ASCN

**Universidade do Minho**
Escola de Engenharia
Mestrado em Engenharia Informática

# Aplicações e Serviços de Computação em Nuvem

Ano lectivo 2023/2024

## Trabalho prático

Grupo 14

Ana Rita Santos Poças, PG53645

João Pedro Vilas Boas Braga, PG53951

Miguel Silva Pinto, PG54105

Orlando José da Cunha Palmeira, PG54123

Pedro Miguel Castilho Martins, PG54146

22 de Dezembro de 2023

# Índice

# Índice de figuras

# Índice de tabelas

# 1 Introdução

Neste trabalho prático da unidade curricular de Aplicações e Serviços de Computação em Nuvem, o objetivo do grupo foi a instalação da aplicação *Laravel.io* utilizando os serviços da Google Cloud. Com base no conhecimento adquirido nas aulas práticas e teóricas, o processo de instalação foi totalmente automatizado através da ferramenta Ansible.

Foi utilizada a ferramenta Kubernetes através do serviço Google Kubernetes Engine que fornece uma plataforma de gestão automatizada dos containers que constituem uma aplicação, visando mecanismos de replicação e escalabilidade automática da aplicação.

Ao longo deste relatório, serão apresentadas de forma detalhada todas as decisões tomadas durante o processo de instalação da aplicação *Laravel.io*. Também será abordada a escolha da metodologia de autoscaling, bem como a implementação do balanceamento de carga utilizando o serviço Google Kubernetes Engine. Adicionalmente, será discutido o uso da ferramenta *JMeter* e dos serviços de monitorização da Google Cloud Platform para avaliação contínua da aplicação *Laravel.io*.

# 2 Instalação e Configuração automática

Para a implementação do projeto recorremos ao Google Kubernetes Engine (GKE) da Google Cloud através da utilização do Ansible, uma plataforma de automatização de tarefas de configuração e gestão de sistemas.

O Ansible desempenha um papel crucial na administração abrangente da aplicação, gerindo eficientemente o ambiente Kubernetes por meio de uma variedade de módulos, como o *k8s* e *k8s_info*. Estes módulos capacitam a criação e o controlo dos diversos componentes Kubernetes essenciais para orquestrar os containers que compõem a aplicação.

Para além de permitir automatizar passos de configuração de sistemas, o Ansible fornece uma ferramenta que adiciona uma camada de segurança, o Ansible Vault que permite encriptar e proteger informação de configuração sensível, como nomes de utilizadores e *passwords*. No nosso projeto, o Ansible Vault foi utilizado para proteger a informação da conta de teste que é utilizada em tarefas de configuração do sistema, apenas permitindo o acesso da informação através de uma password.

Após o arranque do cluster GKE, é possível executar a aplicação e todos os seus componentes através do comando `ansible-playbook laravelio-deploy.yml`. Para encerrar a aplicação usa-se o comando `ansible-playbook laravelio-undeploy.yml`. Para parar a aplicação e voltar a executar a mesma sem que sejam perdidos os dados da aplicação, utiliza-se o playbook `ansible-playbook reload-laravel.yml` que basicamente reutiliza os playbooks de *undeploy* e *deploy* com certas flags para não apagar os dados da base de dados.

# 3 Arquitetura inicial de base

A aplicação *Laravel.io* foi construída com Laravel, uma framework PHP especializada em desenvolvimento Web. Utiliza uma base de dados para armazenar informações cruciais, como contas de utilizadores, publicações, comentários e outros dados relevantes. Uma característica do *Laravel.io* é a flexibilidade na escolha da driver de base de dados que é utilizado pelo servidor aplicacional, e o grupo optou pela utilização do MySQL.

Recorreu-se à utilização do Google Kubernetes Engine (GKE) da Google Cloud para hospedar a aplicação desenvolvida, sendo necessárias as imagens Docker da aplicação *Laravel.io* e da sua base de dados. A partir destas imagens Docker será possível criar os containers que farão parte da aplicação e serão orquestrados pelo ambiente Kubernetes.

A aplicação fica acessível ao exterior através de um ***Service*** do tipo *Load Balancer* na porta 80 num IP fornecido pelo Google Kubernetes Engine.

