

Arquitetura e Desenho de Software



Cap 05

Padrões Arquiteturais

Message Oriented Middleware

Professor:

Vitor Figueiredo

Ano / Semestre:

2023 / 2

Ministrada em:

04-dez

Principais conceitos

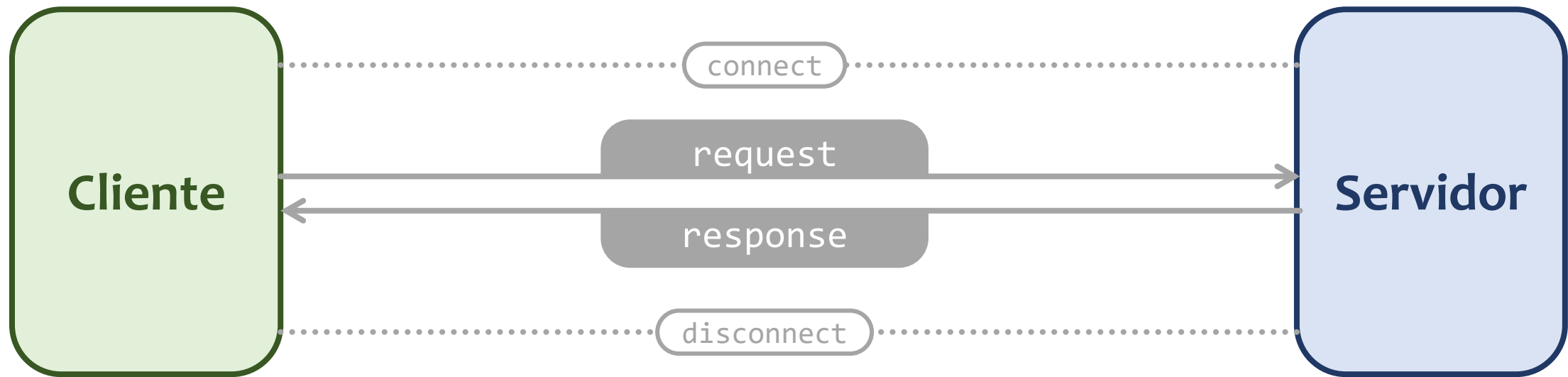
Protocolo MQTT

Broker Mosquitto

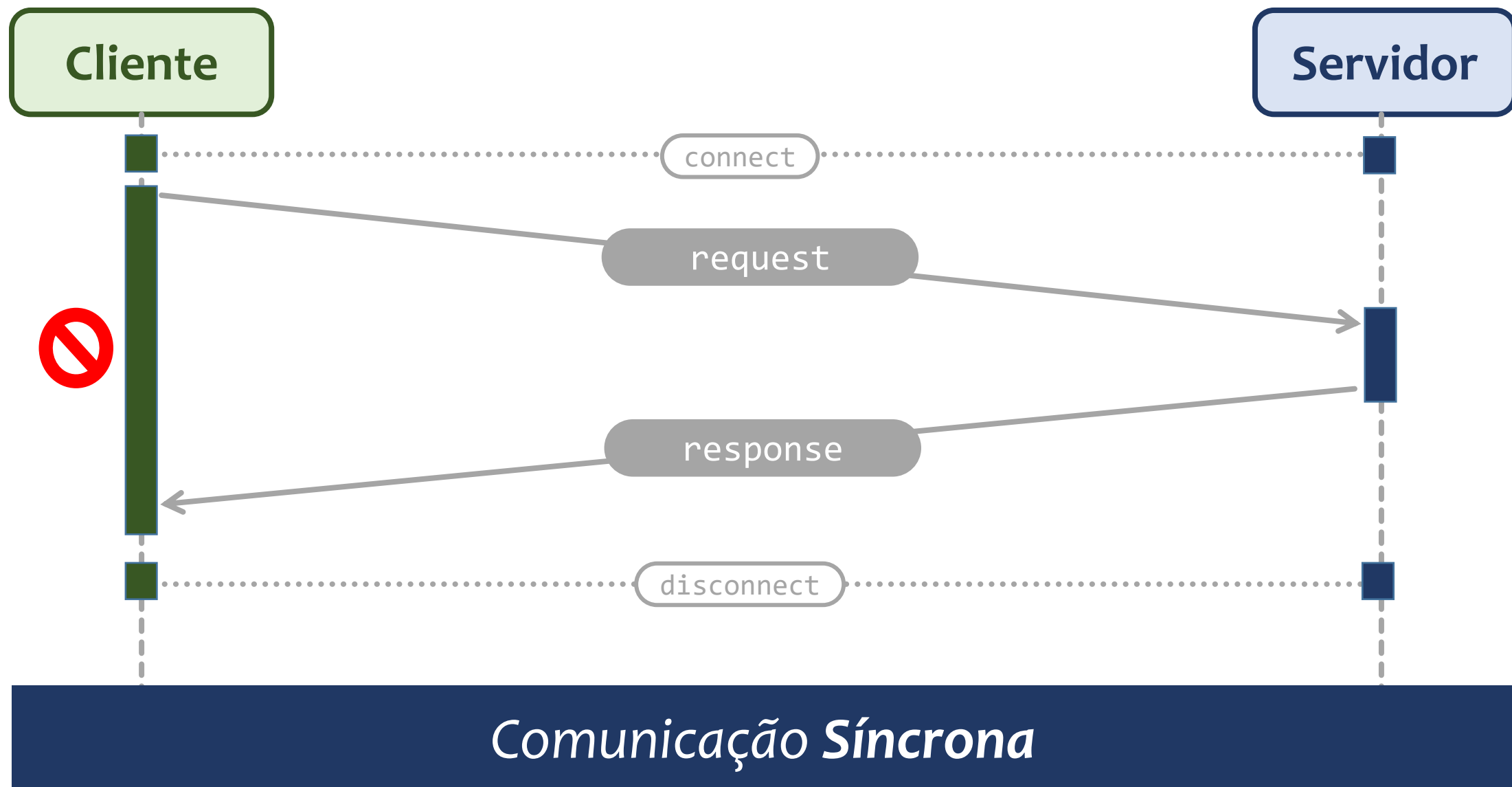
Eclipse Paho

Dashboard com Angular

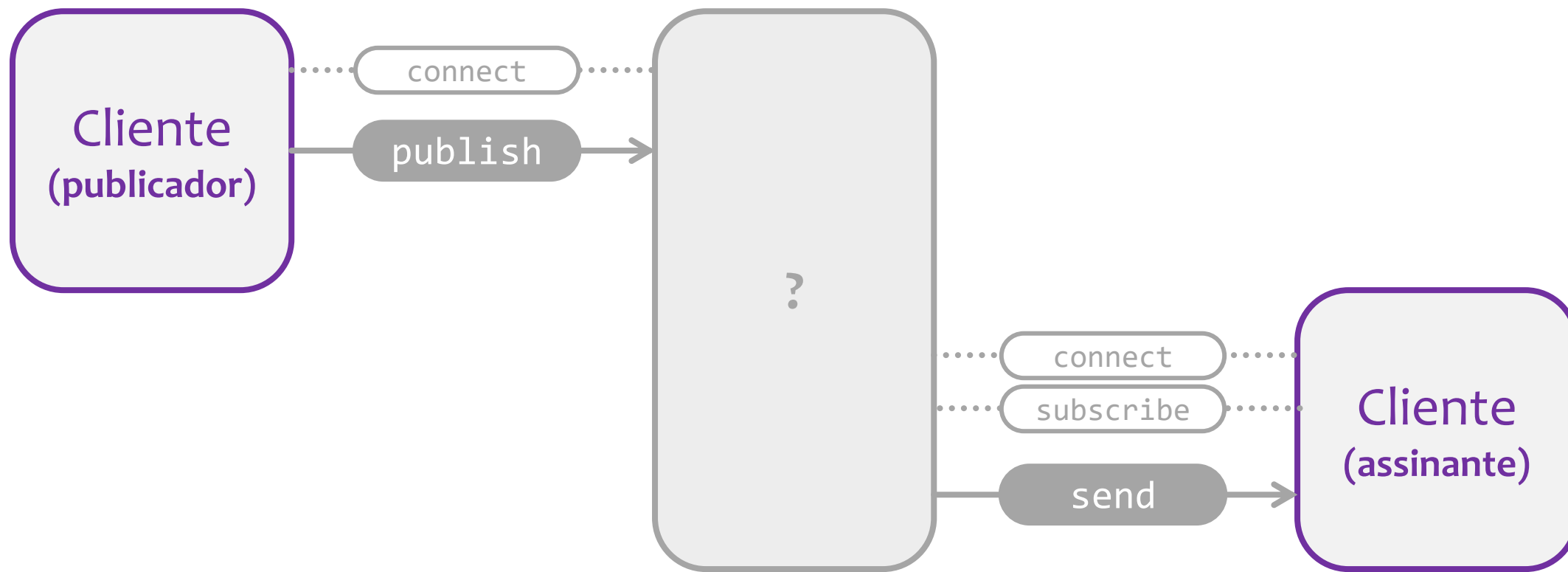
Modelos de integração: **Cliente/Servidor**



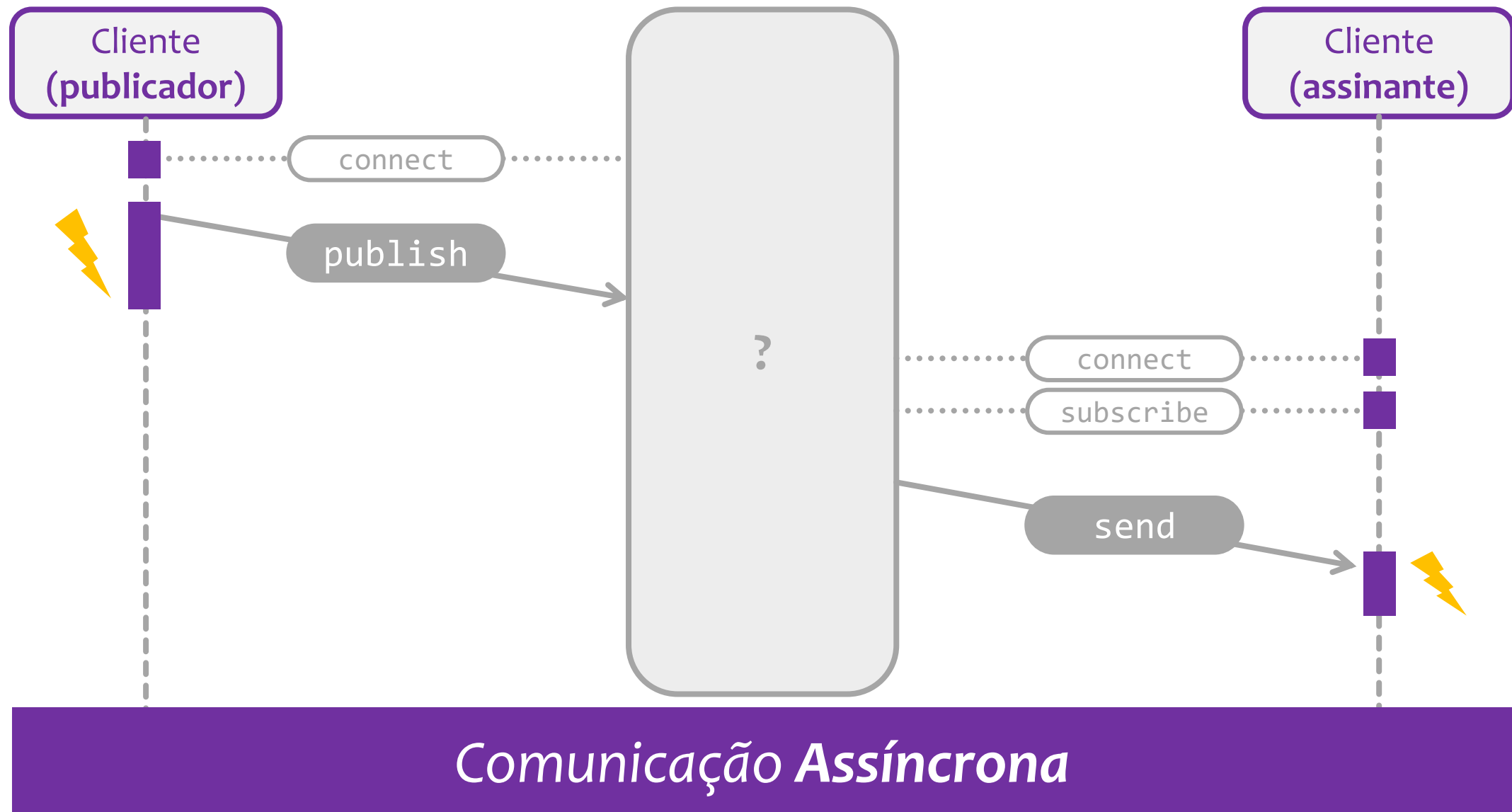
Modelos de integração: Cliente/Servidor



Modelos de integração: Publisher / Subscriber



Modelos de integração: Publisher / Subscriber



O que **Middleware Orientado a Mensagem**?

