



MQTT

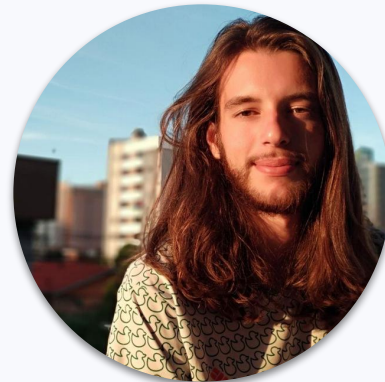
*O padrão de comunicação em IoT e
impacto no seu dia a dia*

Gabriel Prando

Engenheiro de software tribo de Plataforma na **Conta Simples**

Quase Engenheiro de computação pela UTFPR

Entusiasta Node, Go, banco de dados e tech no geral



O que veremos hoje?

- **Introdução**
- **MQTT e universo IoT**
- **Histórico e propósito do MQTT**
- **Importância no seu dia a dia**
- **Arquitetura do protocolo**
- **Mensagem, Tópicos, QoS**
- **Casos de uso**
- **Exemplo**

Introdução

MQTT e universo IoT

- **IoT é definido por uma rede interconectada**
 - Dispositivos físicos, softwares, etc.
- **Área de IoT passa por um crescente na última década**
 - Estimativa de 50 bilhões de dispositivos interconectados em 2023 e mais de 75 bi nos próximos
- **Comum a troca de mensagens entre aplicações**

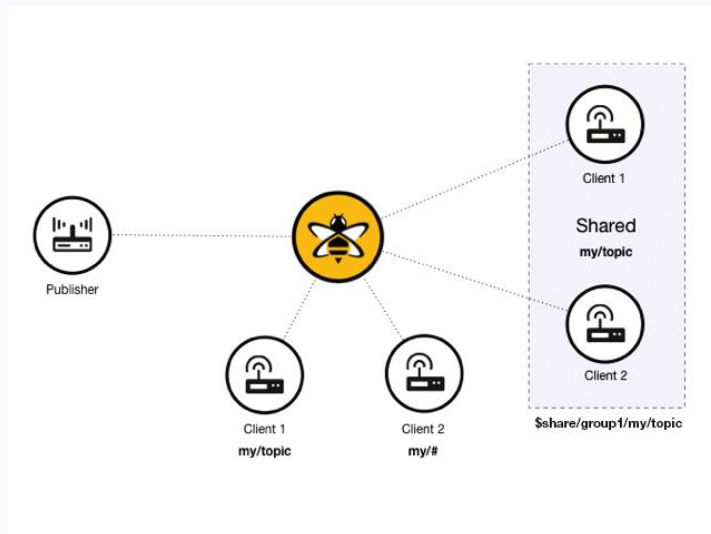
Histórico e propósito do MQTT

- **Criado em 1999 por Andy Stanford-Clark (IBM)**
- **Projetado como um transporte de mensagens de publicação/assinatura extremamente leve**
 - Dispositivos físicos, softwares, etc.
 - Ideal para conectar dispositivos remotos com um pequeno espaço de código e largura de banda de rede mínima
- **Comunicação bidirecional**
- **Escalabilidade**

Arquitetura

Arquitetura

- Utiliza o padrão de publicação e assinatura (pub/sub)



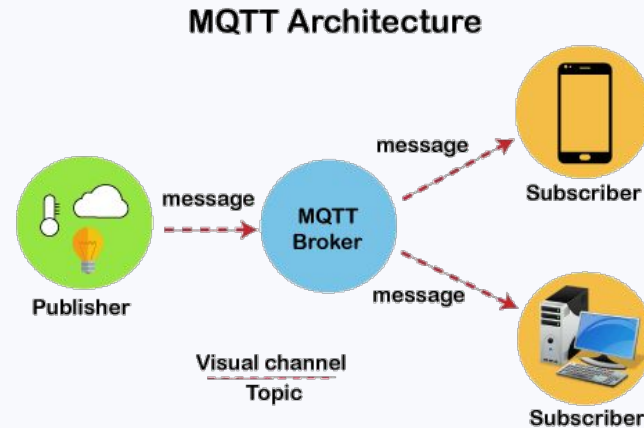
Arquitetura

- **TCP/IP para conexão entre cliente (pub ou sub) e broker**
- **Pode utilizar ou não, autenticação e criptografia**

