MQTT

MQTT (Message Queuing Telemetry Transport) is an open source, lightweight messaging protocol, optimized for low latency. This protocol provides a scalable and cost-efficient way to connect devices using a **publish/subscribe** model. A communication system built on MQTT consists of the publishing server, a broker and one or more clients. It is designed for constrained devices and low-bandwidth, high-latency or unreliable networks.

Installation

To start building MQTT-based microservices, first install the required package:

```
$ npm i --save mqtt
```

Overview

To use the MQTT transporter, pass the following options object to the `createMicroservice()` method:

```
@@filename(main)
const app = await NestFactory.createMicroservice<MicroserviceOptions>
(AppModule, {
  transport: Transport.MQTT,
  options: {
    url: 'mqtt://localhost:1883',
  },
});
@@switch
const app = await NestFactory.createMicroservice(AppModule, {
  transport: Transport.MQTT,
  options: {
    url: 'mqtt://localhost:1883',
  },
});
```

> info **Hint** The `Transport` enum is imported from the `@nestjs/microservices` package.

Options

The `options` object is specific to the chosen transporter. The **MQTT** transporter exposes the properties described here.

Client

Like other microservice transporters, you have several options for creating a MQTT `ClientProxy` instance.

One method for creating an instance is to use use the `ClientsModule`. To create a client instance with the `ClientsModule`, import it and use the `register()` method to pass an options object with the same properties shown above in the `createMicroservice()` method, as well as a `name` property to be used as the injection token. Read more about `ClientsModule` here.

```
@Module({
  imports: [
    ClientsModule.register([
      {
        name: 'MATH_SERVICE',
        transport: Transport.MQTT,
        options: {
          url: 'mqtt://localhost:1883',
        }
      },
    ]),
  ]
  ...
})
```

Other options to create a client (either `ClientProxyFactory` or `@Client()`) can be used as well. You can read about them here.

Context

In more sophisticated scenarios, you may want to access more information about the incoming request. When using the MQTT transporter, you can access the `MqttContext` object.

```
@@filename()
@MessagePattern('notifications')
getNotifications(@Payload() data: number[], @Ctx() context: MqttContext) {
  console.log(`Topic: ${context.getTopic()}`);
}
@@switch
@Bind(Payload(), Ctx())
@MessagePattern('notifications')
getNotifications(data, context) {
  console.log(`Topic: ${context.getTopic()}`);
}
```

> info **Hint** `@Payload()`, `@Ctx()` and `MqttContext` are imported from the `@nestjs/microservices` package.

To access the original mqtt packet, use the `getPacket()` method of the `MqttContext` object, as follows:

```
@@filename()
@MessagePattern('notifications')
```

```
  getNotifications(@Payload() data: number[], @Ctx() context: MqttContext) {
    console.log(context.getPacket());
  }
  @@switch
  @Bind(Payload(), Ctx())
  @MessagePattern('notifications')
  getNotifications(data, context) {
    console.log(context.getPacket());
  }
```

Wildcards

A subscription may be to an explicit topic, or it may include wildcards. Two wildcards are available, `+` and `#`. `+` is a single-level wildcard, while `#` is a multi-level wildcard which covers many topic levels.

```
  @@filename()
  @MessagePattern('sensors/+/temperature/+')
  getTemperature(@Ctx() context: MqttContext) {
    console.log(`Topic: ${context.getTopic()}`);
  }
  @@switch
  @Bind(Ctx())
  @MessagePattern('sensors/+/temperature/+')
  getTemperature(context) {
    console.log(`Topic: ${context.getTopic()}`);
  }
```

Record builders

To configure message options (adjust the QoS level, set the Retain or DUP flags, or add additional properties to the payload), you can use the `MqttRecordBuilder` class. For example, to set `QoS` to `2` use the `setQoS` method, as follows:

```
  const userProperties = { 'x-version': '1.0.0' };
  const record = new MqttRecordBuilder(':cat:')
    .setProperties({ userProperties })
    .setQoS(1)
    .build();
  client.send('replace-emoji', record).subscribe(...);
```

> info **Hint** `MqttRecordBuilder` class is exported from the `@nestjs/microservices` package.

And you can read these options on the server-side as well, by accessing the `MqttContext`.

```
  @@filename()
  @MessagePattern('replace-emoji')
```

```
  replaceEmoji(@Payload() data: string, @Ctx() context: MqttContext): string
  {
    const { properties: { userProperties } } = context.getPacket();
    return userProperties['x-version'] === '1.0.0' ? '🐱 ' : '🐈 ';
  }
  @@switch
  @Bind(Payload(), Ctx())
  @MessagePattern('replace-emoji')
  replaceEmoji(data, context) {
    const { properties: { userProperties } } = context.getPacket();
    return userProperties['x-version'] === '1.0.0' ? '🐱 ' : '🐈 ';
  }
```

In some cases you might want to configure user properties for multiple requests, you can pass these options to the `ClientProxyFactory`.

```
import { Module } from '@nestjs/common';
import { ClientProxyFactory, Transport } from '@nestjs/microservices';

@Module({
  providers: [
    {
      provide: 'API_v1',
      useFactory: () =>
        ClientProxyFactory.create({
          transport: Transport.MQTT,
          options: {
            url: 'mqtt://localhost:1833',
            userProperties: { 'x-version': '1.0.0' },
          },
        }),
    },
  ],
})
export class ApiModule {}
```