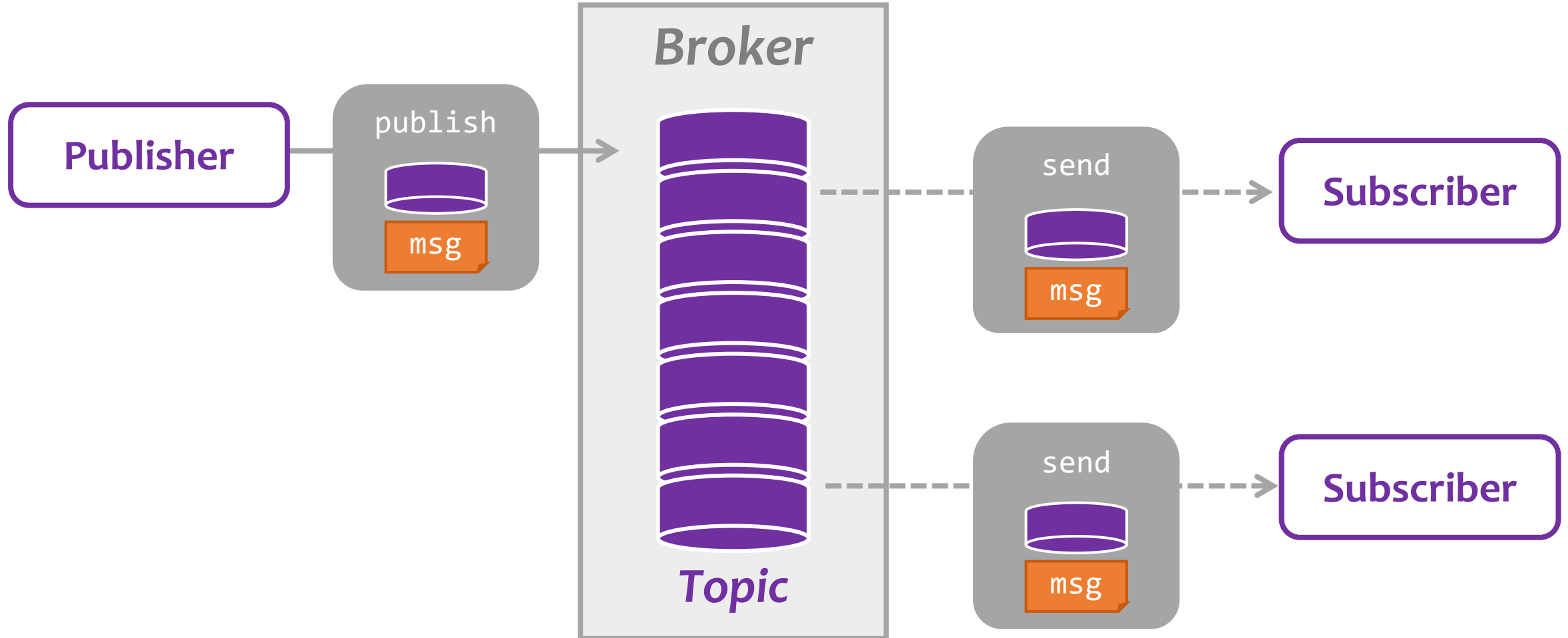
→ Padrão Arquitetural para possibilitar a **Comunicação Assíncrona**

→ Elemento '?' é chamado **Broker**

→ **Topic** é uma estrutura lógica para agrupar mensagens

→ Podem haver vários **publicadores** e também vários **assinantes**

MOM



Soluções MOM



Protocolo MQTT

Existem inúmeros protocolos de MOM

Message Queue Telemetry Transport (MQTT) é um dos mais famosos

Muitos projetos **IoT** têm utilizado o MQTT

Amplamente suportado por bibliotecas e linguagens

Protocolo MQTT

Criado pela IBM em 1999 para monitorar **sistemas industriais**

Pontos fortes:

- **Eficiência em banda de rede**
- **Eficiência energética (baterias)**
- **Extremamente leve**

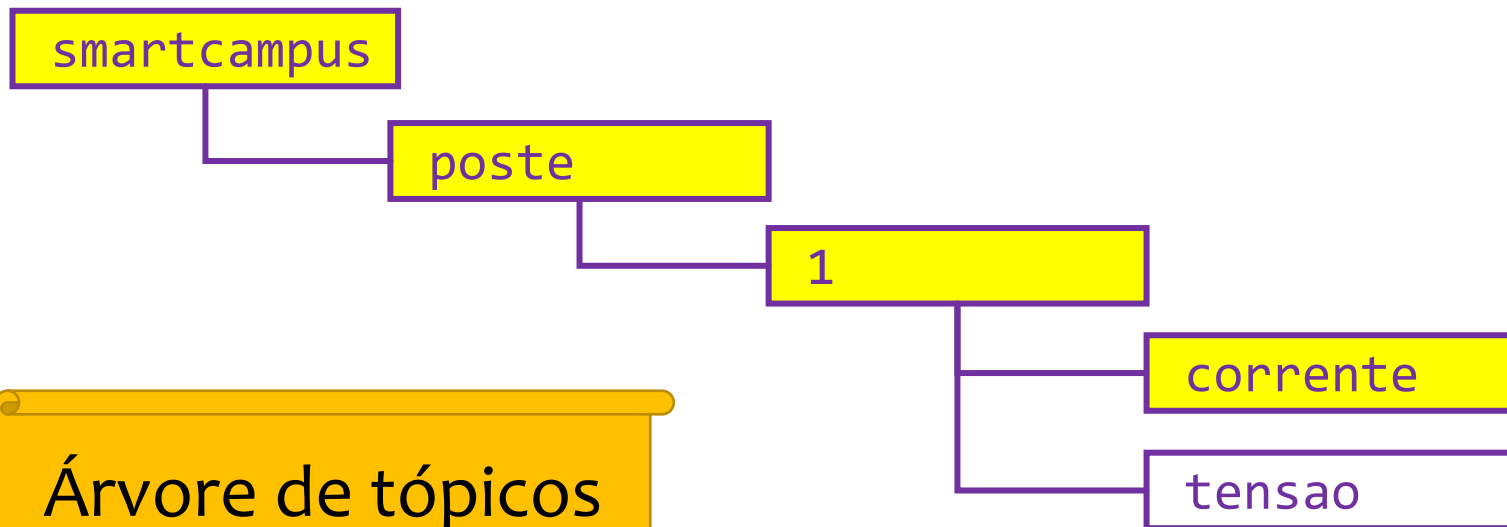
Versões do MQTT

1.0	1999
3.1	2013
3.1.1	2014
5.0	2019

Nomeação de Tópicos

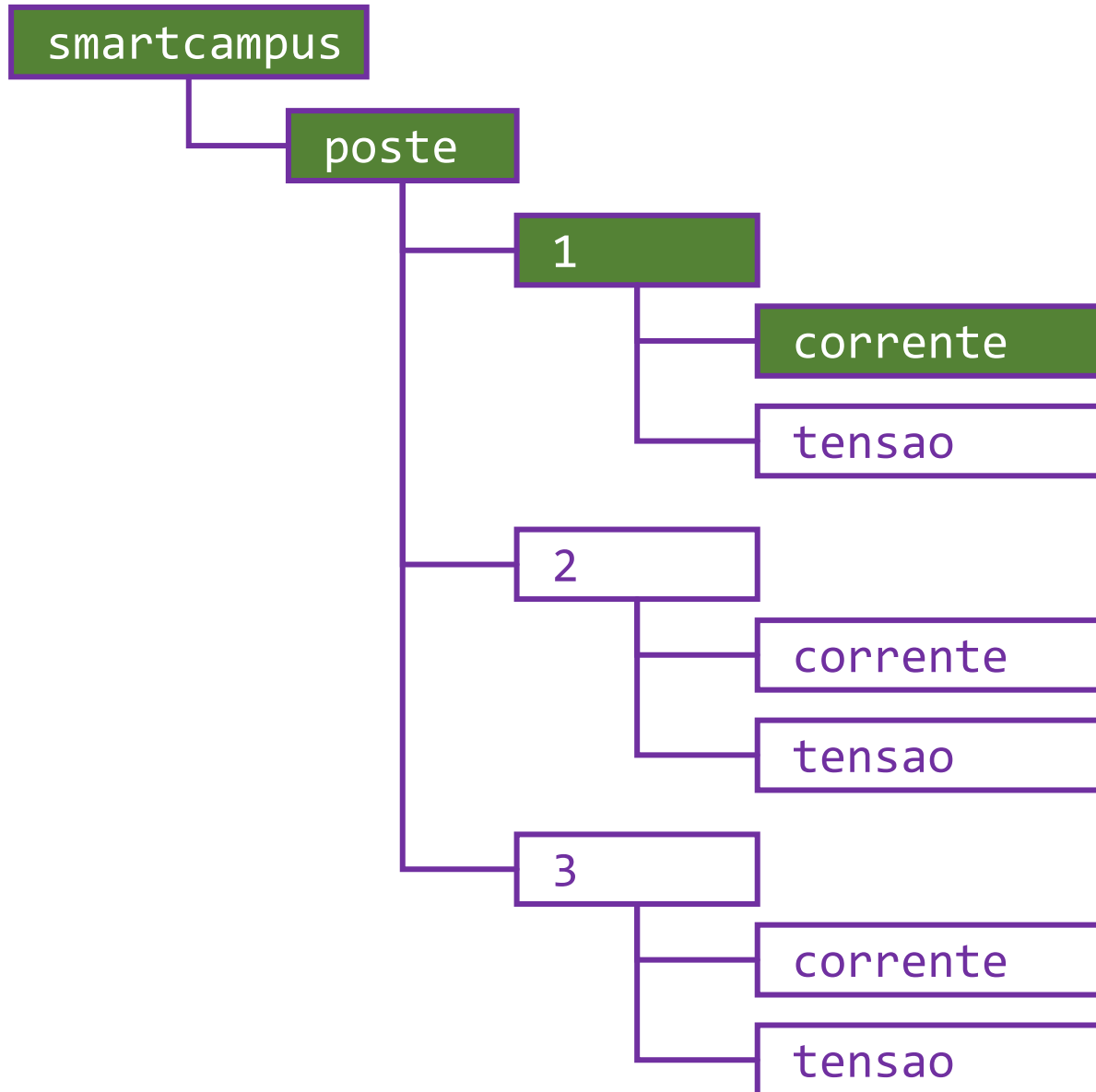
Existem 2 estilos para nomes de tópico:

- Estilo **simples**: sensor
- Estilo **hierárquico**: smartcampus/poste/1/corrente