## 3.1 Análise de escalabilidade da solução inicial

Com base numa configuração inicial simples, que consiste numa única instância de servidor aplicacional e uma instância de base de dados, podemos evoluir para uma solução mais robusta ao explorar mais profundamente os seus problemas.

A escalabilidade surge como uma questão crucial nesta configuração básica, onde o aumento no número de clientes resulta num declínio da qualidade do serviço. Isto ocorre devido à limitação representada pelas únicas instâncias do servidor aplicacional e da base de dados, que se transformam em gargalos de desempenho sem capacidade de escalabilidade.

O **servidor aplicacional** sobrecarregado com um número crescente de clientes, não irá conseguir assegurar respostas eficientes aos pedidos, devido à pressão imposta no servidor aplicacional para lidar com diversos pedidos simultaneamente.

Além disso, outra potencial causa de degradação da aplicação pode ser originada por uma sobrecarga no **sistema de base de dados**, resultante de um acúmulo de queries de escrita à base de dados, sendo estas queries mais morosas dada a necessidade de um cuidado especial para garantir a coerência da informação do sistema.

A falta de redundância para cada componente da aplicação é uma vulnerabilidade significativa para a aplicação. Se ocorrer uma falha em qualquer um dos componentes, toda a aplicação é comprometida, o que representa uma desvantagem crítica em termos de fiabilidade. Se o servidor aplicacional falhar, seja devido a uma sobrecarga de pedidos ou qualquer outra circunstância excecional, a aplicação deixa de ser capaz de responder pedidos. Se a base de dados falhar, o servidor aplicacional não será capaz de aceder aos recursos necessários para fornecer uma resposta, uma vez que a base de dados não se encontra disponível com a sobrecarga.

Na seguinte tabela mostramos o desempenho da aplicação perante diferentes números de clientes simultâneos a aceder à pagina inicial da aplicação, num intervalo de 300ms em 300ms.

| **Num. users** | 5 | 50 | 100 | 500 |
|---|---|---|---|---|
| **Resp. time mean (s)** | 1.002 | 11.396 | 23.340 | - |
| **Min. time (s)** | 0.848 | 1.083 | 1.122 | - |
| **Max. time (s)** | 2.199 | 13.278 | 25.400 | - |

Tabela 3.1: Desempenho da aplicação com diferentes threads(users)

Se considerarmos que o tempo máximo aceitável para resposta seja de 5 segundos, a estrutura base e inicial da aplicação não é capaz de aguentar com 50 utilizadores a requisitar acesso à aplicação, piorando cada vez mais a sua qualidade de serviço à medida que o número de clientes cresce. Com 500 utilizadores a aplicação nem sequer foi capaz de despachar pedidos em tempo útil para a verificação dos seus resultados.

Uma solução para o problema da escalabilidade consiste na replicação dos componentes da aplicação. Isto permitirá de forma geral que a aplicação tenha uma maior capacidade computacional para responder aos pedidos de forma mais rápida e frequente, eliminando os seus gargalos de desempenho.

# 4 Arquitetura e componentes da aplicação

Após a análise da solução inicial, foram implementadas as medidas de replicação para a aplicação no ambiente Kubernetes em que ela está hospedada. Neste ambiente Kubernetes estarão definidos diversos componentes que em conjunto irão ser responsáveis pela manutenção e correto funcionamento da aplicação.

A seguinte imagem fornece uma visão geral dos diversos componentes no ambiente Kubernetes e as suas interações.

