Relatório Final

Taita Ayane , Bruno Claudino ,Bruno Rodrigues , Philip Akpanyi , William Melo

[1]Departamento de Ciência da Computação – Universidade Federal de Roraima (UFRR)

taitacheetahbm@gmail.com

brunocclaudino@gmail.com

brunorcx@hotmail.com

pakpanyi@gmail.com

williamjsmelo@gmail.com

1. Introdução

Este trabalho tem o objetivo de desenvolver uma aplicação distribuída que utiliza um chatbot do Telegram para receber a informação de geolocalização dos usuários do chatbot, armazenando-as em um banco de dados para posteriormente serem plotadas em um mapa a ser exibido em uma interface web. O chatbot é programado em Python, com o uso do wrapper API python-telegram-bot. A aplicação se vale de comunicação por sockets, através de partes interdependentes utilizando o Docker. Optou-se pelo uso do MongoDB como banco de dados orientado a documentos. O serviço de nomes e técnicas de coordenação, bem como modelos de consistência e replicação serão alcançadas ao longo do projeto, o que facilitará alcançar objetivos finais como tolerância a falhas, visto sua estrutura formal. TypeScript foi a linguagem de programação escolhida.

2. Conceituação

O avanço na tecnologia de multiprocessadores, junto ao barateamento dos custos do hardware, ajudaram a popularizar o uso de máquinas multicore de alto poder de processamento e com maior capacidade de memória. Concomitantemente a isso, as redes de computadores tiveram um grande aumento no fluxo de dados, na velocidade de tráfego e no número de dispositivos conectados à rede, permeando cada vez mais o cotidiano de todos.

Com isso foi possível a criação e manutenção de sistemas computacionais maiores, que passaram a incluir grandes números de dispositivos, muitas vezes separados por grandes distâncias, conectados simultaneamente através das redes de computadores. Estes sistemas computacionais são conhecidos como sistemas distribuídos.

Para que estes sistemas possam atingir seus objetivos, eles precisam ser capazes de oferecer uma experiência transparente ao usuário, de forma que o usuário acredite que o sistema funcione como um só, sem precisar se preocupar com a complexidade existente nele. Além disso, um sistema distribuído também deve ser capaz de lidar com eventuais

falhas, com a heterogeneidade entre seus diversos componentes, com a segurança e confiabilidade do serviço, lidar com concorrência de acesso, bem como poder ser escalado para atender a maiores demandas no futuro.

2.1. Heterogeneidade (Middleware)

Em sistemas distribuídos, o Middleware é o software que provê serviços que permitem que os vários componentes de um sistema distribuído mantenham comunicação entre si, ainda que os componentes possuam hardware e sistemas operacionais heterogêneos. O middleware se encontra entre o sistema operacional da máquina e as aplicações que rodam nesse sistema.

Na aplicação distribuída desenvolvida neste trabalho, a heterogeneidade entre os nós de um sistema é alcançada através da conteinerização da aplicação utilizando-se Docker. A conteinerização em Docker permite a modularização do projeto, tornando fácil realizar quaisquer ajustes na aplicação e executá-la em qualquer máquina em que o docker esteja instalado.

A aplicação também utiliza o Express do Node.js, que é um framework de web e roteamento e que consiste essencialmente de uma série de requisições de funções middleware.

Funções de middleware irão pedir a requisição de um objeto, irão tratar a resposta e a próxima função necessária. Finalmente junto com o Express Router, o caminho das rotas será seguido e a camada de middleware tratada.

De modo geral a facilidade de inter-aplicação de comunicação, o gerenciamento de recursos, segurança e recuperação em falhas será prioritariamente realizado por este componente.

Vantagens para comunicação inter-aplicação fornecidas pelo Express:

- Serviços de segurança.
- Dados estatísticos.
- Mascaramento e recuperação de falhas.

2.2. Compartilhamento de recurso

Conectar diversos usuários diferentes, os quais, podem contribuir para o compartilhamento de possíveis buracos torna a troca de informação mais fácil entre a própria comunidade. De modo similar a aplicação pode ter seus recursos compartilhados por uma third party localizada em pontos geograficamente diferentes de onde se é criado uma nova contribuição, além de permitir que os recursos em disco por exemplo, ocorram independentemente da organização onde o aplicativo fora criado, assim, o local onde os dados são armazenados fica escondido do usuário final.