Árvore de tópicos

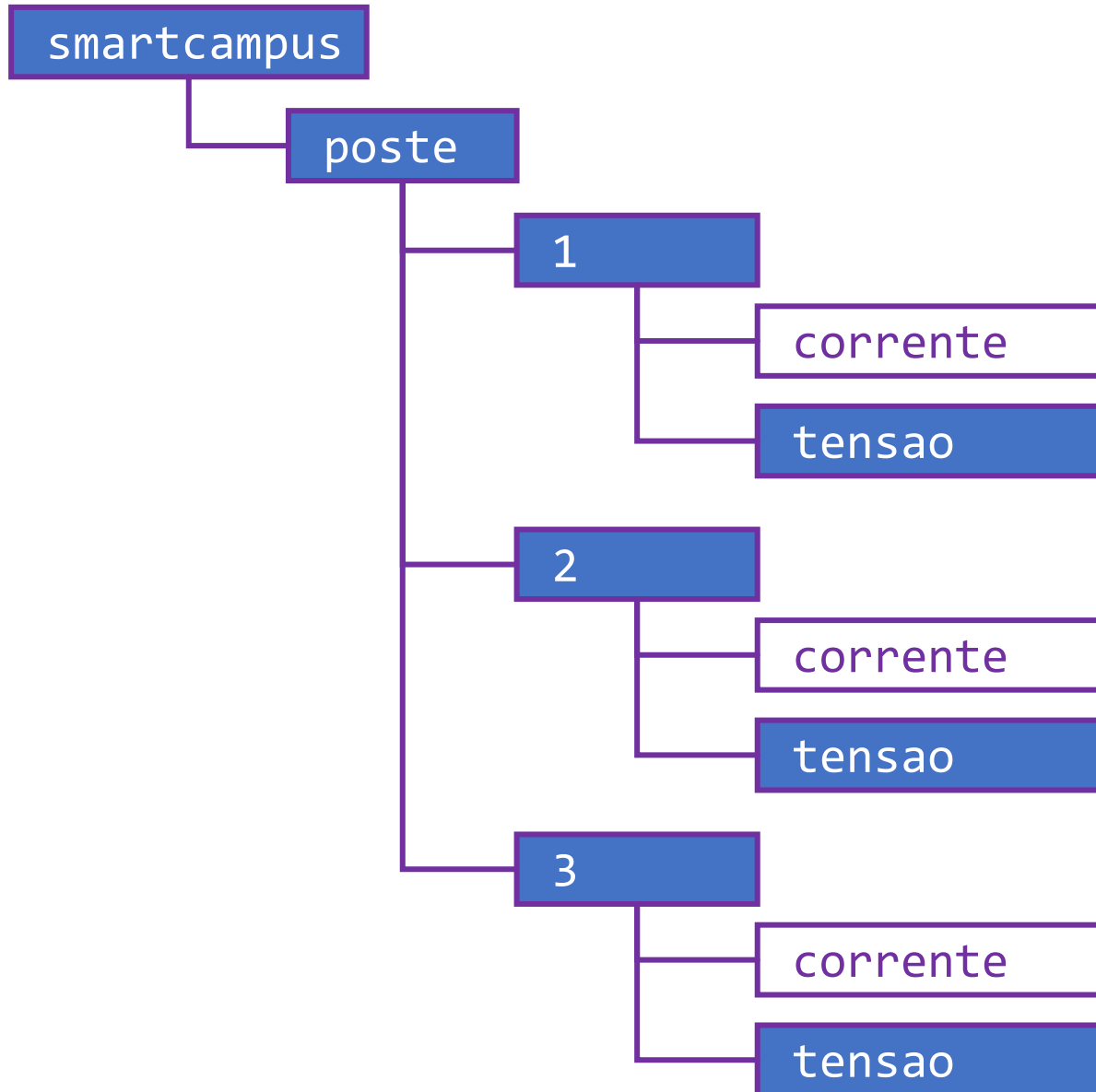
Como assinar tópicos (1/4)



Tópico único

smartcampus/poste/1/corrente

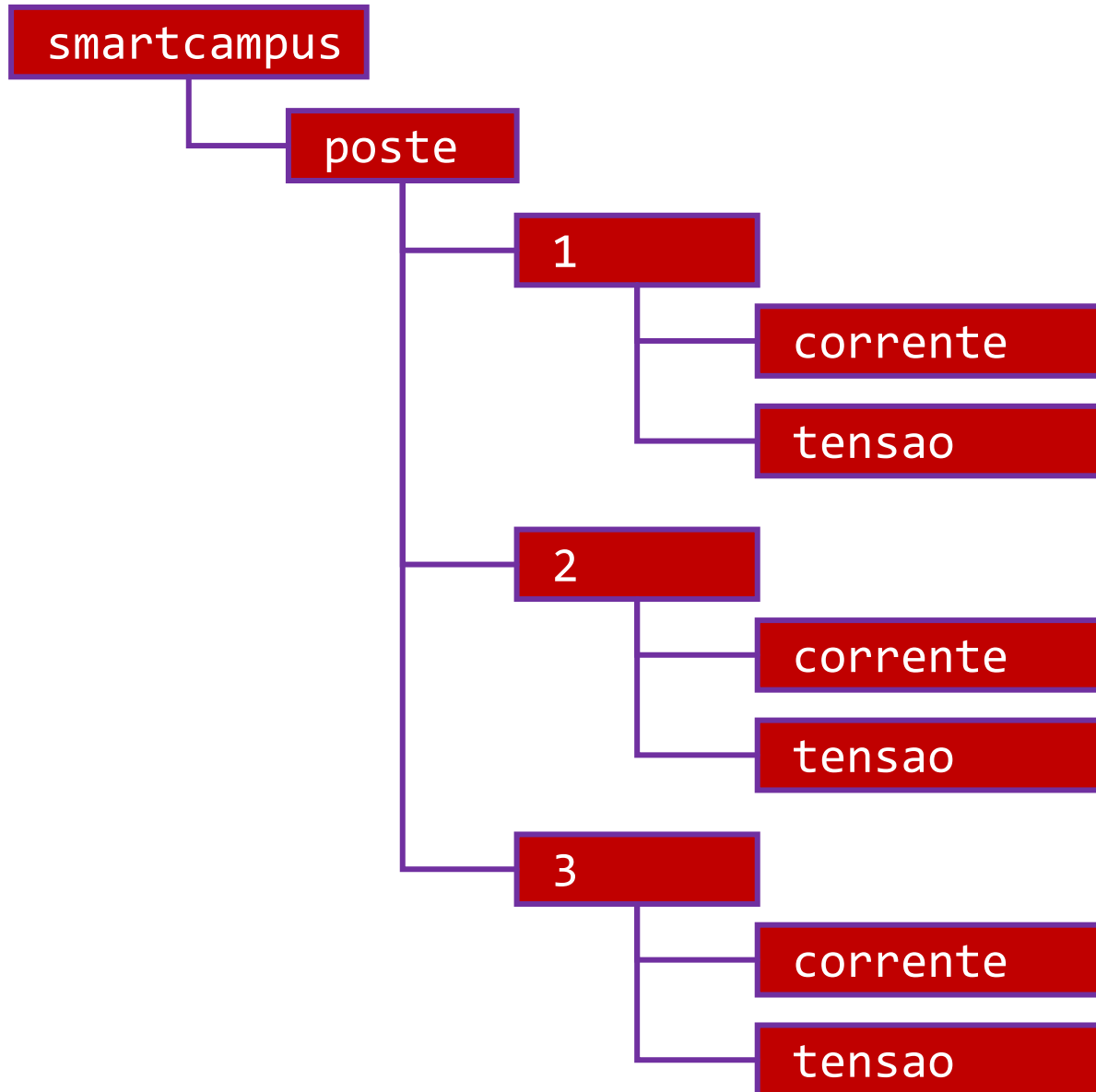
Como assinar tópicos (2/4)



*Múltiplos tópicos no
mesmo nível*

smartcampus/poste/+ /tensao

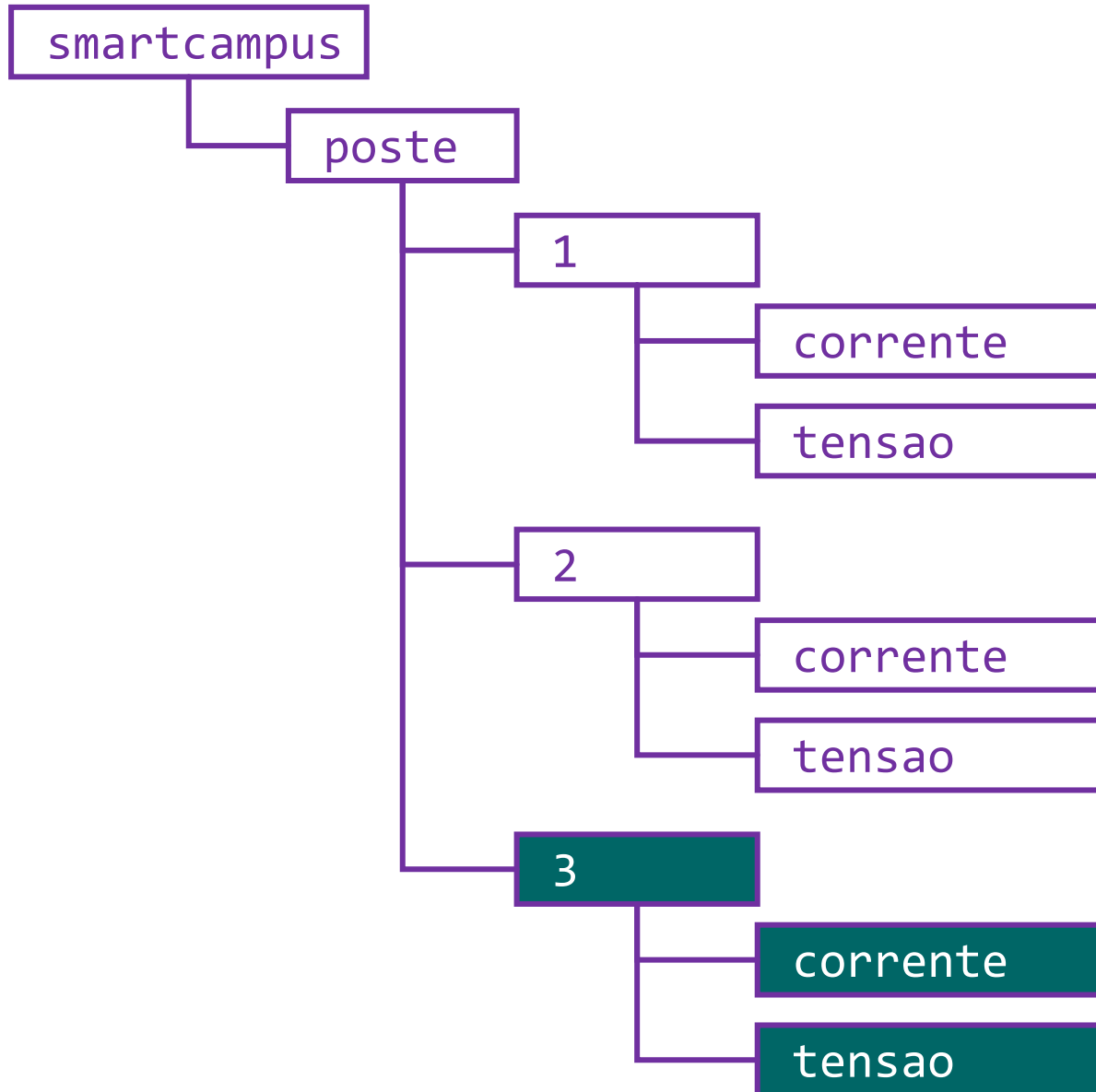
Como assinar tópicos (3/4)



Múltiplos tópicos em *sub-níveis*

smartcampus/#

Como assinar tópicos (4/4)



*Múltiplos tópicos em **sub-níveis***

smartcampus/poste/3/#

CONNECT	Solicitar o estabelecimento de conexão com o broker
PUBLISH	Publica uma mensagem (<i>payload</i>) em um tópico
SUBSCRIBE	Requisita a assinatura a um ou mais tópicos
UNSUBSCRIBE	Requisica a remoção da assinatura ao(s) tópico(s)
DISCONNECT	Solicitar a desconexão com o broker

Login e senha (opcional)

Criptografia com SSL/TSL (opcional)

Retenção de mensagem (**retainFlag**)

Níveis de **QoS**:

0 (at most once)

1 (at least once)

2 (exactly once)

QoS é aplicado tanto para publicação quanto subscrição



Eclipse Mosquitto™
An open source MQTT broker

- Mantido pela **Fundação Eclipse** dentro do projeto **iot.eclipse.org**
- Suporte às versões MQTT 3.1, 3.1.1 e 5.0
- Escrito na linguagem C
- Utilitários como **mosquitto_sub** e **mosquitto_pub**

Exercícios:

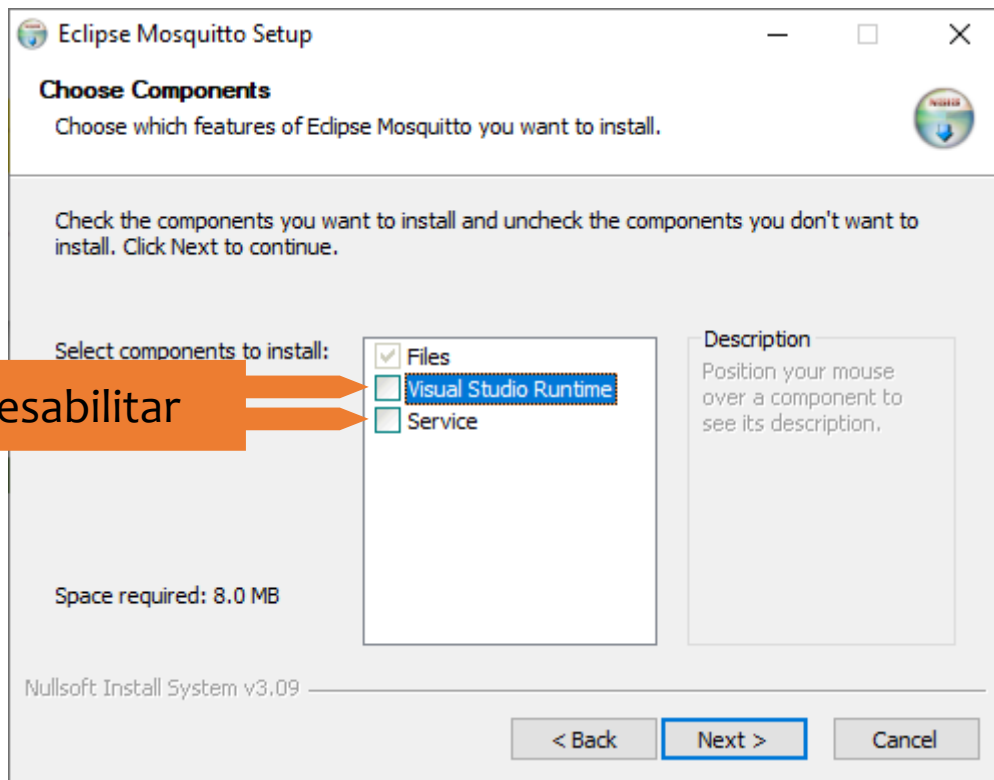
Instalando o **Mosquitto**

1 Baixar <https://mosquitto.org/download>

Windows

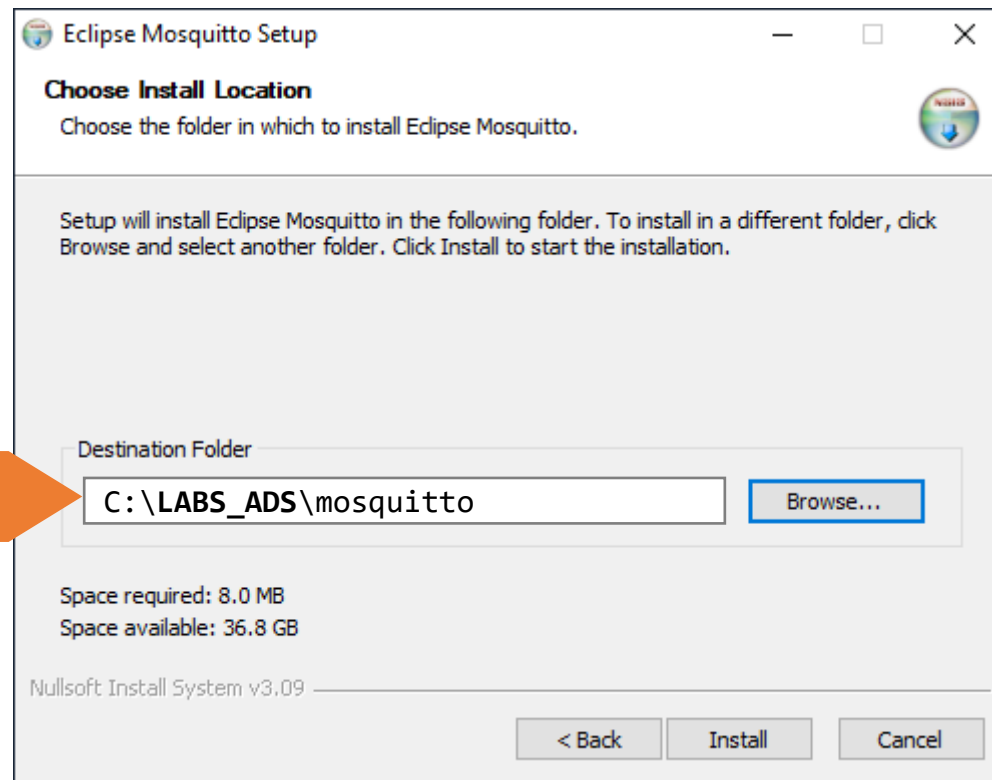
- [mosquitto-2.0.18-install-windows-x64.exe](#) (64-bit build, Windows Vista and later)
- [mosquitto-2.0.18-install-windows-x32.exe](#) (32-bit build, Windows Vista and later)

2 Desabilitar o Mosquito como *service*:

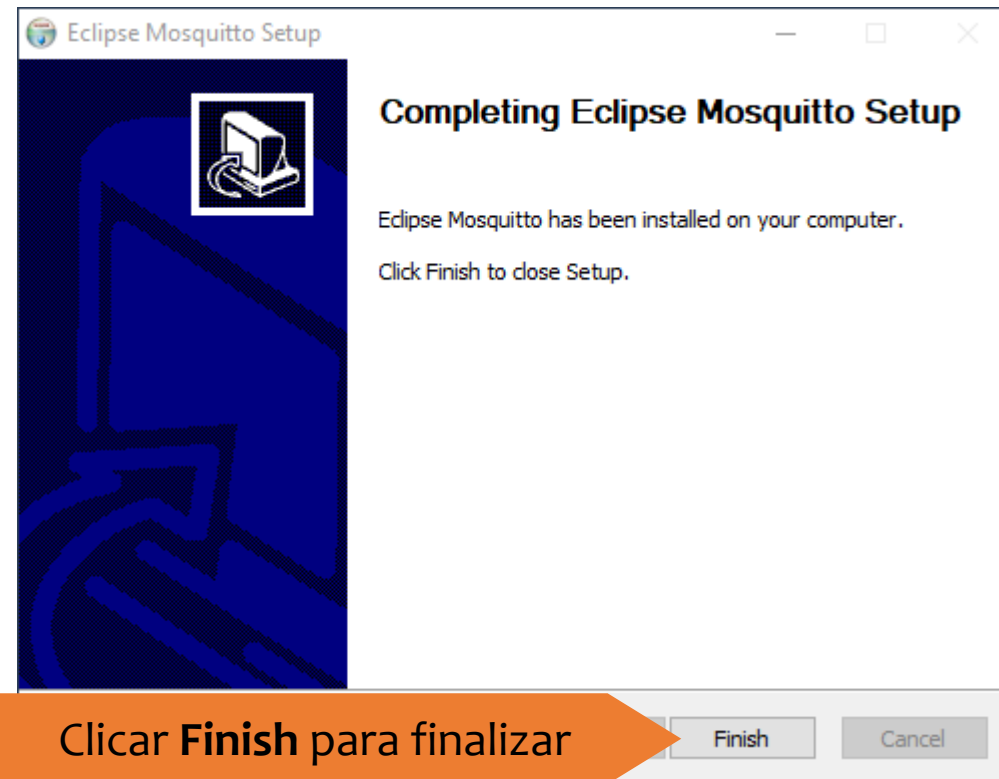


3 Selecionar a pasta de instalação:

Selecionar



4 Completar a instalação



5 Executando o Broker

- Abrir um prompt de comando;
- Ir para a diretório que o Broker foi instalado;
- Iniciar o Broker com o comando:

```
...\mosquitto>mosquitto.exe -v
```

Inicia o serviço do broker

```
C:\LABS_ADS\mosquitto>mosquitto.exe -v
1701022833: mosquitto version 2.0.18 starting
1701022833: Using default config.
1701022833: Starting in local only mode. Connections will only be possible from clients running on this machine.
1701022833: Create a configuration file which defines a listener to allow remote access.
1701022833: For more details see https://mosquitto.org/documentation/authentication-methods/
1701022833: Opening ipv4 listen socket on port 1883.
1701022833: Opening ipv6 listen socket on port 1883.
1701022833: mosquitto version 2.0.18 running
```

A janela fica travada
logando tudo

6 Executando o **Subscriber**:

- Abrir outro prompt de comando e ir a mesma pasta;
- Iniciar o **subscriber** com as seguintes configurações:

Comando	Descrição	Valor
-h	Host	localhost
-p	Porta	1883
-t	Tópico	"sensor"

Assina um tópico

```
...\mosquitto>mosquitto_sub -h localhost -p 1883 -t "sensor"
```

7 Executando um **Publisher**

- Abrir outro prompt de comando e ir para o diretório de instalação;
- Iniciar um **publisher** e enviar uma mensagem:

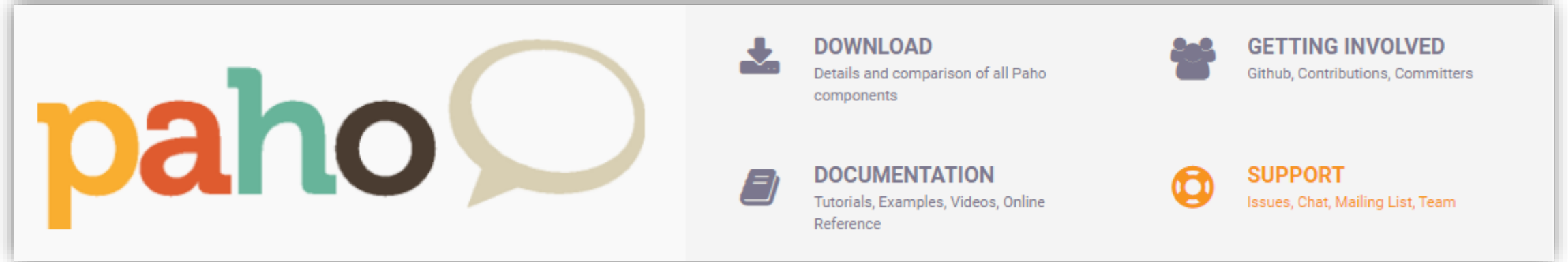
Comando	Descrição	Valor
-h	Host	Localhost
-p	Porta	1883
-t	Tópico	"sensor"
-m	Mensagem	"T:30"

Publica num tópico

```
...\mosquitto>mosquitto_pub -h localhost -p 1883 -t "sensor" -m "T:30"
```

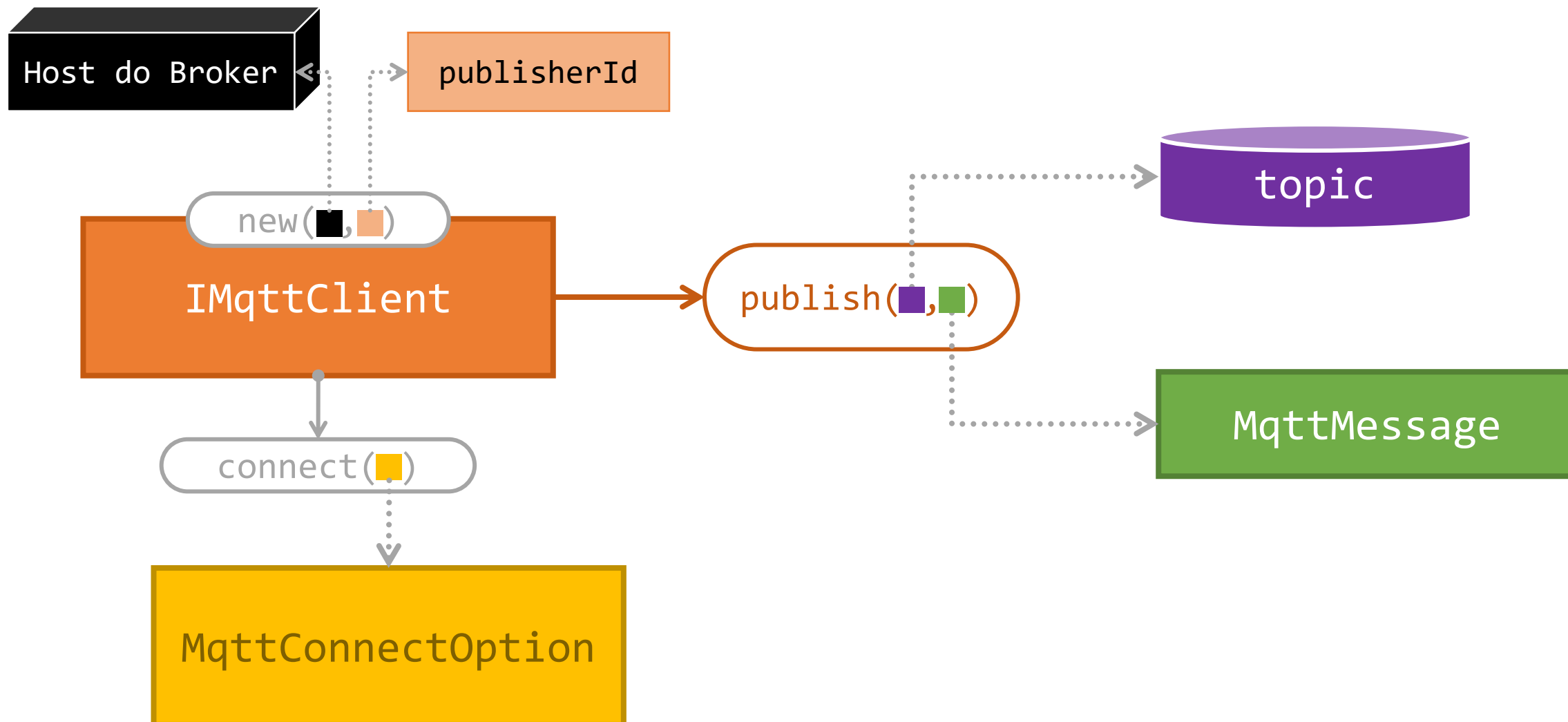
- ❖ A janela do **Subscriber** vai mostrar a mensagem recebida;
- ❖ Publicar outras mensagens e visualizar as janelas do **Broker** e **Subscriber**