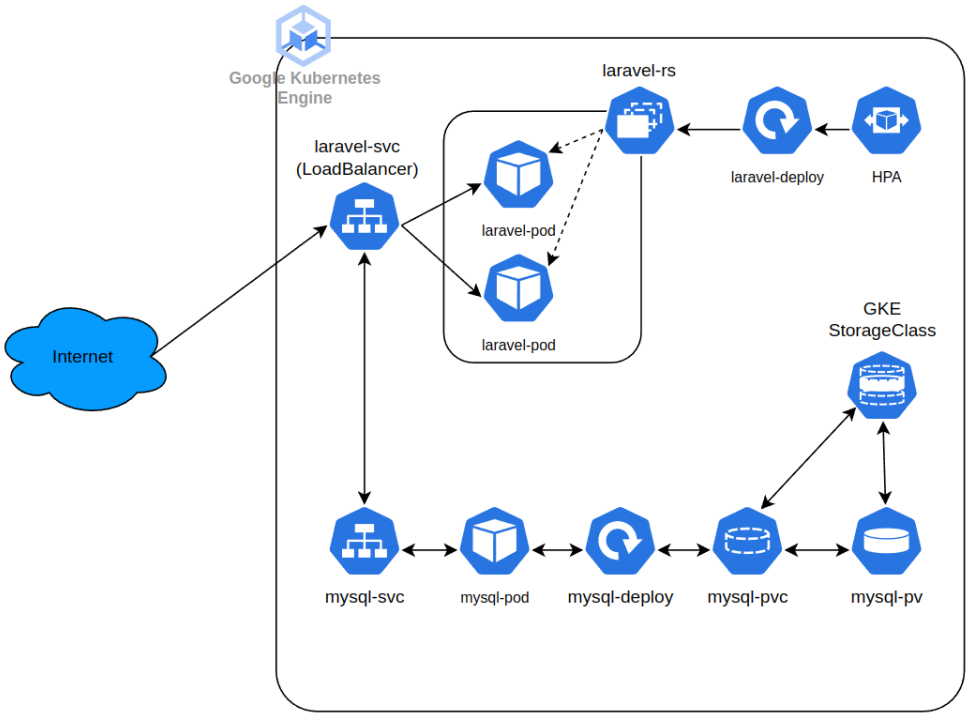


Figura 4.1: Infraestrutura Kubernetes

Na nossa arquitetura, destacamos a existência de um *Persistent Volume Claim* (*PVC*) integrado no *Pod* MySQL que permite a **persistência dos dados** da aplicação. Este *PVC* tira partido dos *StorageClass* padrão que o GKE nos fornece, sem necessidade de criarmos e gerirmos instâncias de *Persistent Volumes*.

Desta forma asseguramos a persistência dos dados da aplicação que estão na instância da base de dados MySQL, oferecendo a sua continuidade independentemente das variações no ciclo de vida dos *pods* MySQL.

A equipa decidiu escolher o servidor aplicacional como o alvo da replicação, uma vez que é possível obter melhorias mais significativas no desempenho de forma geral. Para a replicação da base de dados seriam necessários mecanismos de replicação mais complexos sem conseguir resolver totalmente o seu principal problema relacionado com as operações de escrita. Um desses mecanismos de replicação seria um StatefulSet que garante a coerência da informação entre as diversas réplicas, exigindo um maior rigor e coordenação na replicação do componente.

O *HorizontalPodAutoscaler* (*HPA*) permite aumentar ou reduzir o número de *pods* do *Deployment* relativo aos servidores aplicacionais, consoante a utilização de CPU. Este componente monitorizando a aplicação previne problemas de insuficiência computacional em alturas com maior tráfego e impede que haja uma sub-utilização de recursos em alturas de menor tráfego na aplicação.

De forma a garantir coerência de sessões foi necessário garantir uma afinidade de sessão no *Service* (Load-Balancer) que serve de porta de entrada aos pedidos dos utilizadores para os servidores aplicacionais. Esta afinidade de sessão foi definida através do campo *SessionAfinity* da especificação Kubernetes do *Service*, com o valor ClientIP. Desta forma é garantido que um certo IP irá conectar-se sempre à mesma instância do *Pod* do servidor aplicacional, garantindo que as informações relativas à sua sessão sejam coerentes nos seus vários acessos à aplicação.

No entanto, é importante notar que, embora a abordagem de afinidade de sessão com base no endereço IP do cliente (ClientIP) seja válida para a utilização real da aplicação pelos diversos utilizadores, ela traz desvantagens quando se trata de avaliar o desempenho da aplicação através de benchmarking. Isto deve-se ao facto de que para uma avaliação eficaz do desempenho com carga simulada, torna-se necessário ter uma variedade de endereços IP diferentes, permitindo assim a execução de pedidos em paralelo e proporcionando uma representação mais abrangente das capacidades da aplicação em condições de uso intensivo.

# 5 Avaliação de desempenho

A garantia de desempenho robusto e confiável de uma aplicação é crucial para proporcionar uma experiência positiva aos utilizadores, logo a utilização de ferramentas adequadas é essencial.

