Universidade do Minho
Escola de Engenharia
Mestrado em Engenharia Informática

Unidade Curricular de
Aplicações e Serviços de Computação em Nuvem

Ano Letivo de 2023/2024

Trabalho Prático

Grupo 41
a85635 - André Nunes
a91639 - Afonso Cruz
pg50239 - Beatriz Rocha
pg50289 - Catarina Martins
a83630 - Duarte Serrão

20 de dezembro de 2023

Índice

1 Introdução **1**

1.1 Contextualização 1

2 Arquitetura e Componentes da Aplicação **2**

3 Instalação e Configuração Automática da Aplicação **3**

3.1 Descrição das Ferramentas utilizadas 3

3.2 Arquitetura da Solução 4

4 Exploração e Otimização da Aplicação **6**

4.1 Ferramentas e metodologias utilizadas 6

4.1.1 Google Cloud Monitoring 6

4.1.2 Jmeter 6

4.2 Questões 7

4.3 Escalabilidade e resiliência 9

5 Análise de Resultados **11**

6 Conclusão e Trabalho Futuro **14**

1 Introdução

O presente relatório serve de suporte ao trabalho realizado no âmbito da unidade curricular de Aplicações e Serviços de Computação em Nuvem do curso de Mestrado (Integrado) em Engenharia Informática da Universidade do Minho.

1.1 Contextualização

Neste projeto foi pedido aos alunos que instalassem a aplicação *Laravel.io* recorrendo a serviços da Google Cloud, tal como Google Kubernetes Engine. A instalação e configuração da aplicação devem ser efetuadas de forma automática no menor número de passos manuais possíveis através do uso da ferramenta Ansible.

O *Laravel.io* é uma aplicação web, totalmente gratuita, que funciona como um *hub* central para a comunidade que utiliza a *framework Laravel PHP* e oferece diversos recursos de forma a informar e a auxiliar os membros da comunidade. As principais vantagens e funcionalidade do *Laravel.io* são:

1. Os fórums, que possuem discussões que promovem a entreajuda e partilha de conhecimento entre os desenvolvedores de software.
2. É uma aplicação web que incentiva a criação de um espaço de aprendizagem devido à partilha de tutoriais e recursos, nomeadamente *packages* e *libraries*, o que leva à introdução de diversas e novas soluções que mantêm todo o conhecimento atualizado.
3. Contribui para a criação de ligações e conexões entre desenvolvedores de software o que pode trazer benefícios e novas oportunidades profissionais.

Posto isto, ao longo deste relatório o grupo irá explicar detalhadamente todas as decisões tomadas relativas ao processo de instalação, monitorização e avaliação da aplicação *Laravel.io*.

2 Arquitetura e Componentes da Aplicação

O *Laravel.io* é uma aplicação web que é conhecida como o portal oficial da comunidade para o *Laravel*, uma *framework PHP* open-source, que tem como principal objetivo o desenvolvimento de sistemas web que utilizam o padrão MVC.

Assim, o *Laravel.io*, segue este mesmo padrão de arquitetura, o *Model-View-Controller* (MVC), e é dividido em três componentes:

- **Controlador:** É o responsável por tratar da comunicação entre a componente Modelo e Visualização. De forma geral, a metodologia de funcionamento deste passa por processar o *input* do utilizador, comunicar com o modelo e atualizar ou devolver a informação que volta para o utilizador através da componente visualização.
- **Modelo:** É o responsável pelo armazenamento, manipulação e recuperação de dados. No *Laravel.io* é utilizada a ferramenta *Eloquent ORM*.
- **Visualização:** É o responsável pela camada de apresentação, ou seja, é a componente que tem como função determinar como os dados vão ser expostos aos utilizadores. Neste caso, são utilizadas ferramentas como o *blade templates* e HTML com PHP incoporado.