Programando MQTT com

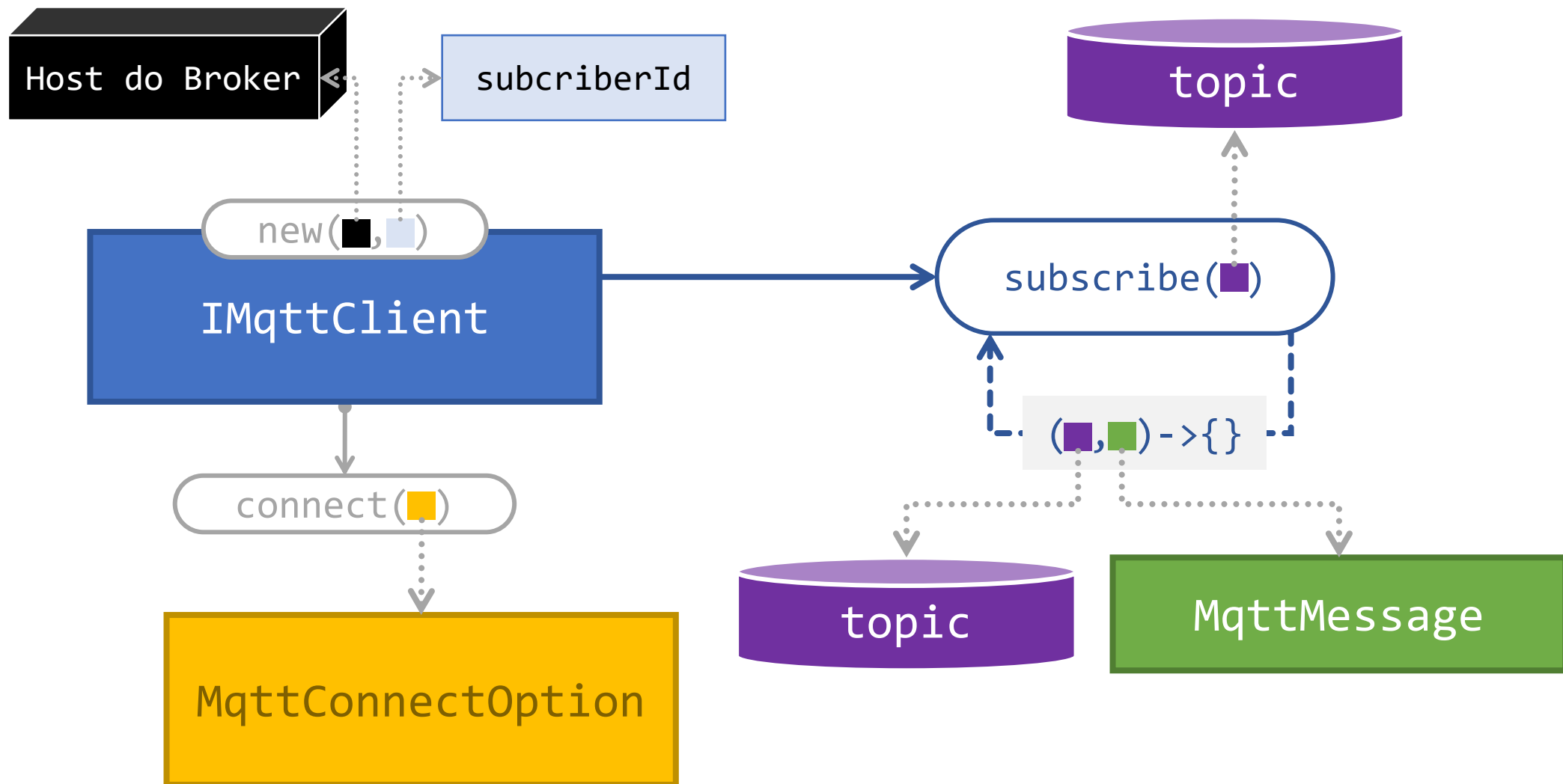


- Biblioteca cliente para acessar **brokers MQTT**
- Versão em **Java**, C, Python, NodeJS.
- Mantido pela **Fundação Eclipse** dentro do projeto **iot.eclipse.org**

Publisher



Subscriber



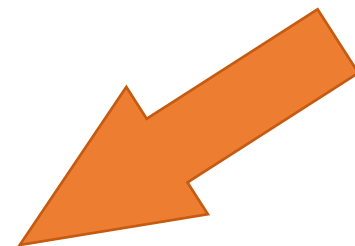
1 Criar um Maven Project acessando start.spring.io

Project	Maven Project
Language	Java
Spring Boot	(versão estável)

Dependencies
<i>(nenhuma dependencia)</i>

Project Metadata:	Group	br.inatel.labs
	Artifact	Padrao_MOM
	Name	Padrao_MOM
	Description	Aplicação MOM com MQTT
	Package name	br.inatel.labs.padrao_mom
	Packaging	Jar
	Java	17

GENERATE CTRL + ↵



2 Importar como Maven Project

- Abrir o Eclipse;
- Importar projeto como Existing Maven Project:

3 Adicionar a dependência do Eclipse Paho

- Abrir o arquivo **pom.xml**
- Adicionar a seguinte dependência:

```
<dependency>  
  <groupId>org.eclipse.paho</groupId>  
  <artifactId>org.eclipse.paho.client.mqttv3</artifactId>  
  <version>1.2.5</version>  
</dependency>
```

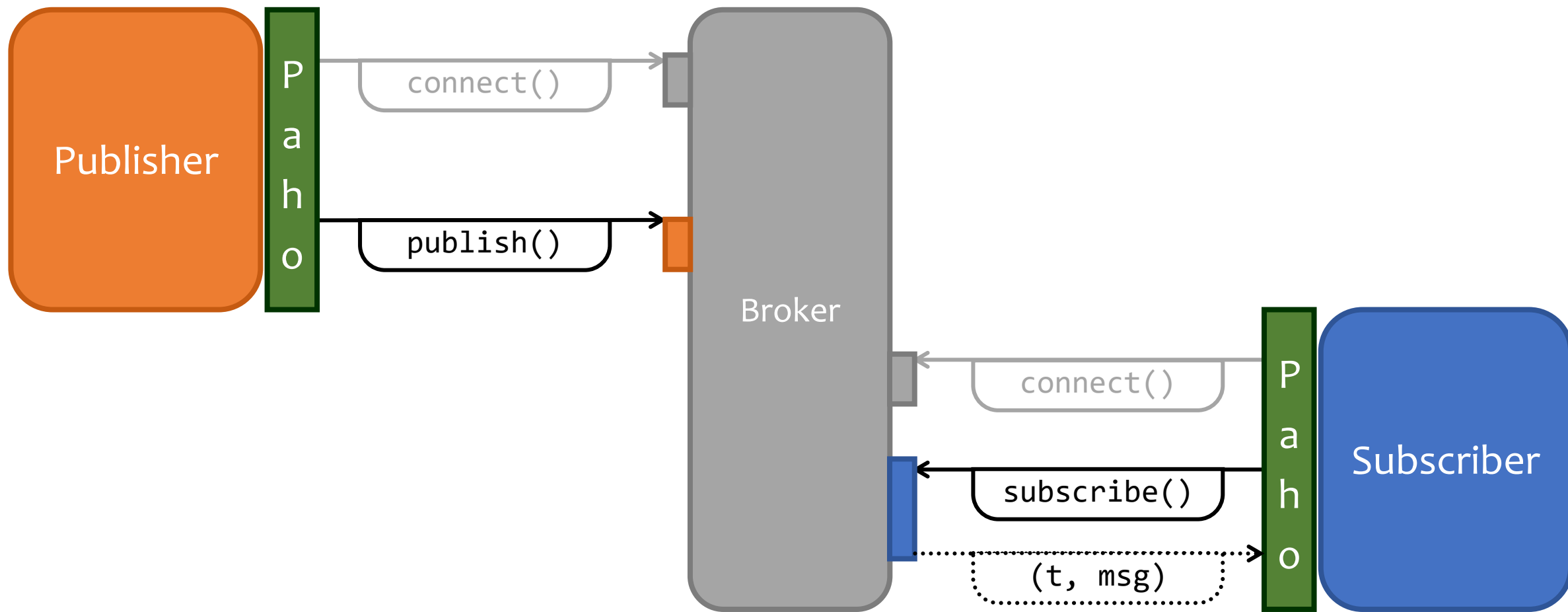
Onde conseguimos esta configuração?

<https://mvnrepository.com/artifact/org.eclipse.paho/org.eclipse.paho.client.mqttv3/1.2.5>

4 Codar a interface que guardas as constantes:

- Criar o sub-pacote **client**
- Nesta sub-pacote, criar a interface **MyConstants**

```
public interface MyConstants {  
  
    String URI_BROKER = "tcp://localhost:1883";  
  
    String TOPIC_SENSOR = "sensor";  
}
```

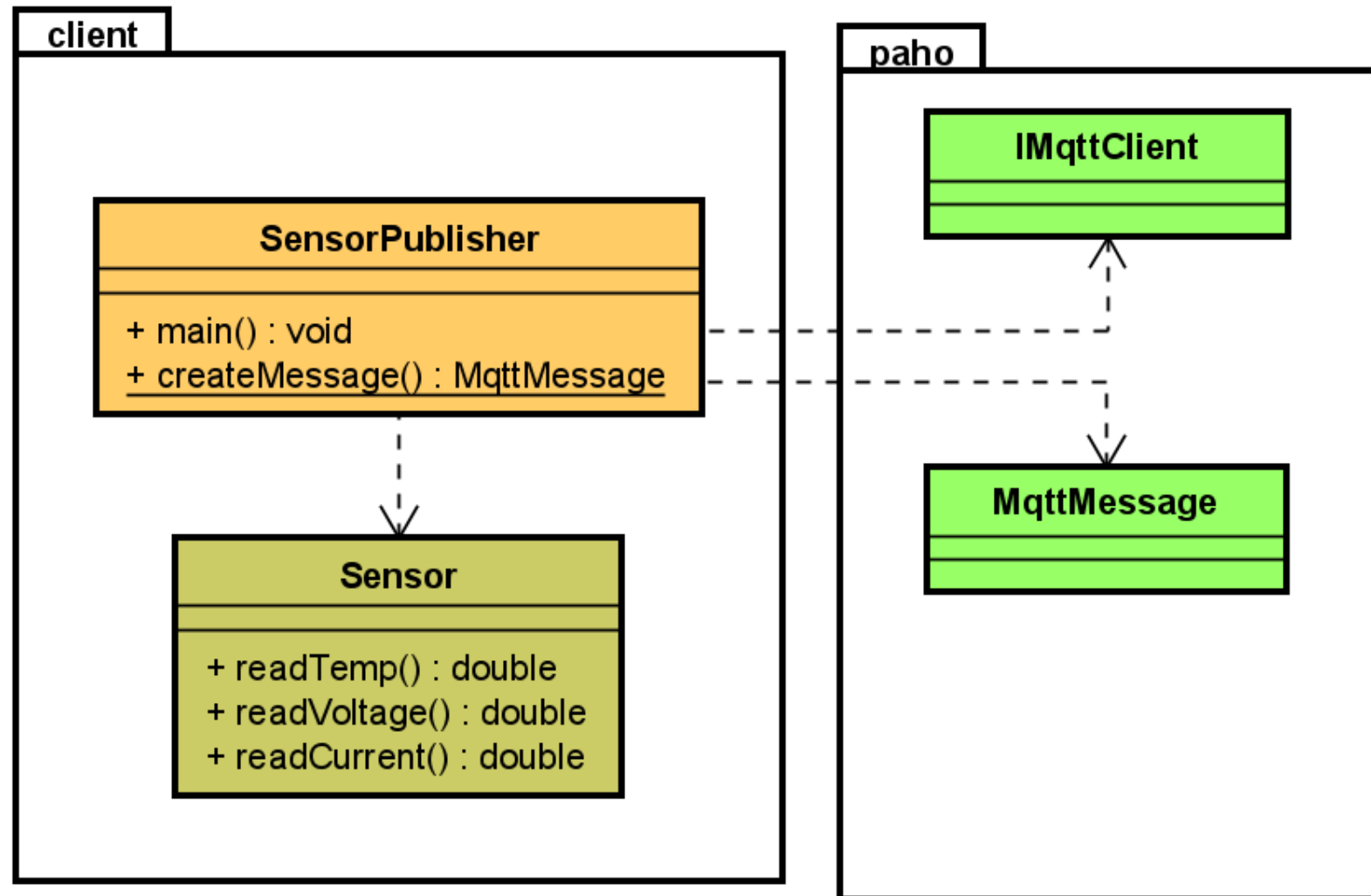


Exercícios:

Codando o **Publisher**

Publisher

P1 Modelo de Classes do Publisher



P2 Codar a **Sensor**

1 de 4

- No sub-pacote client, criar a classe Sensor

```
public class Sensor {  
  
}
```

P2 Codar a **Sensor**

2 de 4

- Codar o método readTemp():

```
public double readTemp() {  
    double value = new Random().doubles(-40, 80)  
        .findAny()  
        .getAsDouble();  
    return value;  
}
```

P3 Codar a **Sensor**

3 de 4

- Codar o método readVoltage():

```
public double readVoltage() {  
    return new Random().doubles(0, 380)  
        .findAny()  
        .getAsDouble();  
}
```

P4 Codar a **Sensor**

4 de 4

- Codar o método readCurrent():

```
public double readCurrent() {  
    return new Random().doubles(0, 60)  
        .findAny()  
        .getAsDouble();  
}
```

P5 Codar a classe **SensorPublisher** 1 de 4

- No sub-pacote client, criar a classe **SensorPublisher**:

```
package br.inatel.labs.padrao_mom.client;  
  
public class SensorPublisher {  
  
}
```

P6 Codar a classe **SensorPublisher** 2 de 4

- Codar o método estático **createMessage()**

```
private static MqttMessage createMessage(double value) {  
    byte[] payload = String.format("T:%04.2f", value).getBytes();  
    return new MqttMessage( payload );  
}
```

P7 Codar a classe **SensorPublisher** 3 de 4

- Continuar codando o método main()

```
//1.criar o publisher:
String publisherId = "Publicador_489"; //colocar sua matricula aqui
IMqttClient publisher = new MqttClient(MyConstants.URI_BROKER, publisherId);
```

```
//2.conectar ao broker
MqttConnectOptions opts = new MqttConnectOptions();
opts.setAutomaticReconnect( true );
opts.setCleanSession( true );
opts.setConnectionTimeout( 10 );
publisher.connect( opts );
```

P8 Codar a classe **SensorPublisher** 4 de 4

- Concluir o método main()

```
//3.criar mensagem com leitura do sensor:  
Sensor sensor = new Sensor();  
double temperatura = sensor.readTemp();  
MqttMessage message = createMessage(temperatura);
```

```
//4.publicar no topic  
publisher.publish(MyConstants.TOPIC_SENSOR, message);
```

```
//5. desconectar  
publisher.disconnect();
```


P9 Testar a classe

- Analisar a janela o **Broker**
- Analisar a janela do **Subscriber**

P10 Programar um loop...

- Vamos implementar um **loop de 100 iterações** onde serão publicadas mensagens aleatórias
- Cada iteração vai aguardar **2 segundos**

Exercícios:

Codando o **Subscriber**

Subscriber

S1 Criar classe **SensorSubscriber** 1 de 2

- No sub-pacote client, criar **SensorSubscriber**
- Codar o método **main()**

```
//1.criar o subscriber  
String subscriberId = "Subscriber_489";//colocar sua matricula  
MqttClient subscriber = new MqttClient( MyConstants.URI_BROKER, subscriberId );
```

```
//2.conectar ao broker  
MqttConnectOptions opts = new MqttConnectOptions();  
opts.setAutomaticReconnect( true );  
opts.setCleanSession( true );  
opts.setConnectionTimeout( 10 );  
subscriber.connect(opts);
```

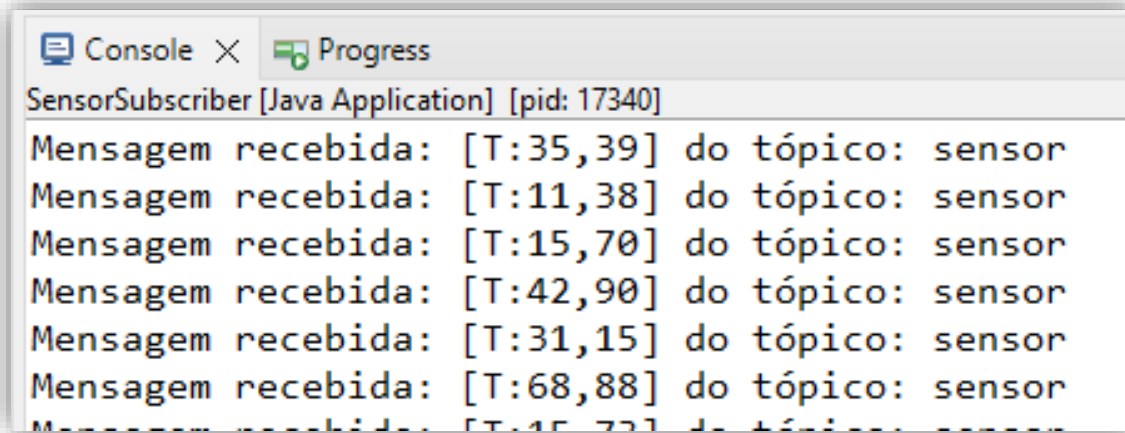
S2 Criar classe **SensorSubscriber** 2 de 2

- Concluir o método main()

```
//3.assinar e esperar por mensagens:  
subscriber.subscribe(MyConstants.TOPIC_SENSOR, (topic, msg) -> {  
    System.out.println("Mensagem recebida: [" + msg + "] do tópico: " + topic);  
});
```

S3 Testando a classe:

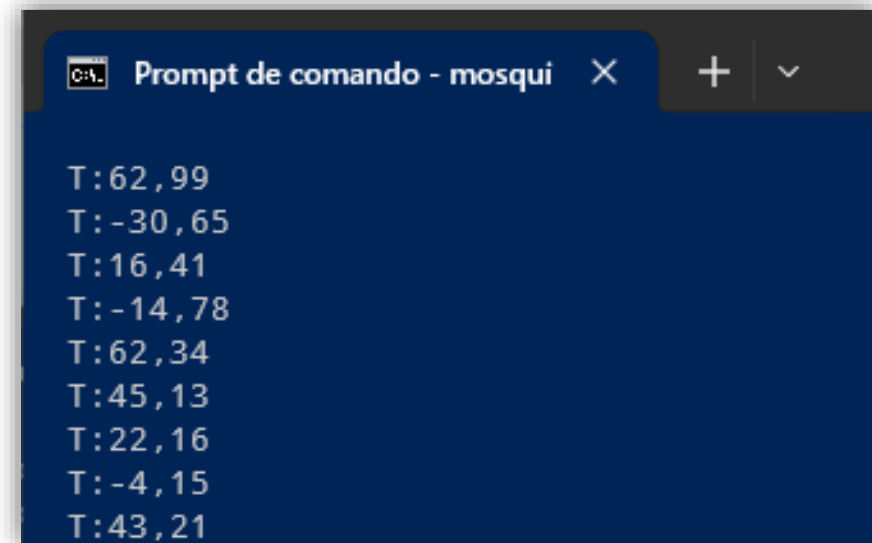
- Executar **SensorPublisher** (publicará em loop)
- Executar **SensorSubscriber** e observar as mensagens chegando
- Observar também a janela do subscriber:



Console × Progress

SensorSubscriber [Java Application] [pid: 17340]

```
Mensagem recebida: [T:35,39] do tópico: sensor
Mensagem recebida: [T:11,38] do tópico: sensor
Mensagem recebida: [T:15,70] do tópico: sensor
Mensagem recebida: [T:42,90] do tópico: sensor
Mensagem recebida: [T:31,15] do tópico: sensor
Mensagem recebida: [T:68,88] do tópico: sensor
Mensagem recebida: [T:15,72] do tópico: sensor
```

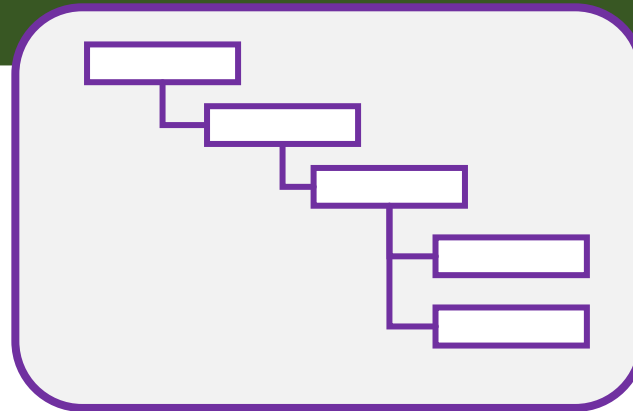


C:\ Prompt de comando - mosqui × + ▾

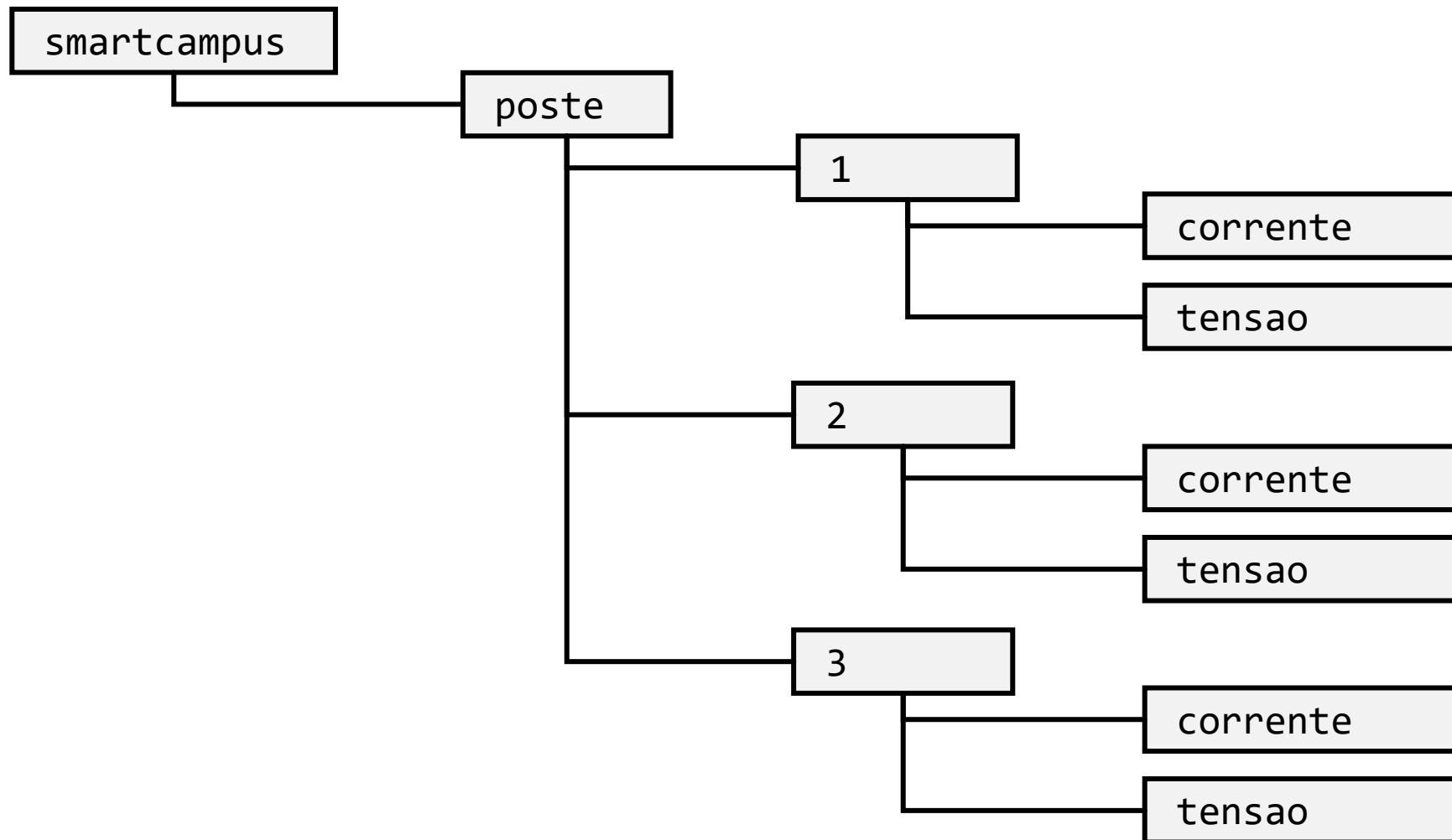
```
T:62,99
T:-30,65
T:16,41
T:-14,78
T:62,34
T:45,13
T:22,16
T:-4,15
T:43,21
```

