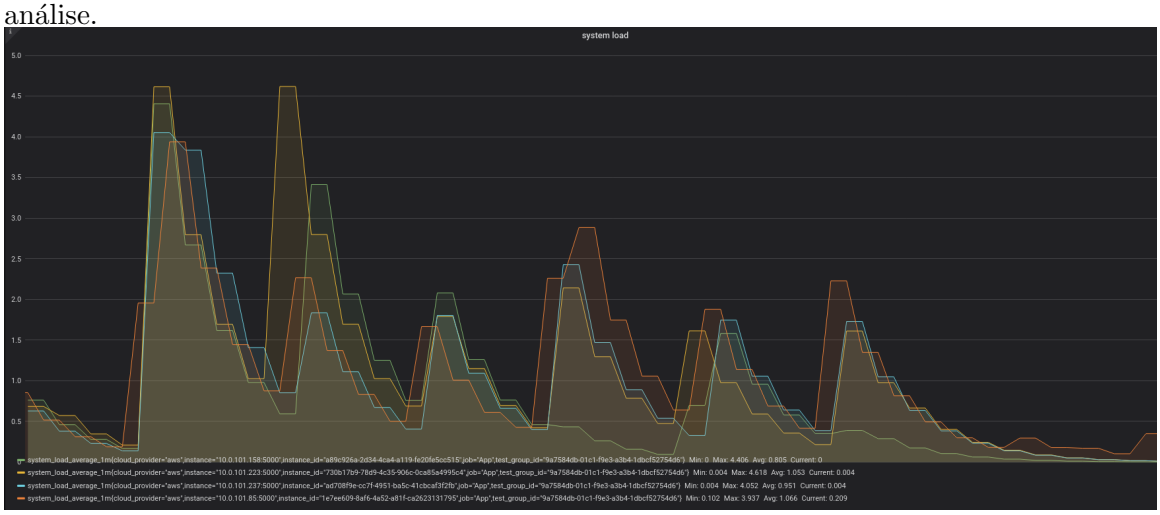

Figura 22: CPU Load do cenário 1

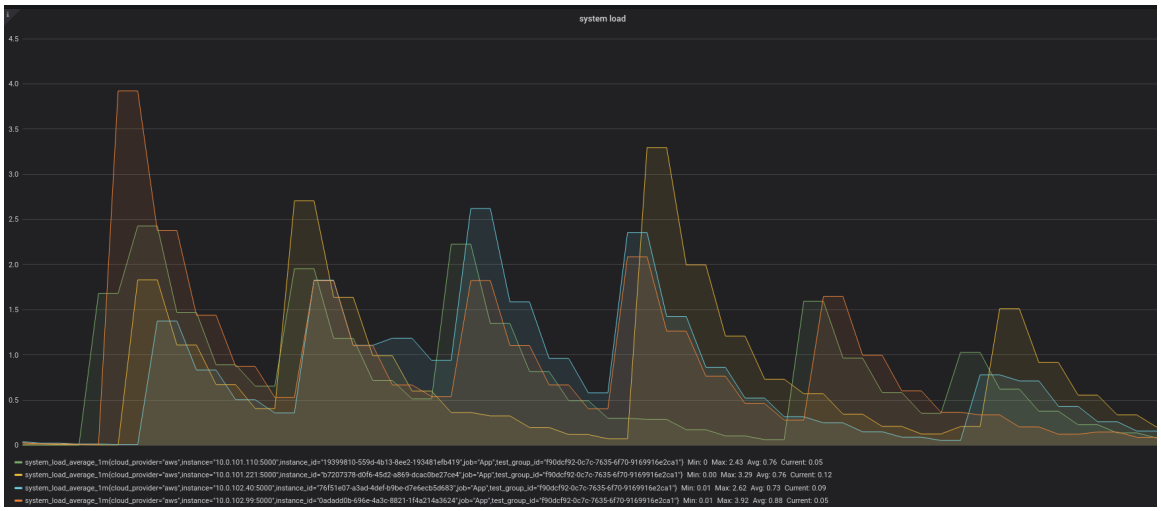

Figura 23: CPU Load do cenário 7

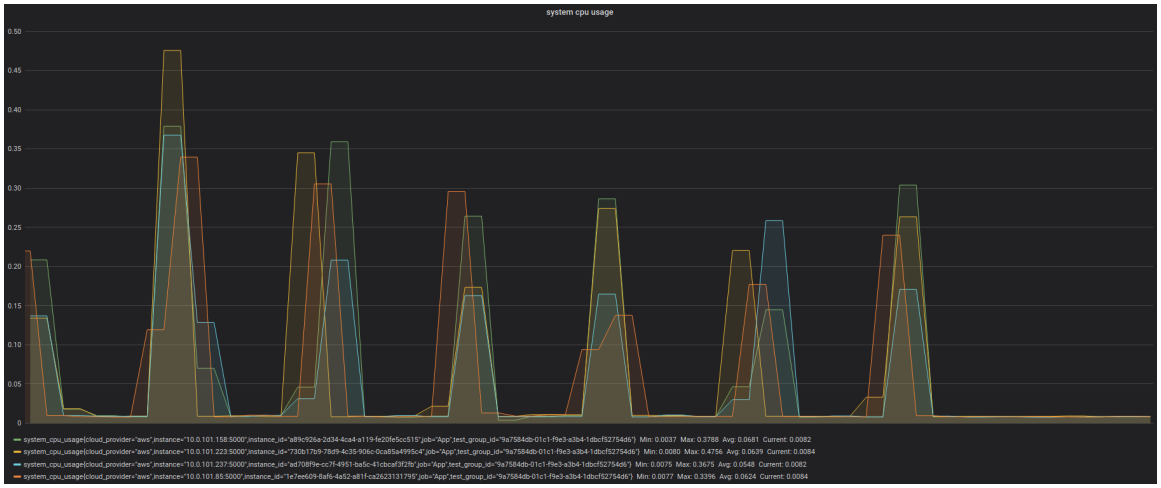

Figura 24: CPU usage do cenário 1

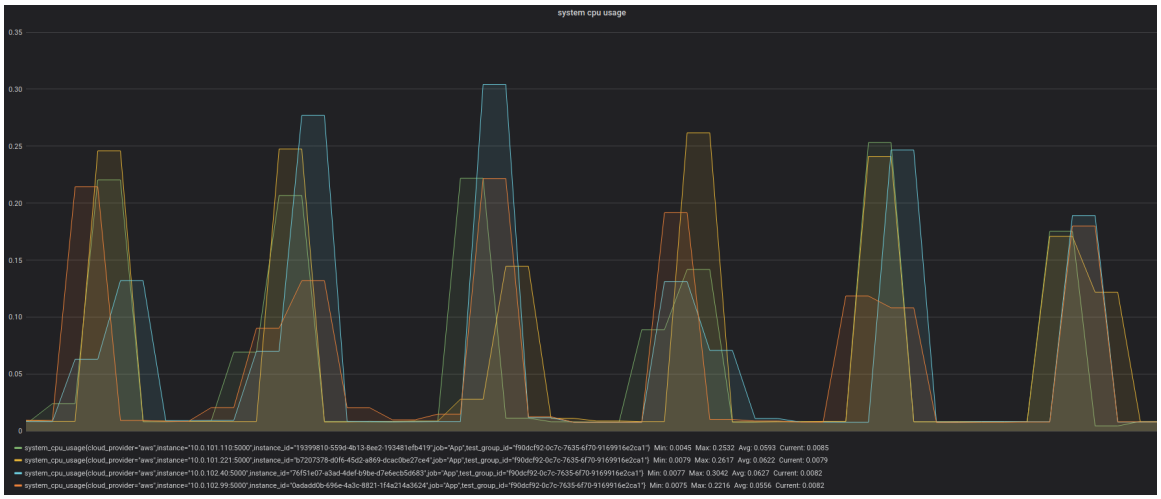

Figura 25: CPU usage do cenário 7

O cenário com posicionamento do tipo spread se mostrou ligeiramente superior ao cenário com o posicionamento de cluster no que diz respeito ao desempenho de CPU. Essa ligeira melhora pode ser dada ao fato do menor compartilhamento de recursos da solução, resultando em um melhor desempenho das instâncias.