Arquitetura

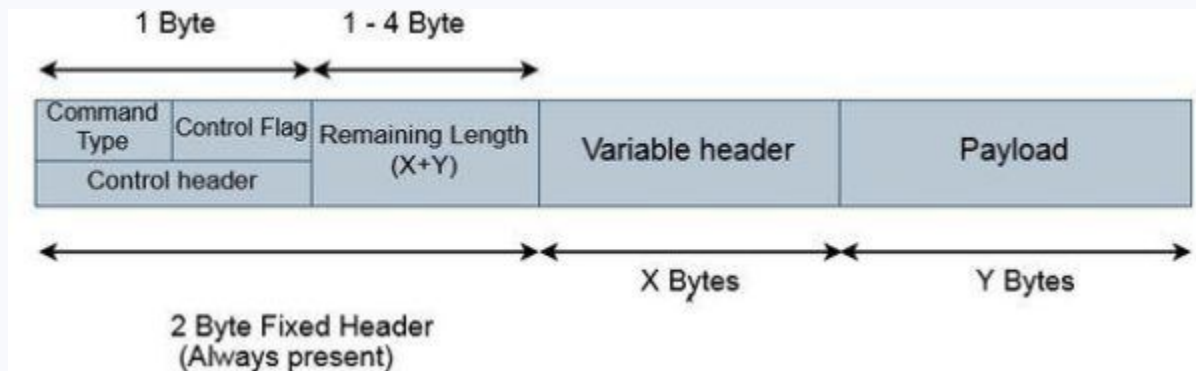
- **Se observa alguns componentes principais:**