Exercícios:

Trabalhando com **Tópicos Hierárquicos**

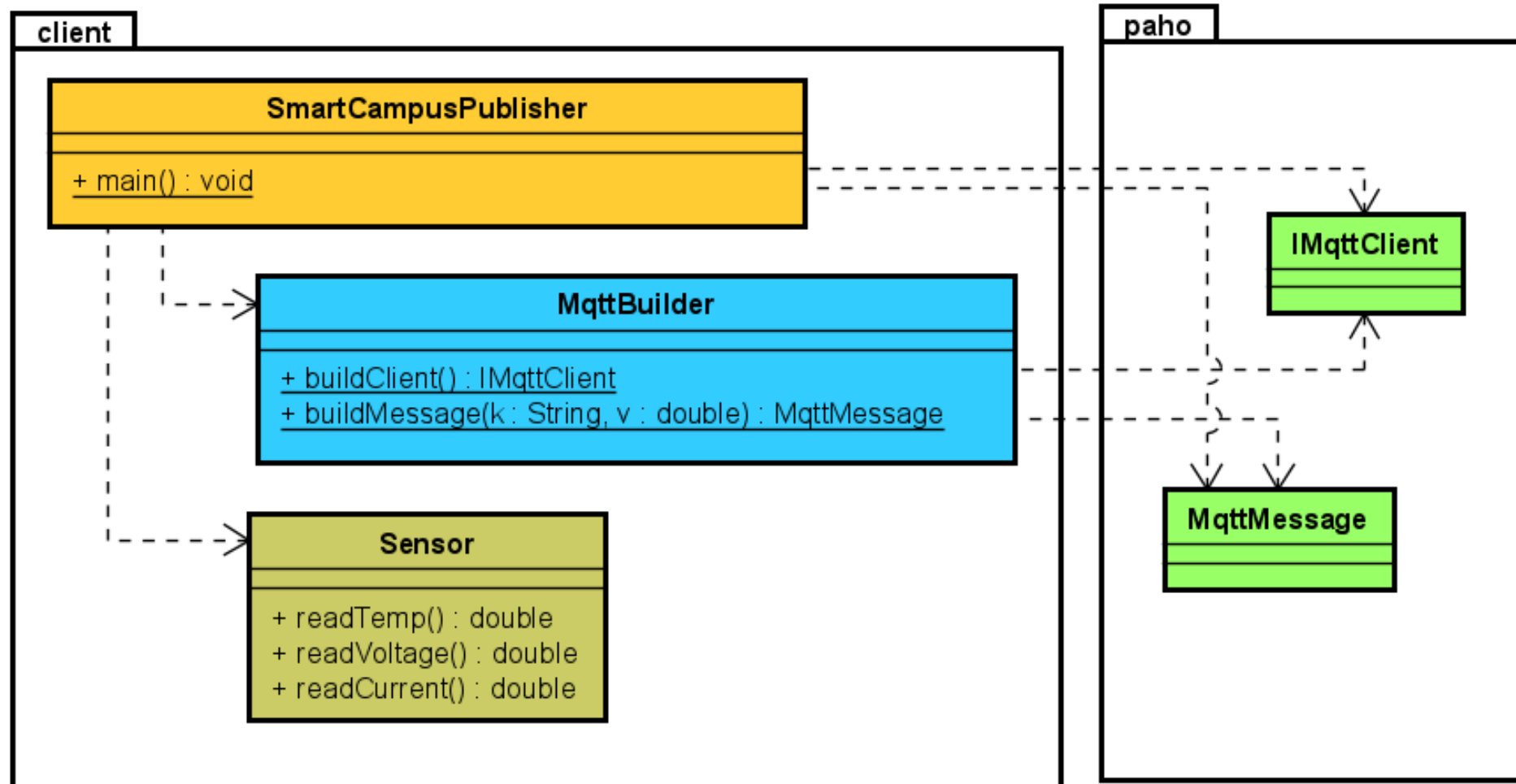


1 Vamos usar a seguinte árvore de tópicos:



Exercício: Tópicos hierárquicos

2 Vamos criar as classes segundo o diagrama:



3 Codar **MqttBuilder**

a) Criar a classe **MqttBuilder**:

```
public class MqttBuilder {  
  
}
```

4 Codar **MqttBuilder**

b) Codar o método **buildClient()**:

```
public static IMqttClient buildClient() throws Exception {  
    MqttClient connectedClient = new MqttClient(MyConstants.URI_BROKER, "Client_489");  
    MqttConnectOptions opts = new MqttConnectOptions();  
    opts.setAutomaticReconnect( true );  
    opts.setCleanSession( true );  
    opts.setConnectionTimeout( 10 );  
    connectedClient.connect( opts );  
  
    return connectedClient;  
}
```

5 Codar **MqttBuilder**

c) Codar o método **buildMessage()**:

```
public static MqttMessage buildMessage(String key, double value) {  
    String valueStr = String.format("%04.2f", value);  
    byte[] payload = String.format("%s:%s", key, valueStr).getBytes();  
    return new MqttMessage( payload );  
}
```

6 Codar a **SmartCampusPublisher**

a) Criar a classe **SmartCampusPublisher**:

```
public class SmartCampusPublisher {  
  
}
```

7 Codar a SmartCampusPublisher

b) Codar o método main()

```
//1.publisher
IMqttClient publisher = MqttBuilder.buildClient();

//2.sensor
Sensor sensor = new Sensor();

//3.postes:
String TOPIC_POSTE1_TENSAO = "smartcampus/poste/1/tensao";
String TOPIC_POSTE2_TENSAO = "smartcampus/poste/2/tensao";
String TOPIC_POSTE3_TENSAO = "smartcampus/poste/3/tensao";

String TOPIC_POSTE1_CORRENTE = "smartcampus/poste/1/corrente";
String TOPIC_POSTE2_CORRENTE = "smartcampus/poste/2/corrente";
String TOPIC_POSTE3_CORRENTE = "smartcampus/poste/3/corrente";
```

8 Codar a SmartCampusPublisher

c) Codar o método main()

```
//4.loop de publicações...  
for (int i=0; i<1000; i++) {  
  
    double valorTensao = 0;  
    MqttMessage msgTensao = null;  
    //publicar mensagens com leitura de tensão para cada poste aguardando 1000ms  
  
    double valorCorrente = 0.0;  
    MqttMessage msgCorrente = null;  
    //publicar mensagens com leitura de corrente para cada poste aguardando 1000ms  
  
}
```

9 Janelas executando `mosquitto_sub`

- Usando o utilitário `mosquitto_sub...`
- Abrir janelas de prompt para executar os comandos:

```
mosquitto_sub -h localhost -p 1883 -t "smartcampus/poste/#"
```

```
mosquitto_sub -h localhost -p 1883 -t "smartcampus/poste/+/tensao"
```

```
mosquitto_sub -h localhost -p 1883 -t "smartcampus/poste/1/#"
```

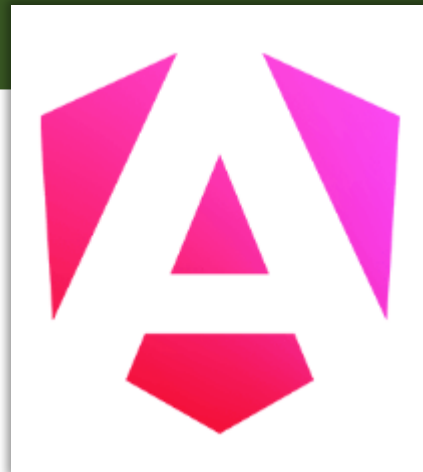
```
mosquitto_sub -h localhost -p 1883 -t "smartcampus/poste/+/corrente"
```


10 Testando...

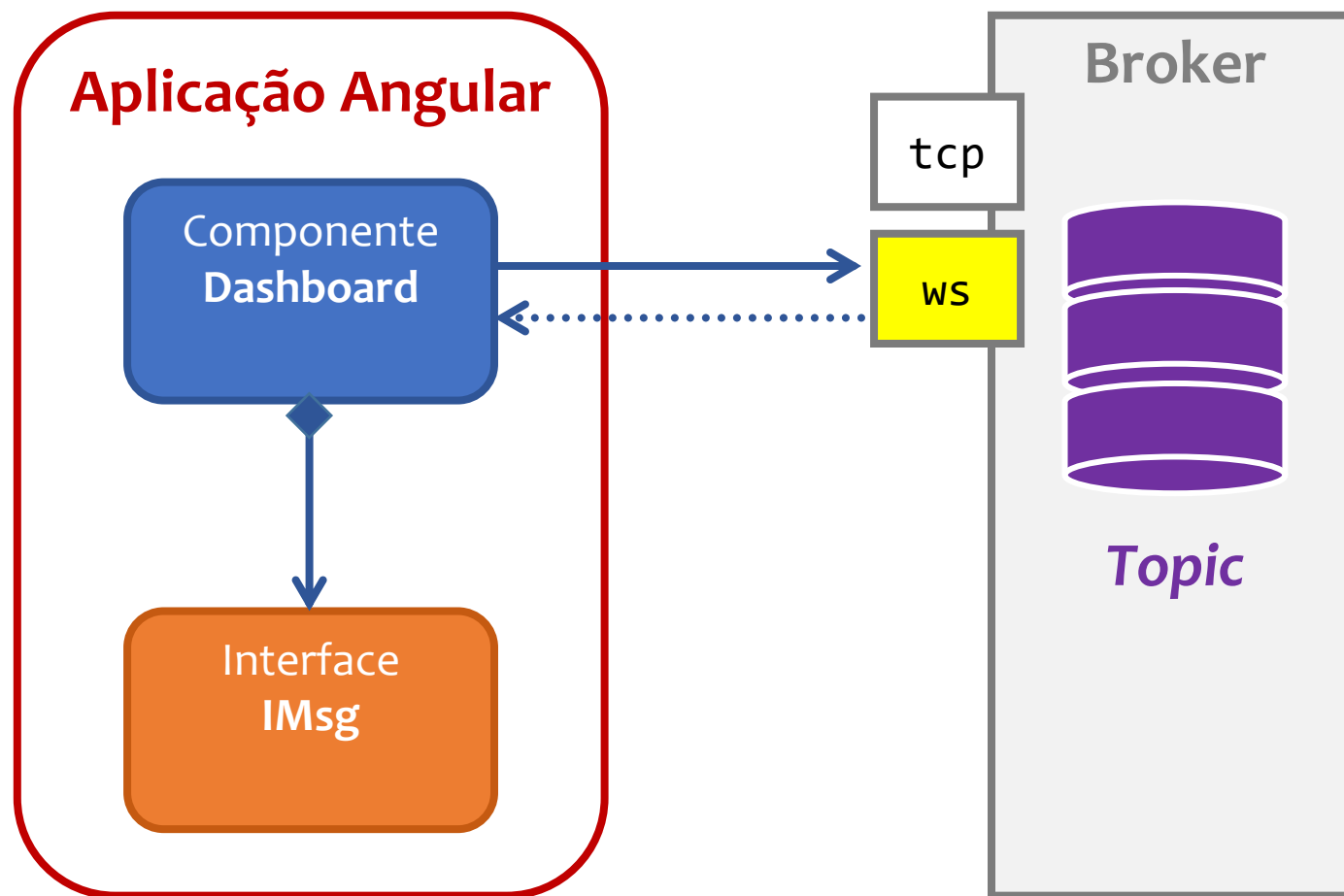
- Executar a classe **SmartCampusPublisher**
- Observar as mensagens que chegam em cada uma das janelas

Exercícios:

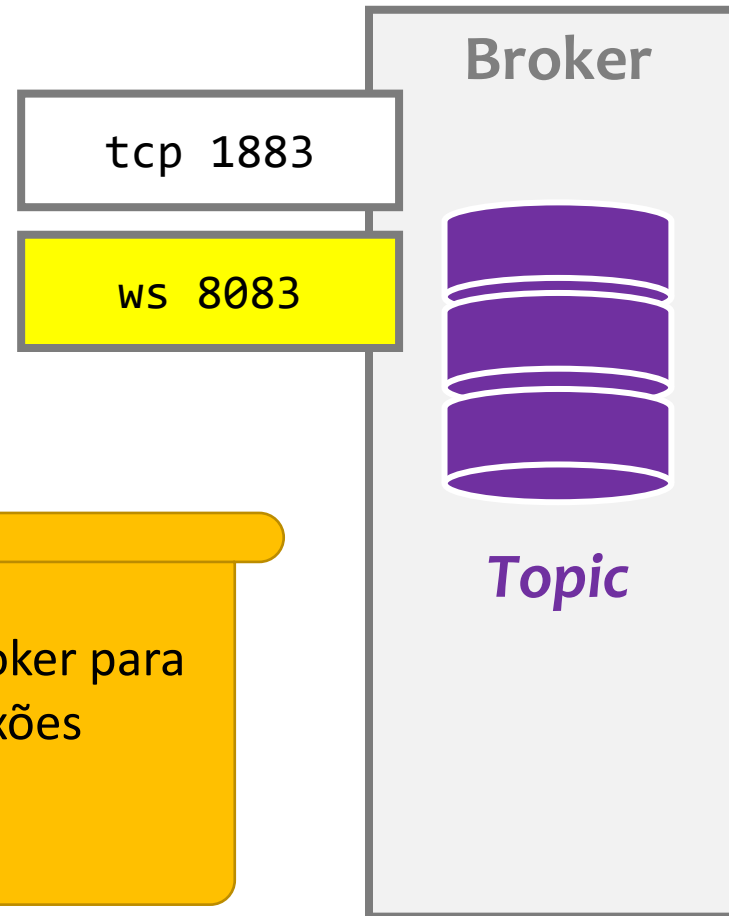
Desenvolvendo um Dashboard MQTT com Angular



1 Arquitetura da solução:



2 Configuração do Broker para WebSocket:



Precisamos configurar o Broker para também receber conexões **WebSockets**

3 Configuração do Broker para WebSockets:

- **Parar o Mosquitto**
- Na pasta de instalação do Mosquitto, abrir **mosquitto.conf**
- Adicionar as seguintes linhas no início do arquivo:

mosquitto.conf

```
listener 1883

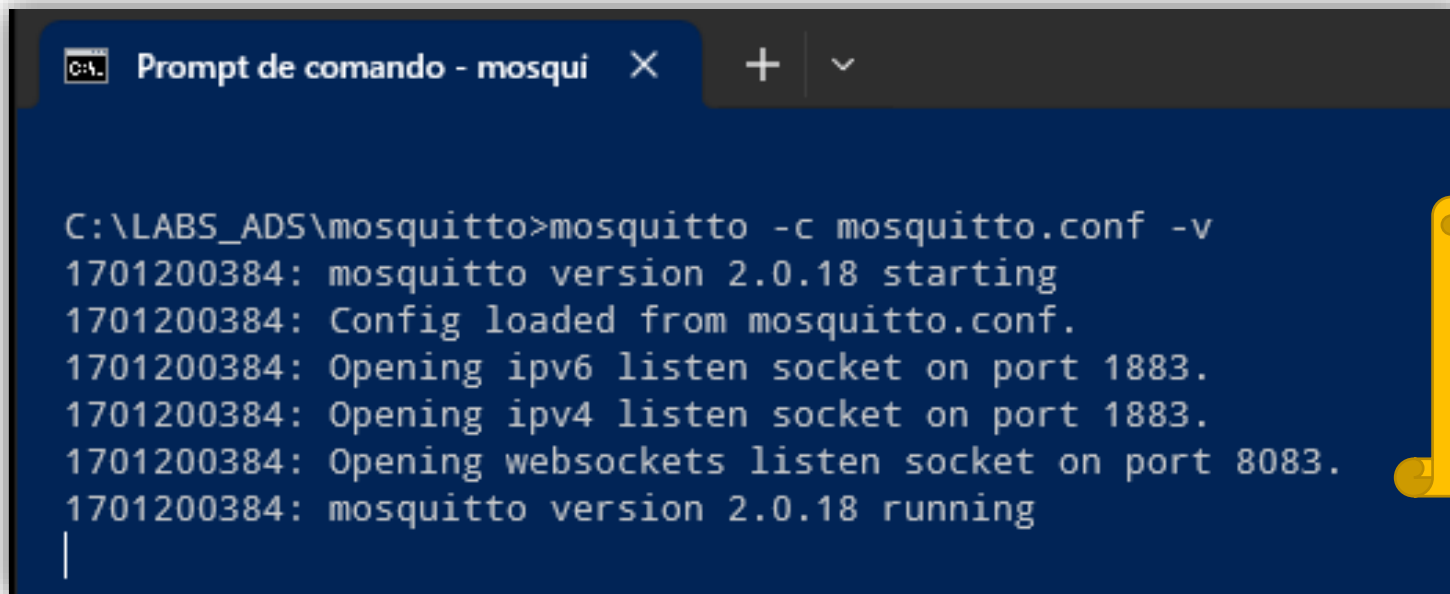
listener 8083
protocol websockets

allow_anonymous true
```

4 Executando o Broker com as novas configurações:

- Com prompt de comando na pasta do Mosquitto:

```
..\mosquitto>mosquitto -c mosquitto.conf -v
```



```
C:\LABS_ADS\mosquitto>mosquitto -c mosquitto.conf -v
1701200384: mosquitto version 2.0.18 starting
1701200384: Config loaded from mosquitto.conf.
1701200384: Opening ipv6 listen socket on port 1883.
1701200384: Opening ipv4 listen socket on port 1883.
1701200384: Opening websockets listen socket on port 8083.
1701200384: mosquitto version 2.0.18 running
```

Deixar a janela executando...

5 Criar o projeto Angular:

- Abrir um prompt de comando;
- Ir para a pasta **workspace_vscode**;
- Criar o projeto Angular com o comando:

```
ng new lab_mom --skip-tests --skip-git --routing=true --style=scss
```

6 Instalar a biblioteca cliente MQTT

- Entrar na pasta do projeto criado:

```
cd lab_mom
```

- Executar o comando:

```
..\lab_mom>npm install ngx-mqtt --save
```


7 Criar pastas:

```
cd src\app
```

```
..\app>mkdir components
```

```
..\app>mkdir models
```

8 Gerar os artefatos

- Gerar a interface para mensagens mqtt:

```
..\app>ng g interface models/i-msg
```

- Gerar o componente dashboard:

```
..\app>ng g component components/dashboard
```

9 Abrir o VSCode

```
..\app>code .
```

10 Importando o módulo **MqttModule**

- Em **app.module.ts**:

```
//...
import {IMqttServiceOptions, MqttModule } from 'ngx-mqtt';
```

```
const MQTT_SERVICE_OPTIONS : IMqttServiceOptions = {
  hostname: 'localhost',
  port: 8083,
};
```

Dados para conexão ao broker

```
@NgModule({
  declarations: [
    AppComponent,
    //...
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    MqttModule.forRoot(MQTT_SERVICE_OPTIONS)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

11 Codar IMsg

- Abrir `i-msg.ts`:

```
export interface IMsg {  
    id ? : number;  
  
    topic : string,  
  
    content : string;  
  
}
```

12 Codar DashboardComponent 1 de 3

- Abrir `dashboard.component.ts`
- Codar...

```
import { MqttService, IMqttMessage } from 'ngx-mqtt';
import { IMsg } from 'src/app/models/i-msg';

@Component({
  //...
})
export class DashboardComponent implements OnInit, OnDestroy {

  TOPIC_SMARTCAMPUS = 'smartcampus/poste/#';

  msgList : IMsg[] = [];

  constructor(private mqttService : MqttService) {

  }

  //...
```

Array com as mensagens vindas do tópico

Injeção de MqttService

13 Codar DashboardComponent 2 de 3

//...

```
ngOnInit(): void {
  try {
    this.mqttService.observe( this.TOPIC_SMARTCAMPUS ).subscribe( (mqttMessage : IMqttMessage ) => {
      const msg : IMsg = {
        id: mqttMessage.messageId,
        topic: mqttMessage.topic,
        content: mqttMessage.payload.toString()
      };
      this.msgList.push( msg );
    });
  } catch( err ) {
    console.error( 'Error ao assinar topico: ' + err );
  }
}
```

Instancia uma IMsg com dados recebidos...

...e adiciona ao array de mensagens

//...

14

Codar DashboardComponent

3 de 3

```
ngOnDestroy(): void {  
  this.mqttService.disconnect();  
}
```


15 Configurar a rota vazia para o DashboardComponent

- Abrir `app-routing.module.ts`

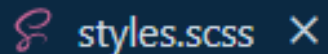
```
const routes: Routes = [  
  {  
    path: '',  
    component: DashboardComponent  
  }  
];  
  
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
export class AppRoutingModule { }
```

16 Estilizar com Bootstrap

- Através do prompt, instalar o Bootstrap com o comando:

```
ng add bootstrap@5.3.2 --save
```

- Editar **style.scss** e adicionar no início o seguinte trecho:

A screenshot of a Visual Studio Code editor tab. The tab is dark blue with a red squiggle icon on the left, the text "styles.scss" in the center, and a white "X" icon on the right to close the tab.

```
@import "bootstrap/dist/css/bootstrap.css";
```

17 Projetar a página do Dashboard 1 de 3

- Editar a página `dashboard.component.html`:

a) Envolver tudo com `div.container`:

```
<div class="container">  
  
    //...  
  
</div>
```

18 Projetar a página do Dashboard

2 de 3

b) Adicionar a parte inicial:

```
<h1>Dashboard MQTT</h1>

<div class="d-flex align-items-baseline justify-content-start gap-2 p-2 border border-2">
  <h6>Tópico:</h6>
  <h2 class="text-primary">{{ TOPIC_SMARTCAMPUS }}</h2>
</div>
```

19 Projetar a página do Dashboard 3 de 3

c) Adicionar a tabela com as mensagens:

```
<table class="table table-primary">
  <thead>
    <tr>
      <td>ID</td>
      <td>Tópico</td>
      <td>Conteúdo</td>
    </tr>
  </thead>

  <tbody>
    <ng-container *ngFor="let m of msgList" >
      <tr>
        <td> {{ m.id }} </td>
        <td> {{ m.topic }} </td>
        <td> {{ m.content }} </td>
      </tr>
    </ng-container>
  </tbody>
</table>
```

Pronto para testar...

20 Testando...

- a) Pelo Eclipse, deixar executando a classe **SmartCampusPublisher**
- b) Pelo prompt, subir o angular:

```
..lab_mom>ng serve -o
```

21 Testando...

c) Abrir o navegador para conferir

Dashboard MQTT

Tópico: [smartcampus/poste/#](#)

ID	Tópico	Conteúdo
1	smartcampus/poste/2/tensao	tensao:112,74
2	smartcampus/poste/3/tensao	tensao:45,93
3	smartcampus/poste/1/corrente	corrente:40,58
4	smartcampus/poste/2/corrente	corrente:47,49
5	smartcampus/poste/3/corrente	corrente:30,82
6	smartcampus/poste/1/tensao	tensao:351,33

22 Testando...

d) Testar assinar outros padrões de tópicos:

smartcampus/poste/1/#
smartcampus/poste/+/corrente
smartcampus/poste/+/tensao
smartcampus/poste/#
smartcampus/#

Desafio: Dashboad MQTT em Angular

- Adicionar botão Limpar na página para limpar as mensagens na tabela

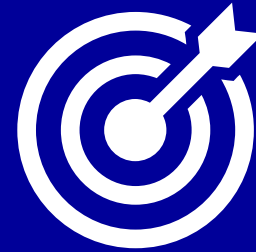
Dashboard MQTT

Tópico: [smartcampus/poste/#](#)

Limpar

ID	Tópico	Conteúdo
1	smartcampus/poste/2/tensao	tensao:77,96
2	smartcampus/poste/3/tensao	tensao:51,03
3	smartcampus/poste/1/corrente	corrente:14,78
4	smartcampus/poste/2/corrente	corrente:49,47
5	smartcampus/poste/3/corrente	corrente:48,41

Concluído







```
padraoArquitetural(mom -> mom.ok());
```