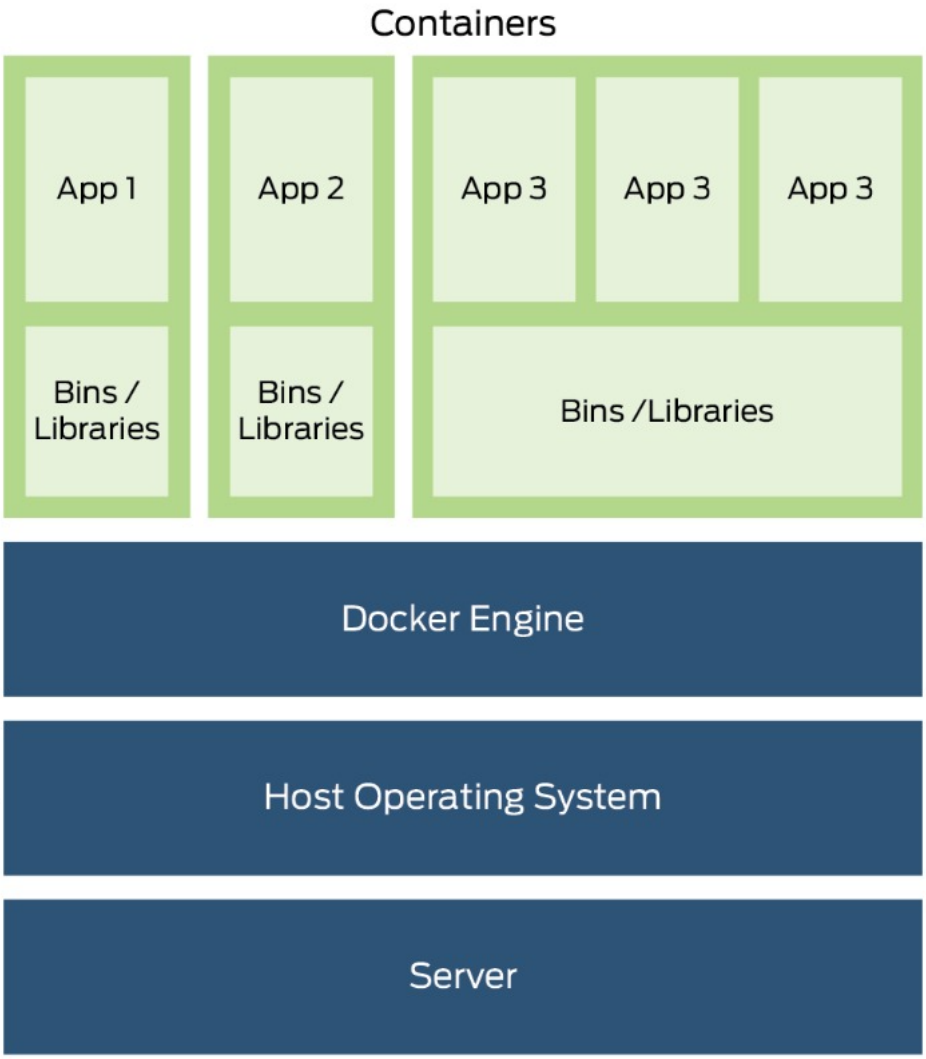

Figura 1. Diagrama exemplificando o funcionamento do Docker.

2.3. Ser aberto

Com os componentes seguindo padrões para descreverem a sintática e semântica do que cada fração irá precisar, neste caso através de protocolos e portas, é possível através dessa interface definir suas funções e permitir a usabilidade e integralidade de suas partes, com as devidas modificações. O banco pode, por exemplo, ser alterado com mais facilidade desde que respeite as requisições e respostas que passaram pelo middleware.

2.4. Escalabilidade

Divide-se em três principais dimensões:

- Tamanho, pode ser escalável a respeito de recursos como, maior número de nós adicionados sem perder a eficiência.
- Escalabilidade geográfica, usuários podem estar geograficamente distantes dos recursos utilizadas(um servidor na nuvem)mas o tempo de resposta dificilmente será notado conforme necessite de mais recursos

- Escalabilidade administrativa, mesmo que seja necessário criar mais organizações do sistemas, será facilmente gerenciado..

2.5. Tolerância a falhas

A principal técnica para tratamento de falhas é a redundância, conceito que é interligado com muitos dos princípios básicos para se adquirir tal tolerância a falhas. É significativo, assim, ter sistemas confiáveis. Os quais podem ter como teor sumário os seguintes tópicos:

- Disponibilidade, propriedade na qual o sistema pode ser usado imediatamente. Em uma visão mais ampla pode ser entendido junto com a capacidade de o sistema ter uma boa portabilidade.
- Confiança, ser executado continuamente sem apresentar falhas por um determinado tempo, preferivelmente de acordo com as especificações do usuário.
- Segurança, mesmo que o sistema não consiga, temporariamente, executar corretamente, nenhum evento catastrófico irá decorrer.
- Manutenibilidade, refere-se a facilidade de um sistema que apresentou falhas em ser reparado.