- Mensagem
- Broker
- Client
- Topic
- Publisher
- Subscriber
- QoS



Mensagem

- Cabeçalho fixo de 2 bytes + cabeçalho variável + payload

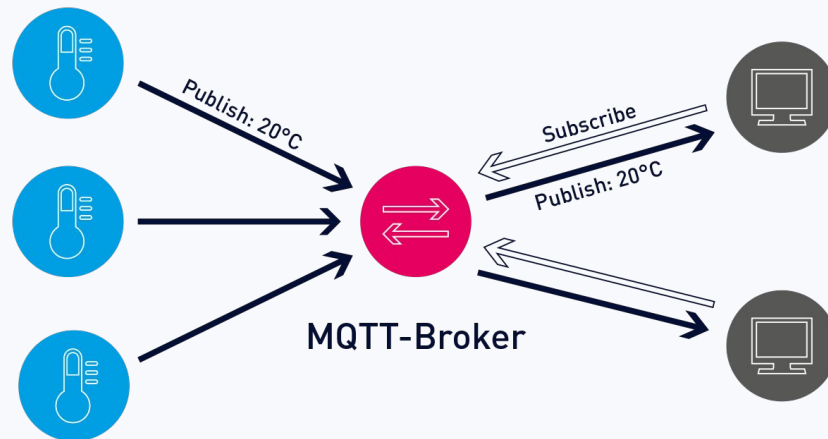


Tipos de Mensage

Valor	Nome	Direção	Descrição
0	Reservado	Proibido	Reservado
1	CONNECT	Cliente para Servidor	Requisição de cliente
2	CONNACK	Servidor para o Cliente	Reconhecimento de conexão
3	PUBLISH	Cliente para o Servidor	
4	PUBACK	Cliente para o Servidor	
5	PUBREC	Publicação recebida	Publicação recebida
6	PUBREL	Cliente para Servidor	
7	PUBCOMP	Cliente para Servidor	
8	SUBSCRIBE	Cliente para Servidor	Pedido de inscrição
9	SUBACK	Servidor para cliente	Reconhecimento de inscrição
10	UNSUBSCRIBE	Cliente para Servidor	Pedido de desinscrição
11	UNSUBACK	Servidor para cliente	Reconhecimento desinscrição
12	PINGREQ	Requisição	Requisição PING
13	PINGRESP	Servidor para cliente	Resposta PING
14	DISCONNECT	Cliente para Servidor	Cliente esta desconectado
15	Reservado	Proibido	Reservado

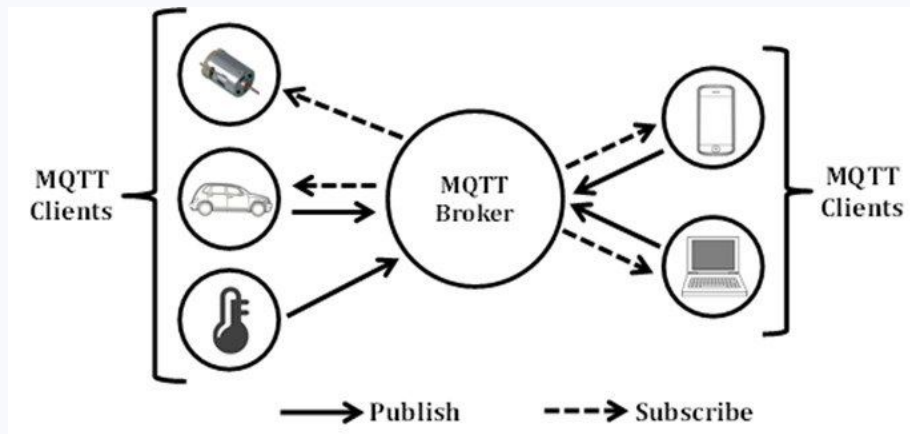
Broker

Distribui as informações aos clientes interessados conectados ao servidor



Client

Dispositivo que se conecta ao broker para enviar ou receber informações



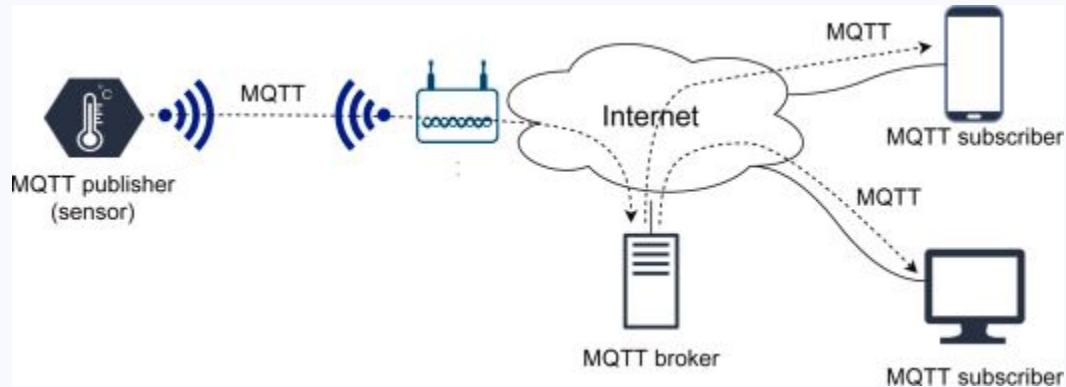
Topic

Os clientes publicam, assinam ou fazem as duas coisas em um tópico

topic level
separator
↓
myhome / groundfloor / livingroom / temperature
└───┬───┘ └───┬───┘
topic level topic level

Publisher

Clientes que enviam informações ao broker para distribuir aos clientes interessados com base no nome do tópico



Subscriber

Clientes que informam ao broker em quais tópicos estão interessados e passam a receber as informações relacionadas