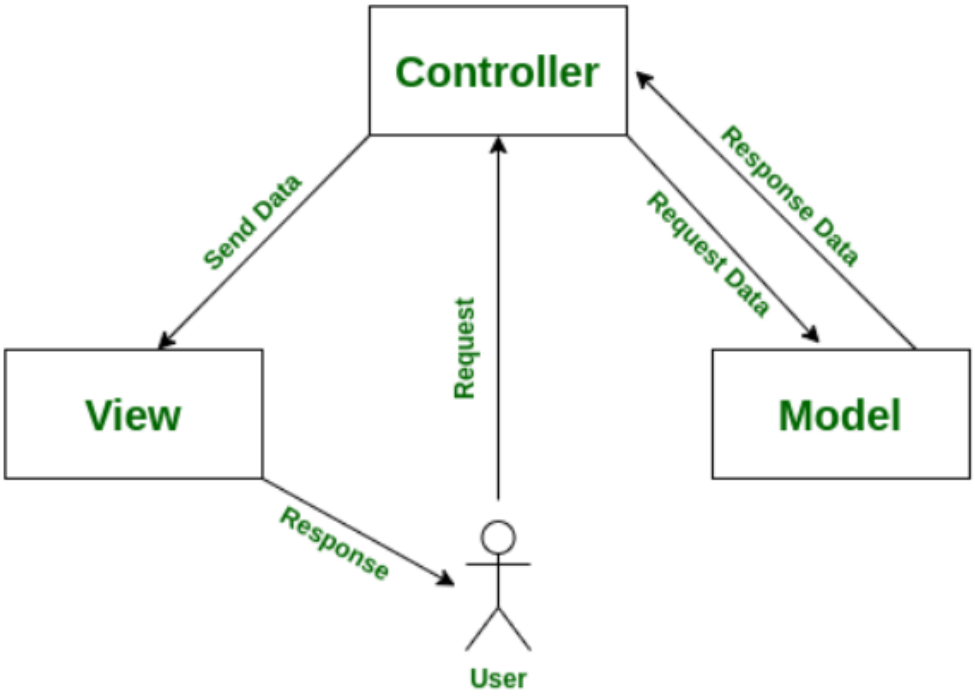

Figura 1: Exemplo do funcionamento do padrão MVC

3 Instalação e Configuração Automática da Aplicação

O desenvolvimento deste projeto divide-se em 2 tarefas principais:

- **Tarefa 1:** Automatizar a instalação e configuração da aplicação *Laravel.io* no **Google Kubernetes Engine (GKE)** da Google Cloud através da ferramenta **Ansible**
- **Tarefa 2:** Compreender e otimizar o desempenho, escalabilidade e resiliência do *Laravel.io*.

Para além disto, a realização da **tarefa 1** divide-se em dois *checkpoints*, o **#1** e o **#2**. Com *checkpoint* ***#1*** procura-se ter uma correta especificação e funcionamento da imagem *docker* para a camada aplicacional do *Laravel.io*. Enquanto que no **#2** pretende-se validar que existe um correto funcionamento da aplicação, após sua a instalação e configuração, e que a mesma é acessível a partir do exterior, através de um *browser*.

3.1 Descrição das Ferramentas utilizadas

De forma a implementar a solução, foram necessárias várias ferramentas como:

- **Google Kubernetes Engine (GKE):** Para a instalação e configuração da aplicação *Laravel.io*.
- **Google SQL:** Para armazenar os dados criados através do *deploy* da aplicação.
- **Google Compute Engine (GCE):** Para criar e destruir a base de dados no Google SQL, os *clusters* no GKE, e dar *deploy* e *undeploy* da aplicação *Laravel.io*. Tudo isto é feito com o auxílio de uma máquina virtual.
- **Ansible:** Para simplificar a configuração dos ficheiros usados no GCE e automatizar a criação e destruição do *cluster* e da base de dados e o *deploy* e *undeploy* do *Laravel.io*.