3. Aplicação

Visando um bem estar social, esta aplicação irá propor uma solução para alertar motoristas, pedestres e habitantes das mais diversas cidades quanto a qualidade de pavimentação das ruas, através de uma interface limpa e intuitiva os próprios cidadãos poderão enviar a localização geográfica de pontos onde se encontram potenciais problemas. Gestores municipais também poderão se orientador para garantir a segurança da população e se responsabilizar por pontos defasados.

3.1. Explicando o Projeto

O projeto consiste em um bot do Telegram que pega a localização do usuário. Esse bot é configurado por um arquivo em python. Ele pega essa localização e a armazena em um banco de dados MongoDB. Além disso existe uma aplicação web, dividida em backend e frontend. O backend se comunica com o banco e permite que outras aplicações façam requisições para esse banco na porta 3001, já o frontend é justamente a aplicação que vai fazer essas requisições, pegando assim as localizações e exibindo elas na porta 3000.

Dada a explicação anterior, todas essas quatro partes do projeto foram colocadas dentro de containers do Docker. Então temos quatro containers: backend, frontend, bot e db. Todos esses containers estão estruturados através de um docker compose que configura os seguintes nomes de serviços para eles: backend, frontend, bot e db. Preferimos dar os mesmos nomes de serviços aos nomes dados aos containers, para ficar mais fácil de ler a aplicação. Como estes containers são criados por um mesmo docker compose, eles estão na mesma rede, chamada projeto-final-default, e por tal motivo podem se comunicar uns com os outros, bastando saber o ip dos outros containers, o que não é uma boa ideia pois cada vez que um container é criado ele ganha um novo ip, porém também é possível se comunicar sabendo o nome de serviço dos containers. É através do nome de serviço que os containers se comunicam e passam dados uns para os outros nas portas anteriormente citadas.

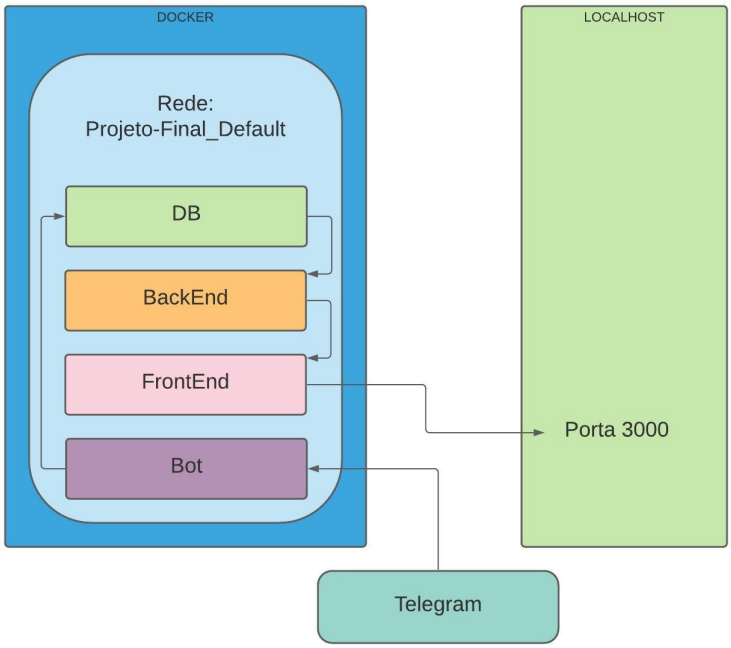

Figura 2. Diagrama da estrutura planejada da aplicação.

Funcionamento resumido:

O usuário inicia uma interação com o bot, que responde solicitando que o usuário envie sua geolocalização (através do clicar de um botão).

O bot então envia os dados recebidos para a porta 27017 do banco de dados.

O frontend requisita à porta 3001 do backend todas as localizações armazenadas no banco de dados.

O backend consulta o banco de dados através da porta 27017, e envia ao frontend, através da porta 3001, os dados recuperados.

O frontend então exibe os resultados com base nos dados recebidos.

No entanto houve um problema na hora de implementar o projeto, por algum motivo o frontend não conseguia fazer requisições ao backend através da porta 3001 deste, mesmo com a configuração estando correta. Presumimos ser algum problema de comunicação do nodejs com as portas internas da rede do docker. Por tal motivo, a porta interna do container do backend que recebe requisições foi redirecionada para se comunicar com a porta do 3001 do localhost e receber as requisições por lá.