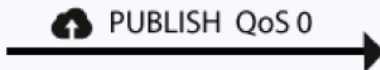
- Cada processo de conexão estabelece um nível de qualidade de serviço desejado:
 - QoS 0
 - QoS 1
 - QoS 2

QoS 0 - No máximo 1 vez

- Semelhante ao UDP, sem confirmação
- Remetente não armazena mensagens para retransmissões futuras



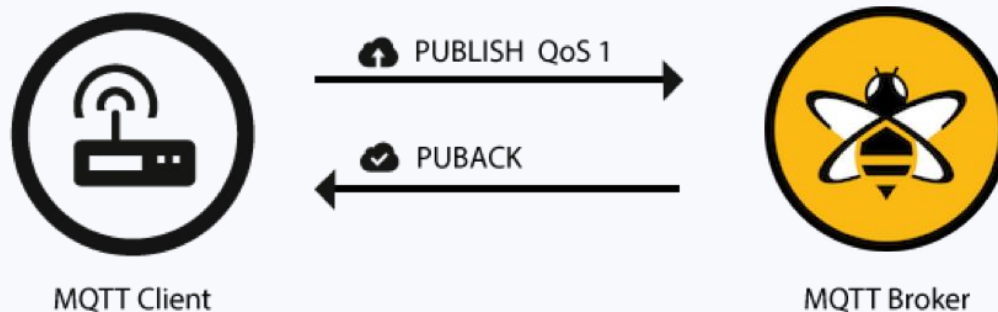
MQTT Client



MQTT Broker

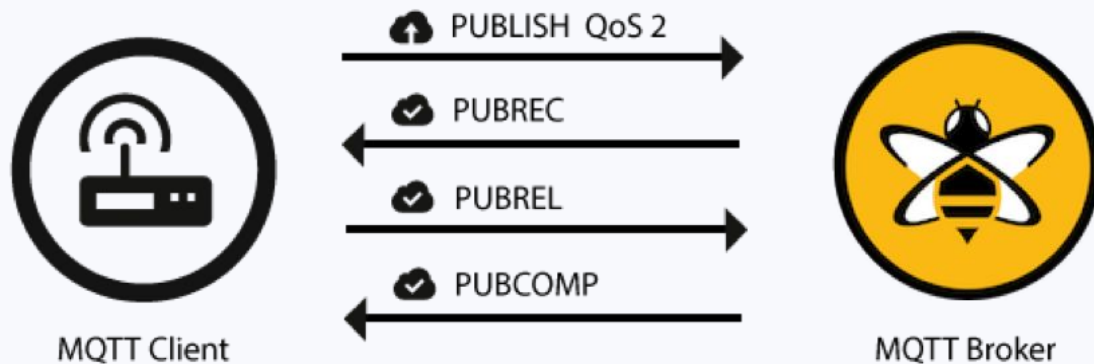
QoS 1- Pelo menos uma vez

- **Há confirmação de entrega**
 - Remetente pode enviar várias mensagens análogas
 - Pode ocorrer atraso no recebimento de feedback
- **Mensagem é armazenada pelo remetente até a confirmação**



QoS 2 - Exatamente uma vez

- **Garante que será entregue 1 única vez**
- **Envio de confirmações de recebimento e confirmações de recebimento de confirmações de recebimento (confirmação bidirecional)**





- **Não existe um QoS melhor ou pior, tudo depende do caso de uso da aplicação, recursos disponíveis, etc**
- **O nível de QoS é acordado entre client e broker, logo em um mesmo tópico podemos ter níveis de QoS heterogêneos**

MQTT vs HTTP

MQTT	HTTP
Publicação / assinatura	Solicitação / resposta
Tem menos complexidade	É mais complexo
O tamanho da mensagem gerada é menor, pois usa o formato binário	O tamanho da mensagem gerada é mais porque usa o formato ASCII
Cabeçalho de 2 bytes	Cabeçalho de 8 bytes
Porta default 1883	Portas 80 ou 8080
Fornecer segurança de dados com SSL / TLS	Não fornece segurança, mas o HTTPS foi desenvolvido para isso