## 5.1 Ferramentas utilizadas

Neste contexto, destacamos o Apache JMeter, uma ferramenta open-source amplamente utilizada para testes de desempenho. Esta ferramenta é capaz de simular carga e posteriormente recolher estatísticas relativas ao desempenho da aplicação consoante a carga simulada que lhe foi imposta.

Para observar métricas mais relacionadas com as máquinas da Google Cloud em que estão hospedados os componentes da aplicação, como CPU e memória, recorreu-se ao Google Cloud Monitoring. É uma ferramenta informativa que indica diversas informações relativamente ao estado dos componentes da nossa aplicação no ambiente da Google Cloud, permitindo criar Dashboards com a informação que desejarmos.

Devido à afinidade de sessão imposta pelo *Load Balancer*, não é possível seguir a mesma abordagem de benchmarking que foi seguida nas aulas em que a carga simulada para a aplicação era proveniente de uma única máquina. Caso toda a carga provenha de uma única máquina, isto é, do mesmo endereço de IP, resultaria na alocação exclusiva de todos os pedidos para um único *pod*, o que contradiz o propósito da replicação dos componentes do servidor aplicacional. De forma a resolver este problema, criamos 8 máquinas virtuais (máximo permitido numa mesma região) através da Google Cloud, que irão gerar a carga simulada que o *Load Balancer* irá ser capaz de distribuir pelos *pods*, amenizando o problema criado pela necessidade de afinidade de sessão.

Estas 8 máquinas virtuais são criadas e provisionadas automaticamente através do playbook Ansible `gcp-create-vms.yml`. As máquinas virtuais estão localizadas numa região (us-east1-b) diferente da localização do Cluster Kubernetes (us-central1-a) pois a Google Cloud restringe um limite máximo de fornecimento de 8 endereços IP por região. Esta distância entre a aplicação e os utilizadores simulados é algo a ter em conta ao analisar o valor da latência de respostas da aplicação, mas como todos os testes serão feitos na mesma região para a mesma região, a reproducibilidade é garantida.

## 5.2 Monitorização

Relativamente à monitorização, criamos a seguinte dashboard que contém gráficos com informações que consideramos mais importantes. Esta *dashboard* mostra várias informações relativas à infraestrutura da aplicação, desde o número de réplicas do componente do servidor aplicacional, até às percentagens de CPU requisitado pelos pods do servidor aplicacional.

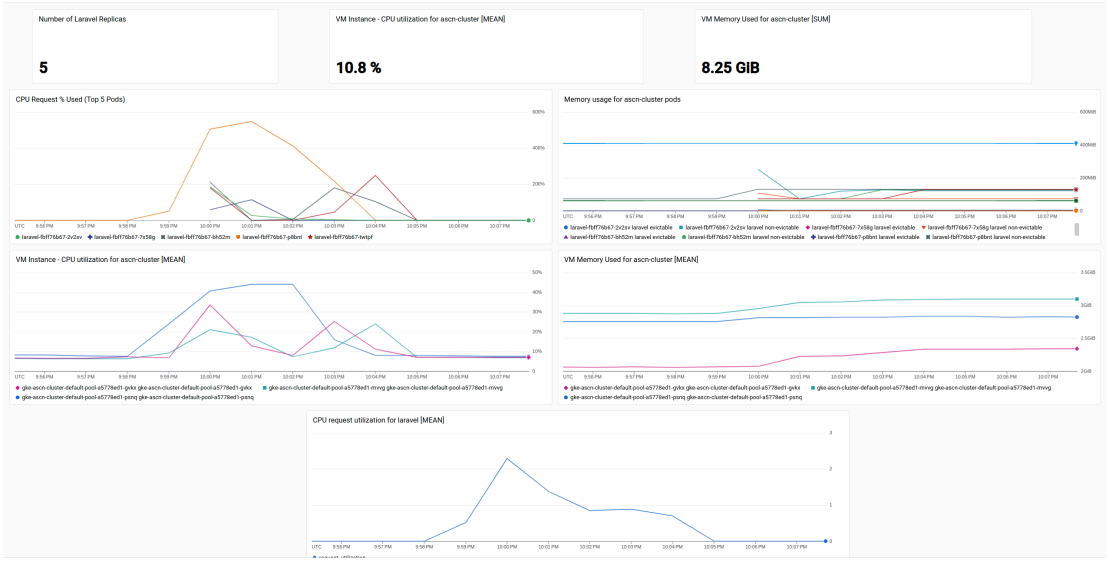


Figura 5.1: Dashboard

Para importar esta dashboard para um projeto GCP, foi criado um playbook Ansible `gcp-import-dashboard.yml`, que importa o ficheiro JSON da dashboard, para o projeto definido na variável Ansible *gcp_project* e permite o acesso às informações da dashboard.