3.2 Arquitetura da Solução

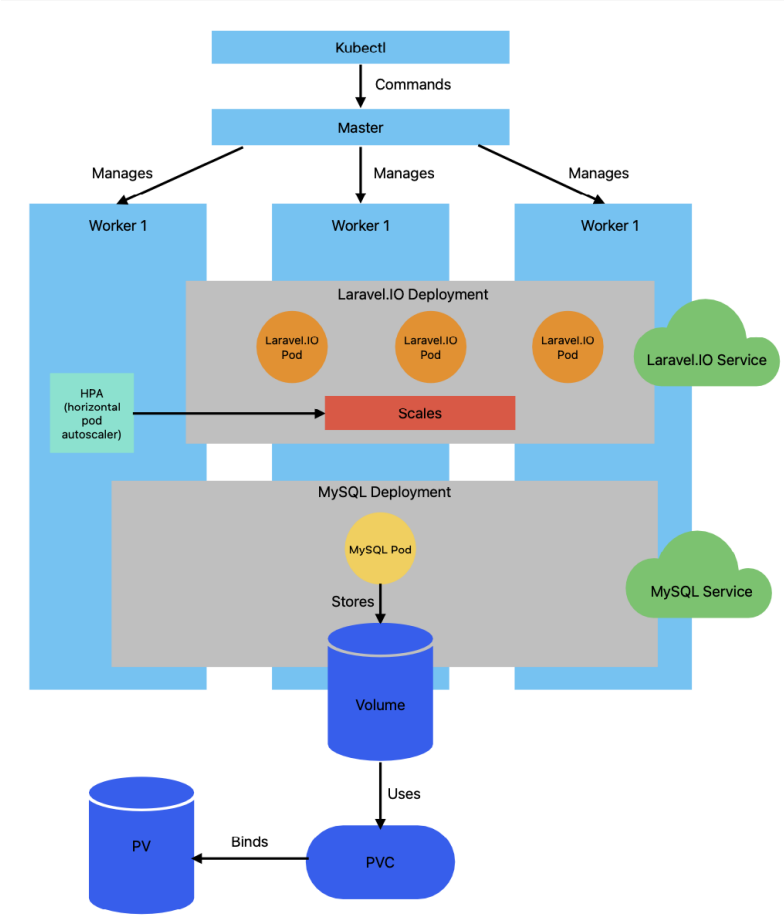

Figura 2: Arquitetura da solução implementada

A implementação do Kubernetes da plataforma Laravel.IO envolve contêineres que encapsulam Laravel, PHP e dependências em Pods. O MySQL é gerenciado por um serviço com persistência de dados garantida por meio de uma reivindicação de volume persistente (PVC). Utilizando o tipo de serviço NodePort para acesso externo, eliminamos a necessidade de um Load Balancer externo, simplificando a arquitetura.

O Horizontal Pod Autoscaler (HPA) ajusta dinamicamente as instâncias do Pod com base no tráfego, optimizando a utilização dos recursos durante os picos de carga. A implementação oferece escalabilidade, fiabilidade e gestão eficiente. O acesso externo é facilitado através do NodePort, atribuindo portas estáticas em cada nó.

- **Laravel.IO Deployment:** A aplicação Laravel.IO é implementada em contentores nos Pods do Kubernetes. Estes contentores encapsulam a aplicação Web Laravel, o tempo de execução PHP e outras dependências necessárias para a plataforma Laravel.IO.

- **MySQL Service and Deployment:** A implementação inclui um serviço MySQL responsável pela gestão da base de dados MySQL utilizada pela plataforma Laravel.IO. A implantação do MySQL garante alta disponibilidade e persistência de dados.
- **Persistent Volume Claim (PVC):** Para lidar com a persistência de dados, é utilizada uma Reivindicação de Volume Persistente (PVC). O PVC está ligado aos Pods MySQL, fornecendo armazenamento estável para ficheiros de base de dados, mesmo que os Pods sejam reprogramados ou substituídos.
- **Laravel.IO Service (NodePort):** O acesso externo à plataforma Laravel.IO é facilitado pelo tipo de serviço NodePort. Os clientes podem aceder à aplicação utilizando o endereço IP do nó e a porta estática atribuída, distribuindo o tráfego pelas instâncias disponíveis.
- **Horizontal Pod Autoscaler (HPA):** O Horizontal Pod Autoscaler (HPA) ajusta dinamicamente o número de Pods Laravel.IO em execução com base no tráfego e na utilização de recursos. Isso garante um desempenho ideal durante cargas de pico e reduz a escala durante períodos de menor demanda, otimizando a utilização de recursos.

Em resumo, esta arquitetura usando o NodePort aumenta a simplicidade e a eficiência, fornecendo acesso externo à plataforma Laravel.IO por meio de portas estáticas em cada nó. Esta abordagem elimina a necessidade de um balanceador de carga externo, tornando-a adequada para ambientes em que a simplicidade e a eficiência de recursos são prioritários, uma vez que só necessitamos de um IP para expor o serviço.