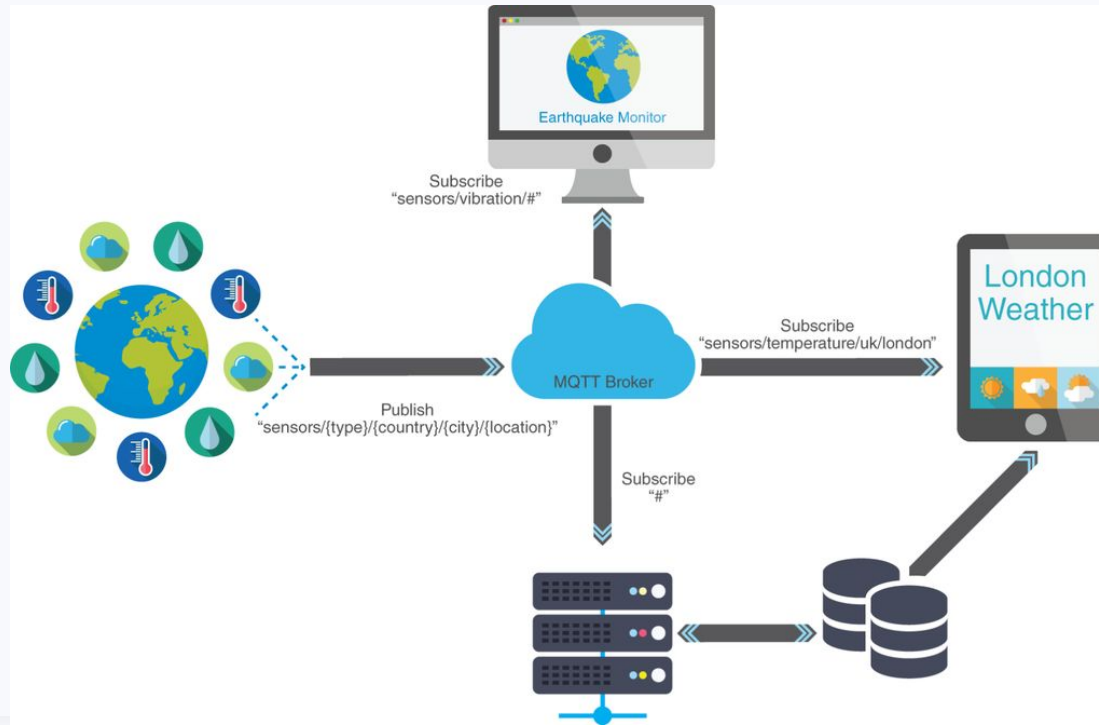
Mas afinal o que podemos fazer
com MQTT?