## 5.3 Análise de desempenho

Aqui apresentamos uma tabela com o desempenho medido com um número crescente de réplicas. Estes valores são resultantes de 80 *users*(threads) simultâneos a enviar pedidos para a página principal num intervalo de 300ms ao longo de 15 iterações.

| **Nº réplicas** | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| **Resp. time mean (s)** | 9.405 | 7.449 | 6.930 | 6.019 | 3.247 |
| **Min. time (s)** | 0.441 | 0.414 | 0.463 | 1.106 | 0.458 |
| **Max. time (s)** | 11.706 | 11.715 | 11.727 | 11.739 | 7.050 |
| **Throughput (req/s)** | 8.63 | 13.48398 | 17.38131 | 17.5296 | 24.47863 |

Tabela 5.1: Desempenho com crescente número de réplicas do servidor aplicacional

Pela tabela pode-se constatar que existem benefícios na replicação do servidor aplicacional, notando-se uma descida no tempo de execução e um aumento do *throughput* (quantidade de pedidos respondidos por segundo) com o aumento do número de réplicas.

Nesta próxima tabela apresenta as métricas resultantes de um diferente número de utilizadores simultâneos, num intervalo de 300ms durante 15 iterações, com 5 réplicas do servidor aplicacional.

| Nº threads | 16 | 32 | 64 | 128 |
|---|---|---|---|---|
| Resp. time mean (s) | 0.823 | 1.25 | 3.288 | 5.354 |
| Throughput (req/s) | 15.24 | 22.7 | 23.3 | 24.6 |
| CPU request | 0.8 | 1.24 | 2.89 | 3.62 |

Tabela 5.2: Métricas obtidas para diferentes cargas

Com os valores obtidos podemos verificar o CPU request que representa a fração de CPU solicitada que está em uso na instância, para cada tipo de carga a que a aplicação é submetida e também tirar elações acerca da eficácia de ter 5 réplicas do servidor aplicacional. Dado que os valores de throughput andam à volta do valor de 24 pedidos por segundo, isto indica que com mais do que 24 pedidos por segundo a chegar à aplicação, esta acumula-os para serem respondidos, o que leva ao aumento do tempo médio de resposta. Para que a aplicação seja capaz de atender atempadamente a mais do que 24 utilizadores em simultâneo a enviar pedidos por cada segundo, o número de réplicas da aplicação tem de ser maior ou os recursos computacionais atribuídos a cada pod terão de ser melhorados.

## 5.4 Escalamento automático

Para testar a aplicação e o seu *HorizontalPodAutoscaler* (HPA) em ação demonstrou-se ser uma tarefa complicada devido à afinidade de sessão implementada na aplicação. Isto não seria necessariamente um problema num contexto real, pois quando precisamos de escalar a aplicação, será para que seja possível atender vários clientes, ou seja diferente máquinas. Esta situação apenas se apresenta como um problema no que toca a benchmarking e à dificuldade em arranjar uma vasta gama de IP's distintos.

Apesar de termos as 8 máquinas virtuais a simular os pedidos vindos de diferentes IP's, o *Load Balancer* irá imediatamente atribuir os IP's dessas máquinas ao *pod* existente (assumindo que se começa com apenas um *pod*). Isto fará com que a subsequente replicação de servidores aplicacionais após a deteção do aumento do tráfego, seja inútil, pois todos os subsequentes pedidos serão na mesma feitos pelas máquinas previamente atendidas e consequentemente, serão atendidos pela mesma instância do *pod* do servidor aplicacional.

Para tentar amenizar esse problema, no playbook Ansible que faz o *load testing* através das máquinas virtuais, tentamos implementar uma espécie de *rampup period* opcional entre o começo do envio das cargas de cada máquina virtual, ou seja, a VM1 começa sem delay, a VM2 começa com 10 segundos de delay, a VM3 com 20 segundos de delay, e assim sucessivamente. Isto surge como uma tentativa de impedir que a sessão de todos os endereços IP das máquinas sejam atribuídos à máquina inicial no ínicio do envio da carga simulada.