4 Exploração e Otimização da Aplicação

4.1 Ferramentas e metodologias utilizadas

De forma a ser possível implementar a solução, foram necessárias várias ferramentas como:

- **Google Cloud Monitoring:** Para simplificar a criação de Dashboards com as métricas escolhidas pela equipa.
- **Jmeter:** Para desenvolver testes de Benchmarking e efetuar a avaliação experimental da nossa solução.

4.1.1 Google Cloud Monitoring

O Google Cloud Monitoring é um poderoso serviço de monitorização no Google Cloud Platform, que fornece informações sobre o desempenho das aplicações e da infraestrutura. Recolhe diversas métricas dos serviços GCP e suporta métricas personalizadas, permitindo aos utilizadores monitorizar aspectos específicos das suas aplicações. Os dashboards personalizáveis oferecem visualizações em tempo real para uma visão geral abrangente.

O grupo focou-se na utilização de memória e CPU bem como throughput, requested cores e bytes ingested como métricas a ser avaliadas pois julgamos ser as mais relevantes no âmbito do nosso projeto, visto que são as mais sujeitas a demandas.

4.1.2 Jmeter

O JMeter é uma poderosa ferramenta de teste de desempenho de código aberto conhecida pela sua versatilidade e interface fácil de utilizar. Utilizado principalmente para testes de carga, stress e desempenho de aplicações Web, o JMeter permite aos utilizadores simular diversos cenários de utilizador e analisar o desempenho da aplicação sob cargas variáveis.

Para analisar o desempenho da nossa aplicação o grupo decidiu desenvolver um teste que efetua 1000 pedidos HTTP GET à aplicação de modo a verificar como esta se comportava devido a esta carga de trabalho.

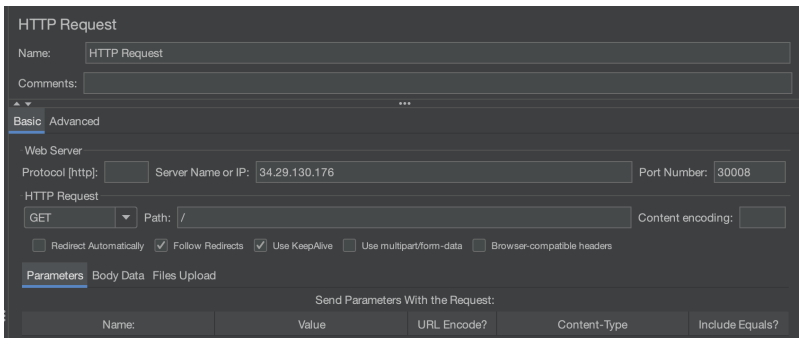

Figura 3: Teste do Jmeter

4.2 Questões

1. **Considerando a instalação base proposta pelo grupo para a Tarefa 1:**

   (a) **Para um número crescente de clientes, que componentes da aplicação poderão constituir um gargalo de desempenho?**

   O serviço NodePort utilizado para expor a aplicação externamente pode se tornar um ponto de gargalo, especialmente se o tráfego externo aumentar substancialmente. Implementar estratégias de balanceamento de carga ou considerar soluções mais avançadas, como um Ingress Controller, pode distribuir eficientemente a carga entre os nós.

   O serviço MySQL pode enfrentar gargalos conforme o número de consultas e transações aumenta. O ajuste inadequado do banco de dados, a ausência de índices eficientes ou a necessidade de escalonamento horizontal do MySQL para distribuir a carga podem ser áreas a serem exploradas para melhorar o desempenho.

   O PVC usado para persistência de dados no MySQL pode enfrentar gargalos em termos de taxa de I/O ou capacidade de armazenamento, especialmente com um grande volume de operações de leitura e gravação. Avaliar e otimizar a configuração do PVC, bem como considerar soluções de armazenamento mais avançadas, pode ser necessário para lidar com a carga crescente.