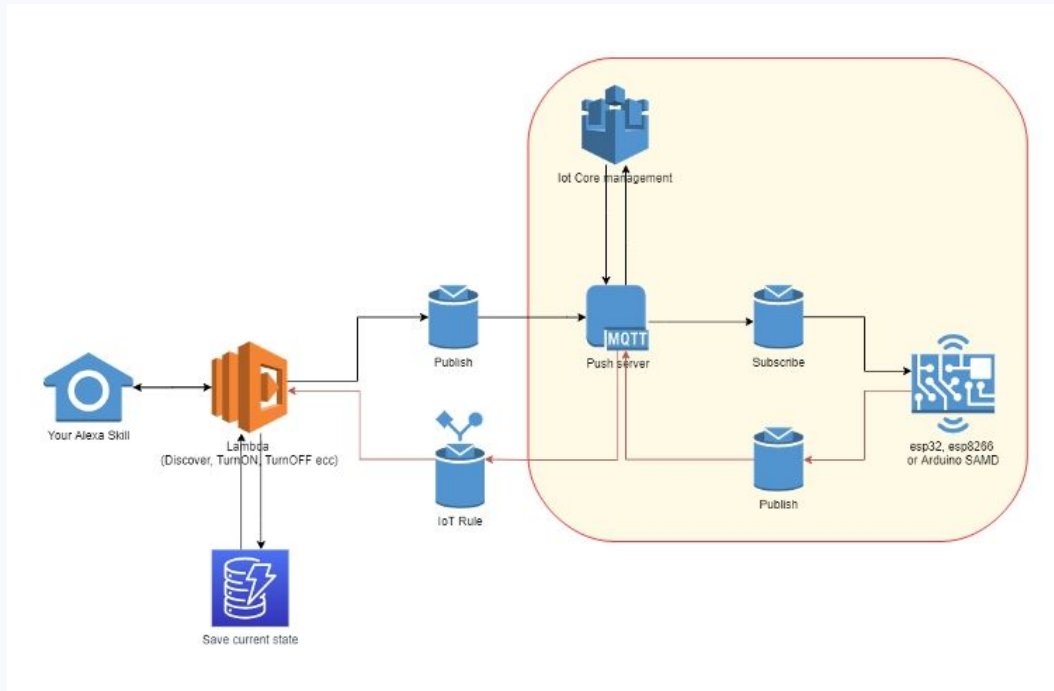
Casos de uso

- Setor automotivo, logístico, fabricação
- Casas inteligentes
- Indústria petrolífera
- Transporte
- Microserviços em geral
- O céu é o limite

Casos de uso



Casos de uso



Exemplo

E se a gente fizesse um
monitor para nosso sistema?

Terminei de assistir a vídeo
aula de programação



Errei tudo

Docker para facilitar a vida

```
1  services:
2    emqx1:
3      image: emqx/emqx:5.0.4
4      ...
5
6    elasticsearch:
7      build:
8        context: elasticsearch/
9
10     ports:
11       - "9200:9200"
12       - "9300:9300"
13     ...
14   kibana:
15     build:
16       context: kibana/
17     ports:
18       - "5601:5601"
19     ...
```

Coletar infos do sistema com node 16

```
1 import os from "node:os";
2
3 export const getSystemData = () => ({
4   loadavg: os.loadavg(),
5   freemem: os.freemem(),
6   cpus: os.cpus()[0].speed,
7   cpuUsage: process.cpuUsage(),
8   memoryUsage: process.memoryUsage()
9 });
```

Configs para MQTT

```
1 export class MqttProvider {  
2   private connectUrl = `mqtt://broker.emqx.io:1883`;  
3  
4   private client = mqtt.connect(this.connectUrl, {  
5     clientId: `mqtt_${Math.random().toString(16).slice(3)}`,  
6     clean: true,  
7     connectTimeout: 4000,  
8     username: "emqx",  
9     password: "public",  
10    reconnectPeriod: 1000,  
11  });
```