- Memória

Avaliamos essa diferença no desempenho de memória utilizando a métrica customizada de tempo de operação em memória, e os resultados podem ser visualizados nas Figuras 26 e 27:

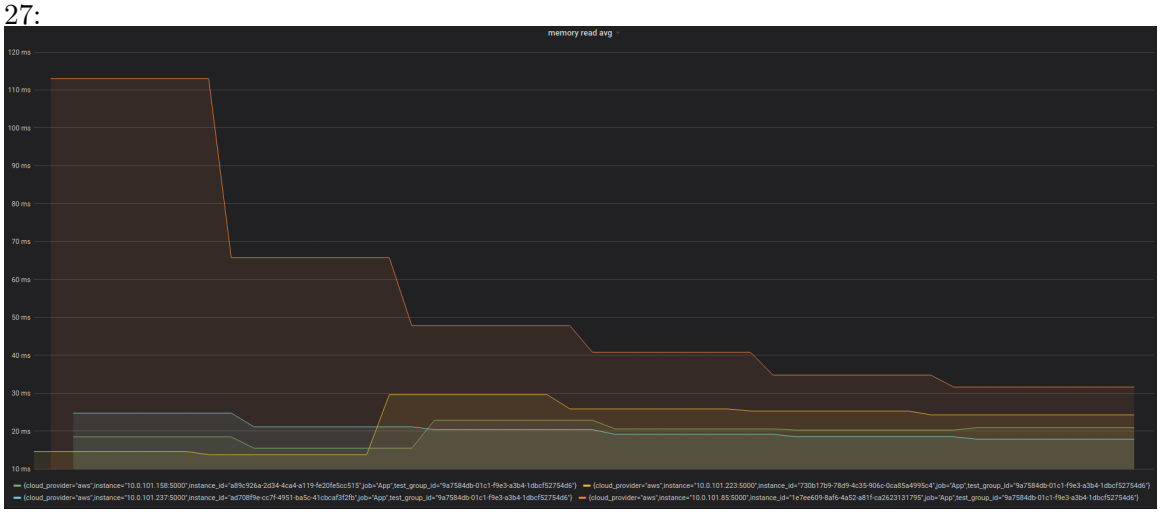

Figura 26: read memory time do cenário 1

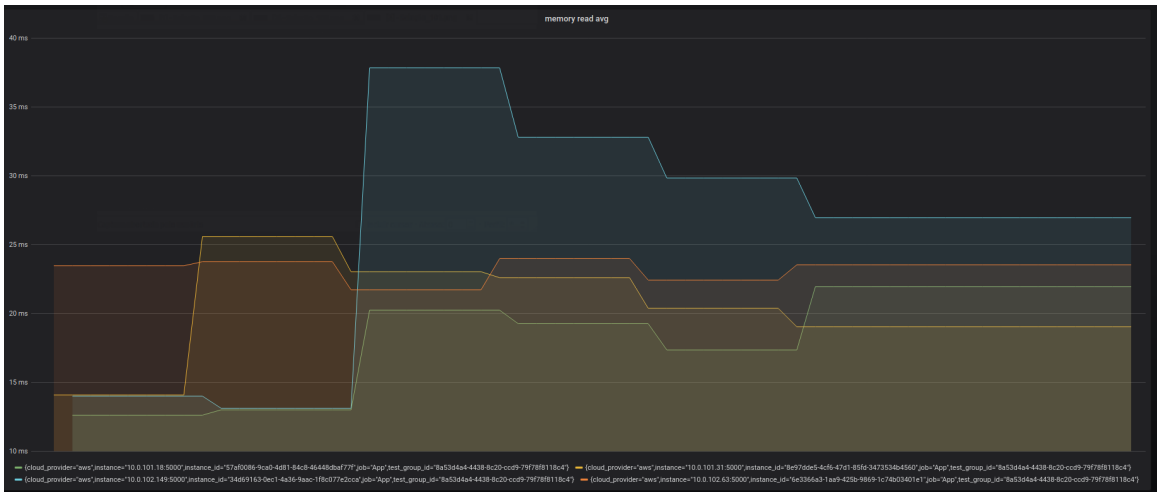

Figura 27: read memory time do cenário 7

Em ambos os cenários, as instâncias de cada cluster demonstraram um desempenho médio dentro de um mesmo intervalo de tempo, logo, acreditamos que a operação utilizada não impactou significativamente as instâncias a ponto de demonstrar uma vantagem de um posicionamento sobre o outro no que diz respeito ao desempenho de memória.