   (b) **Qual o desempenho da aplicação perante diferentes números de clientes e cargas de trabalho?**

   Efetuando testes de carga com o Jmeter o grupo percebeu que a aplicação não tinha resiliência para responder sequer a umas centenas de pedidos HTTP GET. Depois de implementado o HPA, a aplicação tem capacidade para responder a 1000 pedidos HTTP GET em simultâneo sem qualquer tipo de problema o que denota uma grande melhoria no que toca à escalabilidade e resiliência da solução proposta. Este tópico será abordado mais adiante neste relatório na secção de análise de resultados.

   (c) **Que componentes da aplicação poderão constituir um ponto único de falha?**

   O uso do serviço NodePort para expor externamente a aplicação pode ser um ponto único de falha se não for adequadamente gerenciado. Se um nó específico que hospeda o NodePort Service falhar, a acessibilidade externa à aplicação pode ser comprometida. Implementar balanceamento de carga ou considerar outras estratégias de acesso externo pode mitigar esse risco. Embora o PVC (Persistent Volume Claim) seja essencial para a persistência de dados, a falta de uma estratégia de backup adequada pode ser um ponto único de falha também.

2. **Com base nas respostas dadas para as questões anteriores:**

   (a) **Que otimizações de distribuição/replicação de carga podem ser aplicadas à instalação base?**

   De modo a mitigar os problemas identificados numa primeira fase de monitorização e benchmarking o grupo decidiu focar-se no servidor aplicacional implementando um Horizontal Pod Autoscaler (HPA). Esta téncica permite que o sistema faça scale up ou scale down de acordo com as suas necessidades. Consequentemente a nossa solução tornou-se muito mais escalável e resiliente.

   (b) **Qual o impacto das otimizações propostas no desempenho e/ou resiliência da aplicação?**