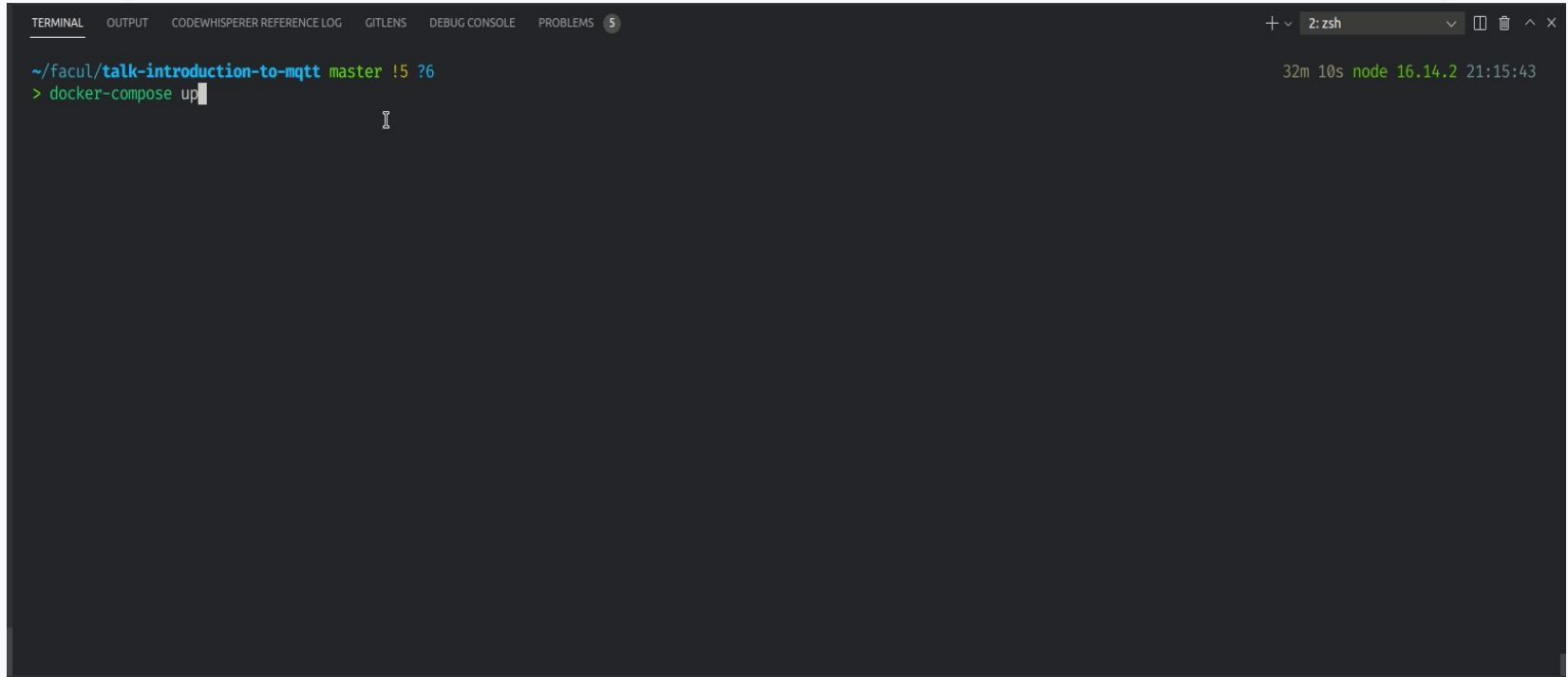

Publicar infos no tópico

```
1  async publishSystemMetrics(): Promise<void> {
2    this.client.on("connect", async () => {
3      console.log("Connected");
4      while (1) {
5        const message = getSystemData();
6        await this.client.publish(
7          this.topic,
8          JSON.stringify({ ...message, date: new Date()}),
9          { qos: 0, retain: false },
10         (error) => {
11           if (error) {
12             console.error(error);
13           }
14         }
15       );
16       await waitSeconds();
17     }
18     this.client.end(true);
19   });
20 }
```

Receber e
enviar para
elastic

```
1  async subscribeTopic() {
2    this.client.on("connect", () => {
3      console.log("Connected");
4
5      this.client = this.client.subscribe([this.topic], () => {
6        console.log(`Subscribe to topic '${this.topic}'`);
7      });
8
9      this.client.on("message", (topic, payload) => {
10        const message = JSON.parse(payload.toString());
11        console.log("Received Message:", topic, message);
12
13        elasticClient.index({
14          index: 'system-stats',
15          body: message,
16          type: "json"
17        })
18      });
19    });
20
21  }
```

