Relatório Técnico

Etapa do projeto: Relatório Final

GRUPO BROKER - TIME 3 GRUPO 19

Integrantes

André Niero Setti, N° 10883901
Enrique Gabriel da Silva Teles, N° 10724326
Mathias Fernandes Duarte, N° 10734352
Tamiris Fernandes Tinelli, N° 10346738

https://github.com/Setti7/iot-2020-broker

Introdução

Nosso grupo realizou a instalação do broker eclipse-mosquito na máquina virtual tau04-vm1, permitindo que a aplicação se conecte com o micro-serviço.

A instalação foi realizada no host *andromeda.lasdpc.icmc.usp.br*, na porta *8041* (para protocolo mqtt), usando o usuário *ssc952-t3.*

O broker foi testado e configurado para atender aos requisitos dos dois grupos com os quais tivemos contato direto, o da aplicação e o do microserviço. Assim, tivemos certeza que nossa solução estava funcional para o time como um todo.

Foi configurado:

1. Persistência, para que nenhum dado seja perdido caso seja apresentado algum problema na execução do broker.
2. Logs, para conseguirmos ter uma visão de erros e avisos dados pelo serviço.
3. Conexão por websockets, pois a maioria dos navegadores não conseguem se conectar através do protocolo mqtt.
4. Conexão por mqtt, para que os dispositivos possam se conectar ao broker se aproveitando das vantagens deste protocolo.

Tecnologias

O servidor mosquitto está configurado usando docker-compose. Foi escolhido essa tecnologia pois alguns integrantes do nosso time já possuíam experiência com docker, além de ter as seguintes vantagens:

1. **REINÍCIO AUTOMÁTICO:** docker-compose permite muito facilmente configurar o serviço para se reiniciar sozinho caso a máquina virtual seja desligada. Isso é alcançado usando a linha *restart: always* na configuração do *docker-compose.yaml*.

2. **PORTABILIDADE:** como a proposta do docker é funcionar indiferente do host, podemos testar e configurar o servidor mosquitto localmente e assim que estivermos satisfeitos, apenas enviamos o único arquivo de configuração para a máquina virtual e ele terá exatamente o mesmo ambiente.

3. **SEGURANÇA:** por ser isolado do resto do sistema, o docker é muito seguro, pois um atacante que conseguir acesso ao container do mosquitto, não conseguirá sair dele, e assim, ter controle da máquina virtual inteira.

4. **CONFIANÇA:** também por ser isolado do sistema, o docker nos traz a confiança que o serviço está bem configurado e que ele está funcionando por motivos claros. Por exemplo: uma pessoa instala um serviço que depende de uma biblioteca em sua máquina local e desenvolve sua solução ali, porém ela não sabe sobre essa dependência. Quando for realizada essa instalação em outro ambiente ela poderá ter vários problemas que não apareceram antes, como dependências não instaladas corretamente, erro de compatibilidade entre as dependências, variáveis do ambiente que sobraram de outra instalação que estavam permitindo uma funcionalidade inesperada, entre outros. Com o docker isso não acontece, e como consequência direta nós conseguimos ter uma visão melhor de como o serviço está se comportando.

Configuração Atual

Na nossa instalação, usamos a imagem oficial do mosquitto (*eclipse-mosquitto:1.6.12*) e roteamos a porta da máquina virtual 1841 para a porta 1883 do container, que é a porta padrão do mosquitto. Fizemos isso ao invés de configurar o mosquitto para usar a porta 1841 pois a imagem apresenta alguns bugs de permissão. Dessa forma, lendo algumas *issues* no *github* que resolveram problemas semelhantes ao nosso, decidimos mudar a menor quantidade de configurações do mosquitto em si, e delegar ao docker o que for possível dessa parte de networking.

Além disso, ativamos a persistência para a pasta *~/mosquitto/data* e os logs de acesso em *~/mosquitto/logs*.

Testamos o servidor e ele pode ser acessado através da internet na porta 8041 no host *andromeda.lasdpc.icmc.usp.br*, ou através da rede local pela porta 1841, usando o protocolo MQTT. Já a comunicação com websocket foi configurado para a porta local 9001.
Toda a configuração do ambiente se encontra em *docker-compose.yml*, sendo o arquivo de configuração do mosquito em *~/mosquitto/config/mosquitto.conf*. A pasta *~/mosquitto* contém 3 volumes do docker (*/config*, */log* e */data*), que são mapeados para dentro do container, permitindo a persistência dos dados e um fácil acesso aos logs.

Problemas Encontrados

Durante a configuração e instalação do mosquito encontramos uma grande dificuldade ao configurar um volume para o serviço utilizar corretamente, ao configurarmos a persistência de dados e o log do serviço.