7 Conclusão

Antes de concluir as análises, vale ressaltar um comportamento marcante observado na fase de testes e obtenção dos dados. Para o conjunto de testes em CPU, a quantidade inicial de requisições por pacote era de 250 requisições. Esse valor se mostrou viável quando avaliado contra clusters de 4 instâncias, entretanto esse volume de requisições se mostrou muito prejudicial para clusters de apenas 1 instância, pois ao realizar os testes com esse volume, os containers "morriam"e eram reiniciados novamente, fazendo com que muitas requisições fossem perdidas ou que retornassem erro. Avaliamos que essa instabilidade dos containers se deu devido ao volume de requisições e decidimos alterar para o valor de 100 requisições por pacote, o que se mostrou viável para todos os cenários testados. Como observado em nossas análises, para instâncias de uso geral, se mostrou relativo ganho de desempenho de CPU para clusters com 4 instâncias, o que não deixa dúvidas com relação a vantagem de um escalonamento horizontal de infraestrutura para essas instâncias. Para instâncias com otimização em operações intensivas de memória, observamos que a instância mais robusta realmente se impõe em operações que intensifiquem especificamente operações de memória. Avaliando-se os demais parâmetros, a diferença não foi tão significativa, tendo em mente a vantagem no tratamento da solução horizontal nos casos de alto número de requests, devido ao balanceamento, em detrimento da versão verticalizada. Já para instâncias de uso otimizado para computação, percebemos pouca diferença de desempenho de CPU entre clusters de 4 instâncias e clusters de 1 instância, mas um desempenho superior de memória para clusters de 1 instância. No entanto vale avaliar o volume de requisições para aplicações que forem fazer uso desse grupo de instâncias, pois sem sombra de dúvidas, utilizar mais

instâncias irá proporcionar uma maior disponibilidade do serviço, visto que ele será capaz de suportar um volume maior de requisições por minuto.

Referências

[1] Scheuner, J.; Leitner, P., *Performance Benchmarking of Infrastructure-as-a-Service (IaaS) Clouds with CloudWorkBench* Companion Proceedings of 10th ACM/SPEC International Conference on Performance Engineering, 53-56, 2019. Disponível em: http://dx.doi.org/10.1145/3302541.3310294

[2] O que é CaaS https://www.redhat.com/pt-br/topics/cloud-computing/what-is-caas

[3] O que são containers e quais as vantagens de usá-los https://cloud.google.com/containers?hl=pt-br

[4] Introdução ao Terraform https://blog.4linux.com.br/introducao-ao-terraform/

[5] Grafana — O que é e como funcionam seus dashboards! https://www.opservices.com.br/grafana/

[6] Tipos de Instância do Amazon EC2 https://aws.amazon.com/pt/ec2/instance-types/

[7] Joel Scheuner, Philip Leitner, *A Cloud Benchmark Suite Combining Micro and Applications Benchmarks*

[8] Worldwide Public Cloud Service Revenue Forecast (Millions of U.S. Dollars) https://www.gartner.com/en/newsroom/press-releases/2020-07-23-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-6point3-percent-in-2020

[9] Repositório do projeto no GitHub https://github.com/pfg-cloud

[10] Horácio Martins, Filipe Araújo, Paulo Rupino da Cunha, *Benchmarking Serverless Computing Platforms*

[11] Joel Scheuner, Philip Leitner, *Estimating Cloud Application Performance Based on Micro-Benchmark Profiling*

[12] JMeter https://jmeter.apache.org/

[13] Pricing AWS https://aws.amazon.com/pt/ec2/pricing/on-demand/

[14] Grupos de posicionamento https://docs.aws.amazon.com/pt$_br/AWSEC2/latest/UserGuide/placement-groups.html$