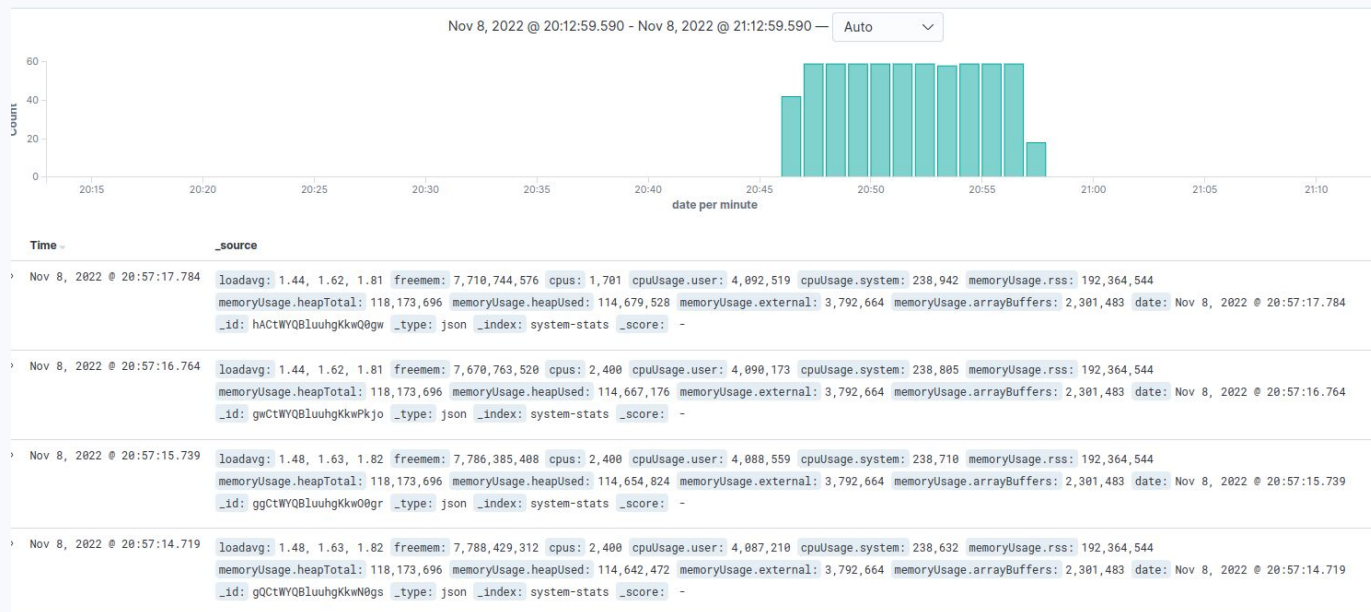
Pub/Sub



A screenshot of a Visual Studio Code terminal window. The terminal has a dark background with light green text. The top bar shows tabs for 'TERMINAL', 'OUTPUT', 'CODEWHISPERER REFERENCE LOG', 'GIT LENS', 'DEBUG CONSOLE', and 'PROBLEMS' (with a '5' icon). The terminal prompt is '~/.facul/talk-introduction-to-mqtt master !5 ?6'. The command '> docker-compose up' is entered, with the cursor at the end. The status bar at the bottom right shows '32m 10s node 16.14.2 21:15:43'.

```
~/facul/talk-introduction-to-mqtt master !5 ?6  
> docker-compose up
```

Visualizando no kibana



Referências

Introduction to MQTT. SparkFun Learn. Disponível em:

<<https://learn.sparkfun.com/tutorials/introduction-to-mqtt/all>>. Acesso em: 7 Nov. 2022.

ESUG. **MQTT.** SlideShare iOS. Disponível em: <<https://www.slideshare.net/esug/mqtt-186206125>>.

Acesso em: 7 Nov. 2022.

SILVA JUNIOR, M. P. d. Análise entre protocolos HTTP e MQTT em projetos IOT. Nov. 2021.

Monografia (TCC) – Universidade Tecnológica Federal do Paraná.



E por hoje é só

Gabriel Prando - Software Engineer (gabrie.prando@contasimples.com)
Cloud Core | SMC - Plataforma