A seguinte figura demonstra o comportamento do HPA com um *load* de 80 users simultâneos entre um intervalo de 300 ms por 15 iterações com o *rampup period* mencionado previamente.

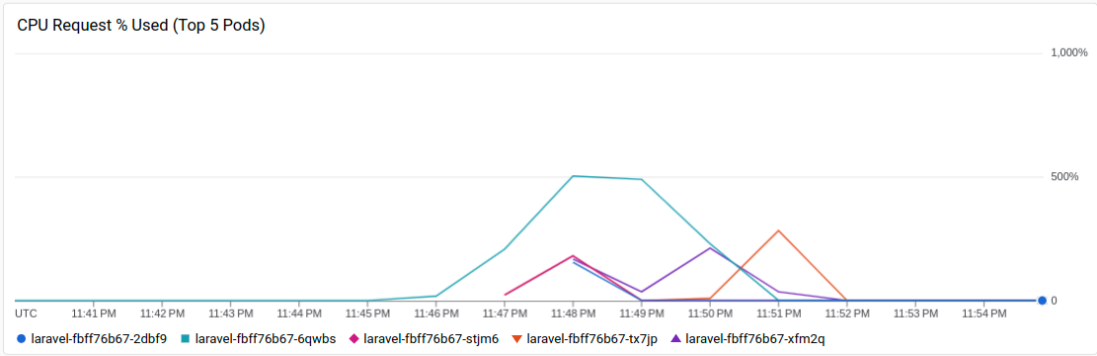


Figura 5.2: Comportamento do HPA

Como podemos ver pela figura, o pod inicial recebe vários pedidos o que faz aumentar a utilização de CPU, sendo esta utilização detetada pelo HPA que irá escalar o número de réplicas do servidor aplicacional. Pode-se detetar este escalamento, no surgimento das novas linhas que representam os pods/réplicas inicializados pelo HPA. Apesar da implementação deste *rampup period*, o pod inicial ainda ficou com uma boa parte da carga. Isto deveu-se ao facto que o HPA não recebe as informações das métricas do cluster imediatamente, havendo um certo período de tempo em que o HPA ainda não sabe totalmente do estado da métrica. isto conjugado com o facto de que a réplica gerada pelo HPA não é instantaneamente criada, justifica a maior carga que é notada no primeiro pod nesta experimentação.

# 6 Conclusão

A realização deste trabalho prático foi fundamental para consolidar todo o conhecimento adquirido ao longo das aulas teóricas e práticas da disciplina.

O grupo considera que obtivemos êxito ao automatizar de maneira significativa o processo de provisionamento e deployment da aplicação *Laravel.io*. Além disso, destacamos a capacidade da aplicação armazenar os dados de forma persistente e ser capaz de escalar de acordo com o número de pedidos que chegarem à aplicação.

No entanto, reconhecemos que há espaço para melhorias. Um dos principais pontos a melhorar seria a gestão de coerência de sessões, na qual a afinidade de sessão relativamente ao IP da máquina poderia ser eliminada, promovendo uma melhor distribuição de carga pelas réplicas do servidor aplicacional existentes. Essa remoção da afinidade poderia ser possível através da utilização da ferramenta Redis para a gestão de sessões. Esta ferramenta permitiria um acesso comum a informações de sessão para todas as réplicas do servidor aplicacional, descartando a necessidade de um cliente se conectar sempre ao mesmo *pod* para manter a coerência de sessão. Outra melhoria seria aplicar replicação também para a base de dados MySQL através do componente Kubernetes StatefulSet. Este componente faria com que a base de dados fosse replicável, mantendo a coerência dos dados das várias réplicas o que é crucial ao lidar com bases de dados.

Em suma, aprendemos conceitos de Kubernetes, que é uma ferramenta bastante em voga nos dias de hoje no que toca à gestão de aplicações em containers, a ferramenta de automatização Ansible, e tivemos a possibilidade de interagir com a Google Cloud Platform num contexto prático. Com tudo isto, o trabalho prático deu-nos a conhecer os vários desafios da instalação de uma aplicação de software no mundo real e algumas das diversas ferramentas disponíveis para solucionarmos os seus problemas.