A implementação final ficou assim:

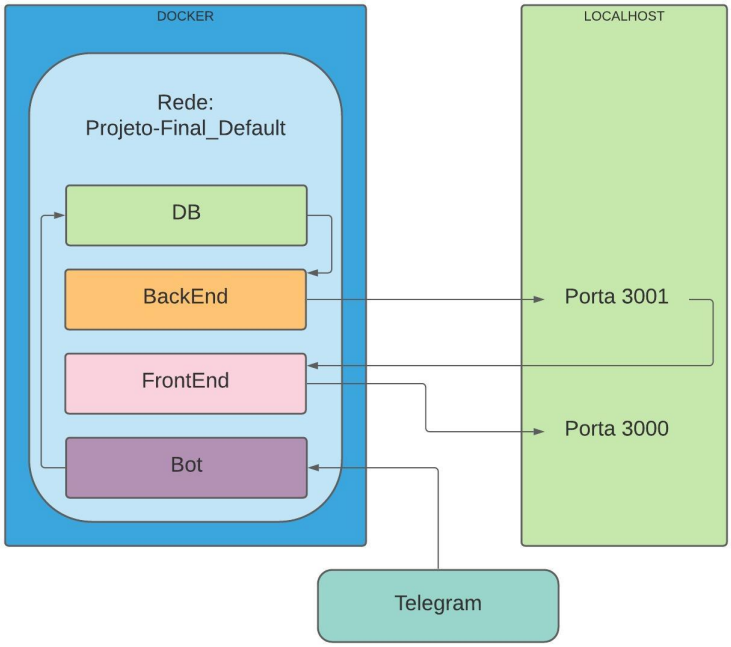

Figura 3. Diagrama da estrutura final da aplicação.

Funcionamento resumido:

O usuário inicia uma interação com o bot, que responde solicitando que o usuário envie sua geolocalização (através do clicar de um botão).

O bot então envia os dados recebidos para a porta 27017 do banco de dados.

O frontend requisita à porta 3001 do localhost, onde está configurada a porta 3001 do backend, todas as localizações armazenadas no banco de dados.

O backend consulta o banco de dados através da porta 27017, e envia ao frontend, através da porta 3001, a mesma do localhost, os dados recuperados.

O frontend então exibe os resultados com base nos dados recebidos.

4. Conclusão e Trabalhos Futuros

O desenvolvimento deste projeto atingiu o seu objetivo inicial de utilizar um chatbot baseado no Telegram para a captura em um banco de dados e exibição em uma interface web das localizações fornecidas ao bot pelos usuários. Tudo isso se valendo das vantagens presentes no Docker para a modularização, portabilidade e isolamento da aplicação, facilitando a implementação do projeto em máquinas heterogêneas.

A aplicação, em seu presente estado, pode ser utilizada de forma experimental e prática. No entanto, para a aplicação prática pode vir a exigir uma maior quantidade de dados fornecidos pelo usuário, como nome do denunciante, data de queixa, data de resolução (que seria atualizada uma vez que o denunciante constatasse que a prefeitura resolveu o problema), foto do problema denunciado, etc. Haveria também que se implementar uma forma de evitar falsas denúncias, ou denúncias duplicadas, o que poderia requerer um sistema de cadastro e login, bem como diferentes permissões de edição e visualização de denúncias.

A API do telegram também dá ao desenvolvedor muito mais ferramentas para criação de robôs mais complexos e completos. Sendo assim, a aplicação poderia até ser expandida para aceitar mais formas de relatos e denúncias referentes à cidade. O próprio bot poderia ser programado de forma a aceitar mais interações do usuário e conversar de forma mais humana, respondendo às conversas mais naturalmente e não apenas em forma de comandos.

5. Instruções de Execução

5.1. Executando a aplicação

ATENÇÃO: modifique o arquivo do bot.py para se conectar com o token do seu bot.

ATENÇÃO: os passos 4, 5, e 6 podem ser bem demorados, vai depender do seu sistema (no linux é mais rápido que no windows, pois o windows roda uma pequena vm do linux para poder então usar o docker) e da potência da sua máquina. Então se demorar, seja paciente e espere, não saia fechando as coisas.

PASSOS

1. mude o arquivo bot.py dentro da pasta bot, coloque nele o token do seu bot
2. certifique-se de ter o docker instalado e rodando na sua máquina.
3. abra o terminal na pasta no projeto
4. use o comando a seguir para criar as images: docker-compose build
5. use o comando a seguir para criar e rodar os containers: docker-compose up -d
6. no seu navegador, acesse: localhost:3000
7. no endereço acima você verá o site com o mapa e os pontos onde estão os buracos marcados. Se não apareceu nenhum ponto para você, deve ser porque você ainda não enviou sua localização no bot.

5.2. Utilizando o bot

ATENÇÃO: não adianta usar o bot no telegram do computador, pois ele não permite enviar a localização, então use pelo celular

1. no seu celular, instale o telegram
2. deixe seu gps ativado
3. abra o telegram, depois abra o chat com o seu bot (que ja deve ter sido criado previamente).
4. digite o seguinte comando para iniciar a conversa com o bot: /start
5. siga as instruções no menu que aparecerem e envie sua localização.