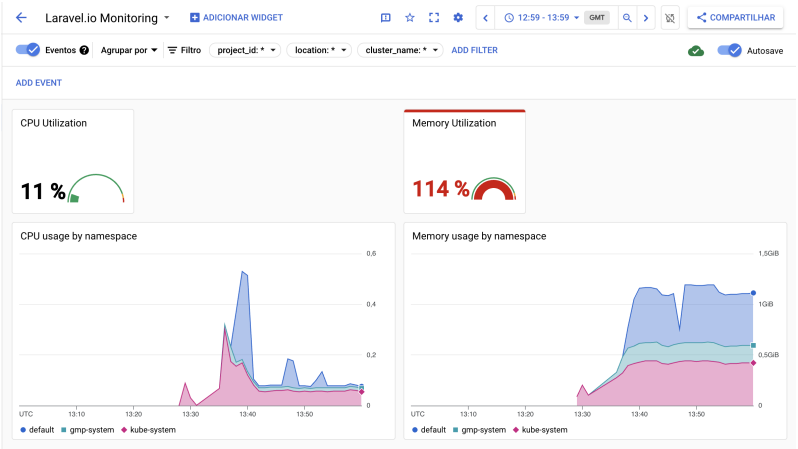

Figura 4: Monitorização pré implementação do HPA sem carga de trabalho

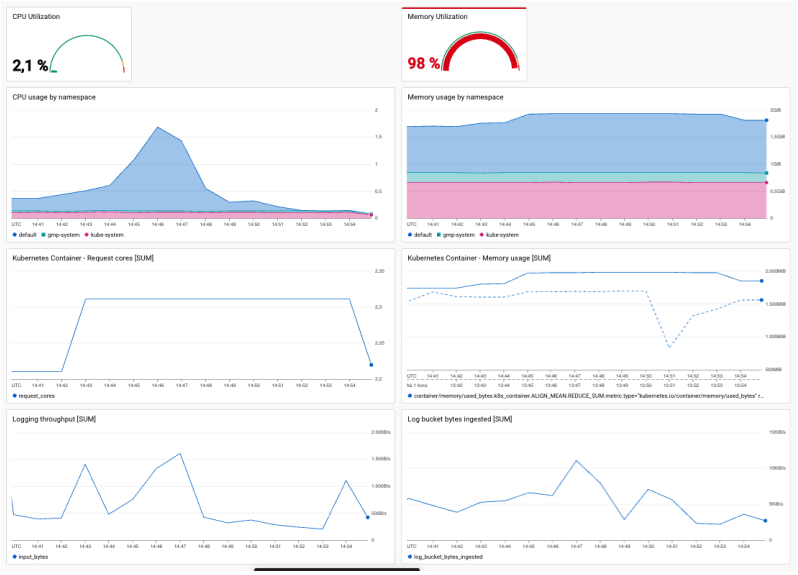

Figura 5: Monitorização pós implementação do HPA com carga de trabalho

Como podemos ver na imagem acima, após a implementação do HPA o uso de memória não ascende os 98% mesmo efetuando uma carga de trabalho de 1000 utilizadores realizando pedidos HTTP GET. O que contrasta e muito com os 114% monitorizados antes dessa implementação apenas efetuando o seed da DB e sem carga de trabalho.

4.3 Escalabilidade e resiliência

Esta implementação Kubernetes da plataforma Laravel.IO exemplifica uma arquitetura robusta concebida para escalabilidade e resiliência, tirando partido das principais funcionalidades e melhores práticas do Kubernetes. Este modelo de implementação garante o tratamento eficiente de cargas de trabalho variáveis, ao mesmo tempo que melhora a capacidade da plataforma para suportar e recuperar de potenciais falhas.

No centro desta arquitetura está a utilização do Kubernetes, uma poderosa plataforma de orquestração de contentores, para gerir e implementar a aplicação Laravel.IO. Os recursos de escalabilidade inerentes do Kubernetes tornam-no uma escolha ideal para cargas de trabalho dinâmicas, permitindo o dimensionamento contínuo da aplicação em resposta às mudanças nas demandas.

A utilização de um serviço MySQL e a implementação num cluster Kubernetes servem de base para a persistência e gestão de dados. Esta abordagem aumenta a resiliência do sistema global, garantindo uma solução de armazenamento fiável e consistente para a plataforma Laravel.IO. A integração de uma Reivindicação de Volume Persistente (PVC) fortalece ainda mais a persistência de dados, protegendo contra a perda de dados e fornecendo estabilidade mesmo em face de reprogramações ou substituições de Pods.

A implementação do Laravel.IO e o tipo de serviço NodePort associado contribuem para a escalabilidade e acessibilidade da plataforma. O tipo de serviço NodePort expõe a aplicação Laravel.IO externamente, permitindo que os clientes se conectem por meio de portas estáticas em cada nó. Isto facilita o tratamento de uma base de utilizadores crescente e garante que a aplicação permanece acessível à medida que a procura aumenta.

A introdução de um Horizontal Pod Autoscaler (HPA) amplia ainda mais o aspeto de escalabilidade da implantação. O HPA ajusta dinamicamente o número de Pods Laravel.IO em execução com base em métricas como a utilização da CPU e uso de memória. Esse escalonamento automatizado garante a utilização ideal dos recursos durante períodos de maior tráfego, aumentando a capacidade da plataforma de lidar com cargas de trabalho variáveis de forma eficiente.

Em termos de resiliência, a arquitetura foi concebida para resistir a falhas e garantir um funcionamento contínuo. A utilização de um serviço MySQL e de um PVC promove a fiabilidade dos dados, enquanto o HPA ajuda a manter um desempenho consistente, mesmo com cargas flutuantes. O tipo de serviço NodePort permite o acesso externo e, juntamente com os mecanismos de recuperação automatizados do Kubernetes, garante uma elevada disponibilidade através da redistribuição do tráfego em caso de falhas do Pod.

Em conclusão, a implementação do Kubernetes da plataforma Laravel.IO exemplifica uma solução bem concebida que dá prioridade à escalabilidade e à resiliência. A utilização do MySQL para a gestão de dados, juntamente com PVCs, garante a persistência e a fiabilidade dos dados. A combinação de serviços NodePort e HPA suporta a acessibilidade e o escalonamento dinâmico, tornando a plataforma apta a lidar com cargas de trabalho variáveis. Esta arquitetura não só acomoda o crescimento, como também melhora a capacidade da plataforma para recuperar de falhas, fornecendo uma base sólida para um ambiente Laravel.IO fiável e de elevado desempenho.