De acordo com o exemplo dado na descrição da imagem (*eclipse-mosquitto:1.6.12*), devemos mapear o arquivo de configuração do mosquitto (*mosquitto.conf*) para o caminho absoluto */mosquitto/config/mosquitto.conf* do container. Além disso, o mosquitto pode ser configurado para persistir dados e escrever o log em um arquivo do disco dado que seja montado mais dois volumes (*/mosquitto/data* e */mosquitto/log*, respectivamente).

Como todos esses arquivo são filhos de */mosquitto* pensamos em simplificar e criar apenas um volume: criamos uma pasta local com a mesma hierarquia que o mosquitto espera (com */data*, */log* e */config*) e mapeamos ela para a pasta */mosquitto* do volume do container, porém isso não funcionou. O mosquitto mesmo tendo acesso ao volume não conseguia ler e escrever os dados corretamente.

Depois de alguma pesquisa, descobrimos que era necessário 3 volumes diferentes, um para cada pasta filha de */mosquitto* (*/config*, */data* e */log*). Ao testarmos essa solução verificamos que o problema anterior foi resolvido, mas surgiu outro: o mosquitto recebia o erro de falta de permissão ao abrir o arquivo de log.

Após procurar por ajuda na comunidade, descobrimos que a imagem usa um *uid:guid* fixo para o user *mosquitto* da imagem que é 1883:1883, então foi necessário atualizar as permissões das pastas dos volumes para esse *uid:guid* (executando “sudo chown -R 1883:1883 mosquitto/”), além de configurar o docker-compose para utilizar esse usuário. Feito isso, a instalação funcionou sem mais problemas.

Testes Realizados

Para nos certificarmos que nosso serviço funciona corretamente, realizamos uma série de testes ao decorrer de nossas instalações.

Após a primeira instalação do broker, testamos localmente se ele estava funcionando. Para isso, utilizamos os comando ‘*mosquitto_sub -t "test" -p 1841*’ em um terminal e ‘*mosquitto_pub -m "message from mosquitto_pub client" -t "test" -p 1841*’ em outro. Ao verificar que a mensagem aparece no terminal que está inscrito neste tópico, concluímos que o broker foi instalado corretamente e estava acessível localmente pelo protocolo mqtt. Também realizamos

outros testes mudando os tópicos e utilizando wildcards na inscrição e, como esperado, tudo funcionou sem problemas.

Também testamos o acesso ao broker pela internet, para verificar que o roteamento de portas estava correto. Analogamente ao teste anterior, usamos o comando '*mosquitto_sub -h andromeda.lasdpc.icmc.usp.br -p 8041 -t "test"'* para nos inscrever ao tópico "*test"* e, em outro terminal executamos '*mosquitto_pub -h andromeda.lasdpc.icmc.usp.br -p 8041 -t "test" -m "test over internet" -q 2'* para enviarmos uma mensagem ao mesmo tópico. Como foi visto que a mensagem foi corretamente publicada e recebida pela nossa inscrição, confirmamos que o roteamento de portas estava correto e, portanto, que nosso broker estava acessível também pela internet como planejado.

Em seguida, o grupo de aplicação requisitou acesso ao broker por websockets, dado que a maioria dos navegadores não aceitam conexões por mqtt. Assim, após configurarmos o broker para esse caso de uso, também o testamos.

Porém, não possuímos uma ferramenta para testar a comunicação por websockets, então criamos um script para isso. Tal script é o "test_websockets.py". Ele se inscreve em todos os tópicos do broker e manda para o console as mensagens recebidas, permitindo que nós verifiquemos que a nossa configuração do websockets também estava funcionando.

Para executar esse script, basta antes instalar a dependência *paho-mqtt* pelo pip (com *pip3 install paho-mqtt*) e em seguida executá-lo: *python3 test_websocket.py*.

Execução

Por termos usado o docker-compose, basta usar o comando "docker-compose up" para inicializar o serviço (com o argumento -d caso queira executá-lo em plano de fundo). Em um sistema novo, também é necessário dar as permissões corretas para a pasta *mosquitto/* como ressaltado na seção "Problemas Encontrados", que pode ser corrigido ao usar o comando "sudo chown -R 1883:1883 mosquitto/".

Para verificar se o broker está em execução pode-se checar o status do container com "docker-compose ps" e para parar a execução do mesmo, basta "docker-compose down".

Referências

https://docs.docker.com/
https://docs.docker.com/compose/

https://hub.docker.com/_/eclipse-mosquitto
https://mosquitto.org/man/mosquitto-conf-5.html
https://github.com/eclipse/mosquitto/issues/1078