5 Análise de Resultados

No sentido de verificar como é que o nosso sistema reagiria a um elevado número de pedidos decidimos implementar um mecanismo de benchmarking utilizando a ferramenta Jmeter e o monitoring disponibilizado pelo GCP. Para tal utilizamos a ferramenta Jmeter para desenvolver um conjunto sintético de requests HTTP para a página {https://<app_IP>:<app_Port>/}. Começamos por definir uma Thread Group que representa os benchmark clients que efetuam requests HTTP. Após a definição de benchmark clients, implementamos um timer aos requests a ser feitos à página do Laravel.IO pretendida. Com isto, e variando o número de threads pertencentes à Thread Group e/ou o número de ms em que o timer efetua requests à página, pretendemos visualizar através das ferramentas de monitorização definidas anteriormente o comportamento e a variação das métricas monitorizadas no cluster. Para avaliar a nossa implementação, definimos um dado benchmark client irá executar 1000 pedidos ao URL mencionado anteriormente. Com este testes recolhemos alguns dados. Durante o período observado, podemos verificar que houve um aumento da percentagem de CPU e memória usados na totalidade de todos os nós e que como seria de esperar houve um aumento no loading throughput.

Figura 6: Logging throughput

Como podemos verificar pelo gráfico acima, durante a execução dos testes o Logging throughput foi variando de acordo com os pedidos recebidos, chegando a um pico de 1500 B/s e atingindo um limite inferior abaixo dos 500 B/s o que indica que a quantidade de testes afetou a resposta do sistema como seria de esperar. No entanto isto não foi suficiente para afetar a resiliência da solução proposta.

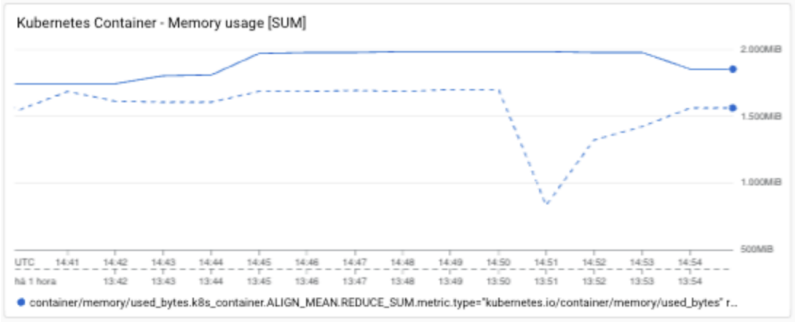

Figura 7: Memory Usage

Relativamente ao uso de memória foi possivel constatar que os teste fizeram com que o sistema consumisse a maior parte dos recursos dísponíveis. Sendo este o recurso mais usado pela aplicação. Com isto podemos constater que a nossa aplicação necessita de otimizações no âmbito da utilização de memória uma vez que a utilização deste recurso se aproxima bastante do máximo atingindo uma utilização de 98% dos recursos existentes.

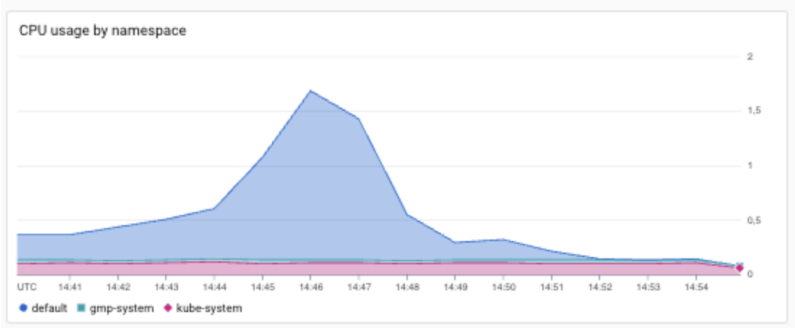

Figura 8: Cpu Usage

No que toca à utilização de CPU podemos constatar que os recursos alocados são mais que suficientes. Uma vez que apenas 11% dos recursos foram usados na resposta a todos os pedidos de teste. No entanto houve um aumento visível da utilização deste recurso.

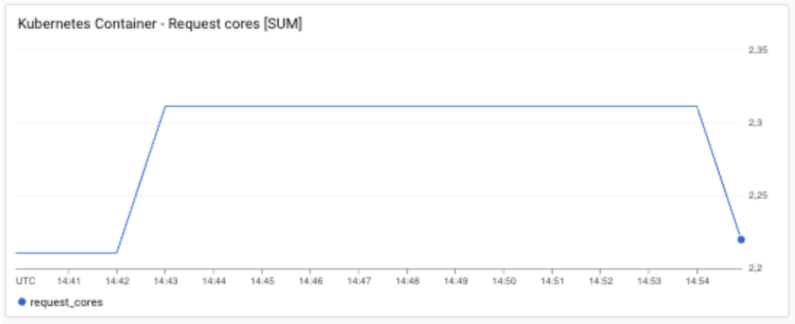

Figura 9: Requested cores

Com recurso à metrica de cores requisitados podemos verificar que a utilização do HPA é válida, uma vez que com o aumento de pedidos está métrica aumenta também o que prova que são criados mais pods de acordo com as necessidades do sistema. Utilizando o HorizontalPodAutoScaler, quando a aplicação se encontra sobrecarregada, isto é, com uma taxa de utilização de CPU superior a 70%, será criada mais uma réplica do servidor aplicacional. Este componente é iniciado com um pod e serão criadas réplicas da forma explicada anteriormente, sendo o número máximo criado 5.

O Jmeter também forneceu algumas informações relevantes para a análise da performançe da aplicação.

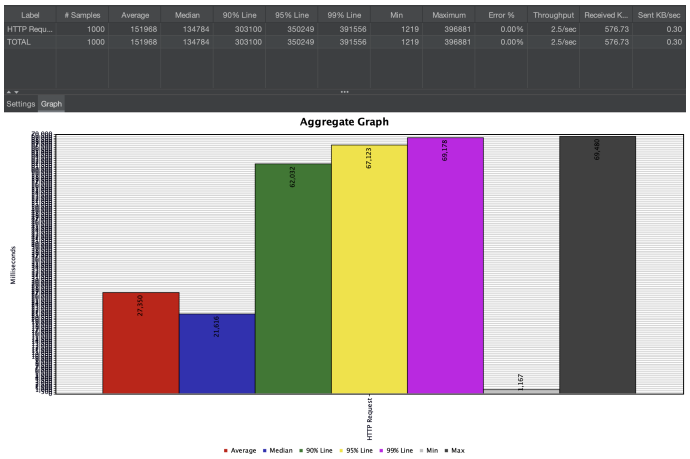

Figura 10: HTTP Request Agreggate graph

Este gráfico é de extrema relevância pois permite-nos perceber quais os tempos de resposta aos pedidos feitos. Podemos verificar que a média de tempo de resposta foi de 27,35 milisegundos e que a mediana foi de 21,616 milisegundos. Houve também um mínimo de 1,167 milisegundos e um máximo de 69,48 milisegundos. Estes resultados mostram que a nossa implementação responde em tempos basntante rápidos e que está preparada para lidar com um grande número de pedidos num curto espaço de tempo.

6 Conclusão e Trabalho Futuro

Ao examinar a arquitetura e a implementação da aplicação Laravel.IO no ambiente Kubernetes, os detalhes críticos em torno de componentes como o serviço NodePort, o serviço MySQL com PVC, o Horizontal Pod Autoscaler (HPA) e a persistência de dados foram cuidadosamente analisados. Foi sublinhado que, embora a implementação atual seja robusta, certos componentes podem necessitar de ajustes à medida que a base de utilizadores se expande.

As principais áreas de foco foram identificadas, incluindo o serviço NodePort, o serviço MySQL e o código da aplicação Laravel.IO, com sugestões específicas de otimização. A discussão sobre possíveis pontos únicos de falha enfatizou a importância de estratégias como replicação, backups adequados e métodos de recuperação eficazes.

Em resumo, a arquitetura delineada demonstra uma sólida compreensão dos princípios do Kubernetes e das práticas recomendadas de desenvolvimento. As sugestões fornecidas visam melhorar a escalabilidade, a resiliência e a eficiência geral da aplicação. No entanto é necessário efetuar otimizações que visem melhorar ainda mais a utilização dos recursos de memória uma vez que estes são os recursos em maior demanda pela aplicação.