



cloud.iO Introduction

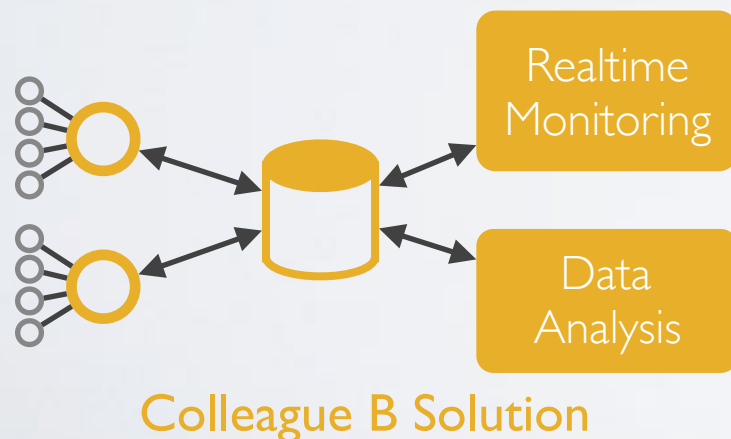
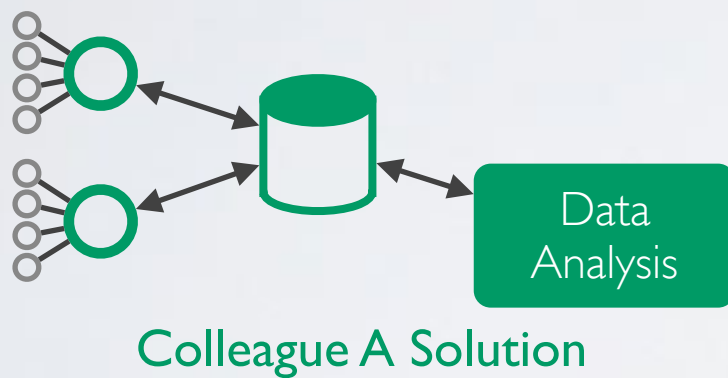
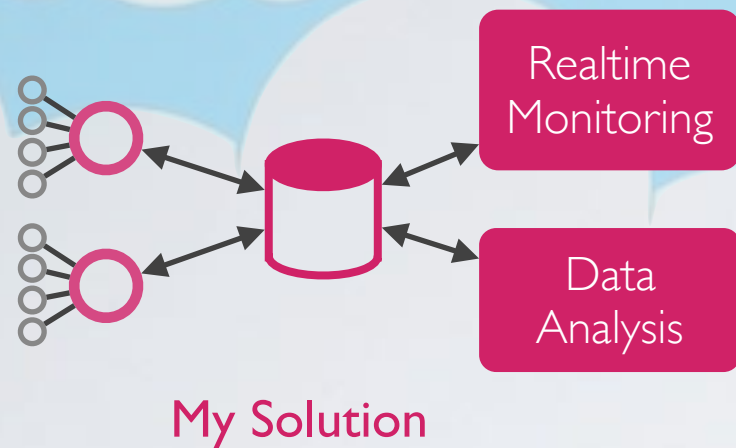
Michael Clausen





Why **cloud.iO**?

Motivation for cloud.iO



- **Our problems:**

- Each time a new solution for similar problems is developed.
- There is not that much budget to create a stable/flexible solution.
- Data analyst has to work each time with different systems.
- Data of different projects are not simple to compare and may have to be converted first.
- Scientists can not easily share their data with others.
- Control often is not possible, as the data passes by a database.
- Monitoring is mostly very inefficient (Database Polling for example)





cloud.iO Concepts

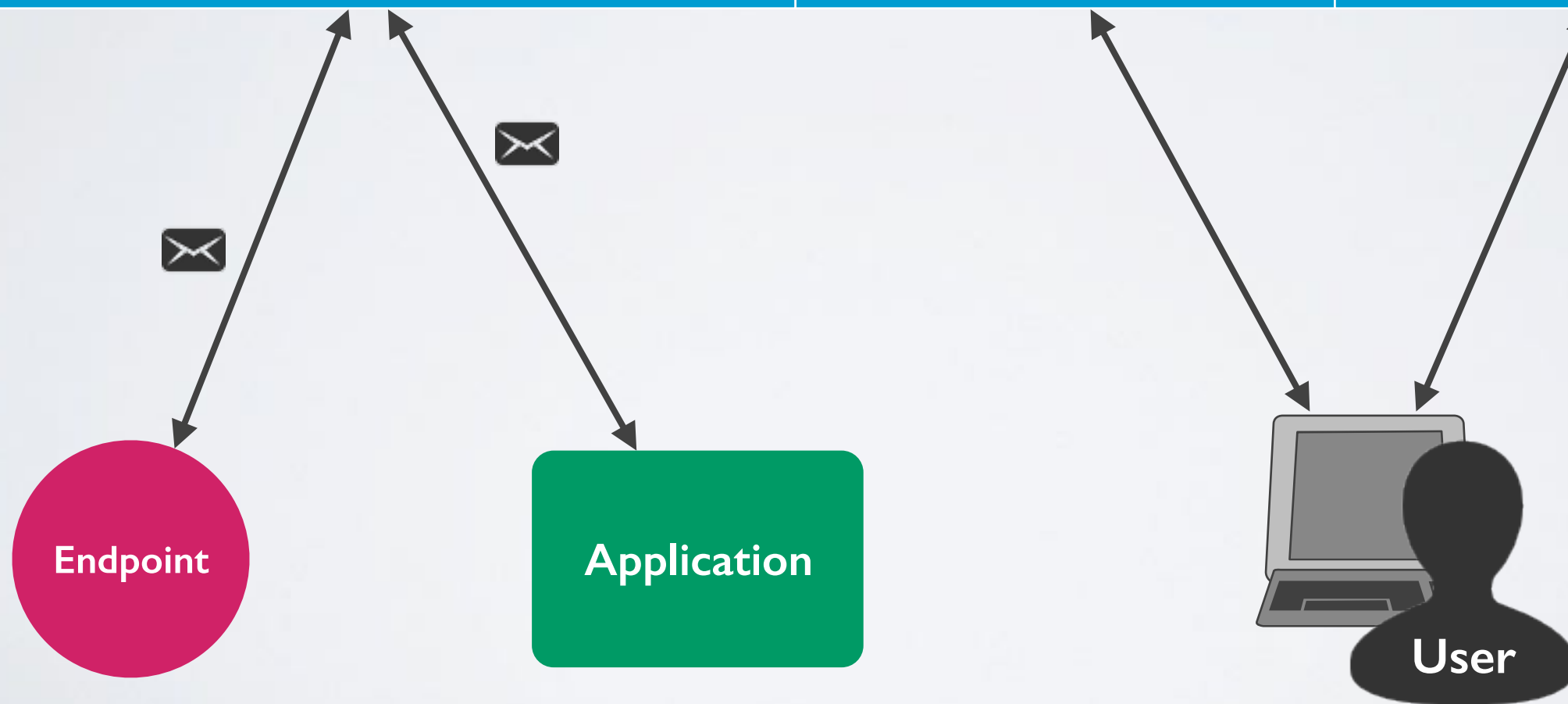
Overview



Message/Topic based API

REST API

Static Website (Using REST API)



Actors



Endpoint



Application



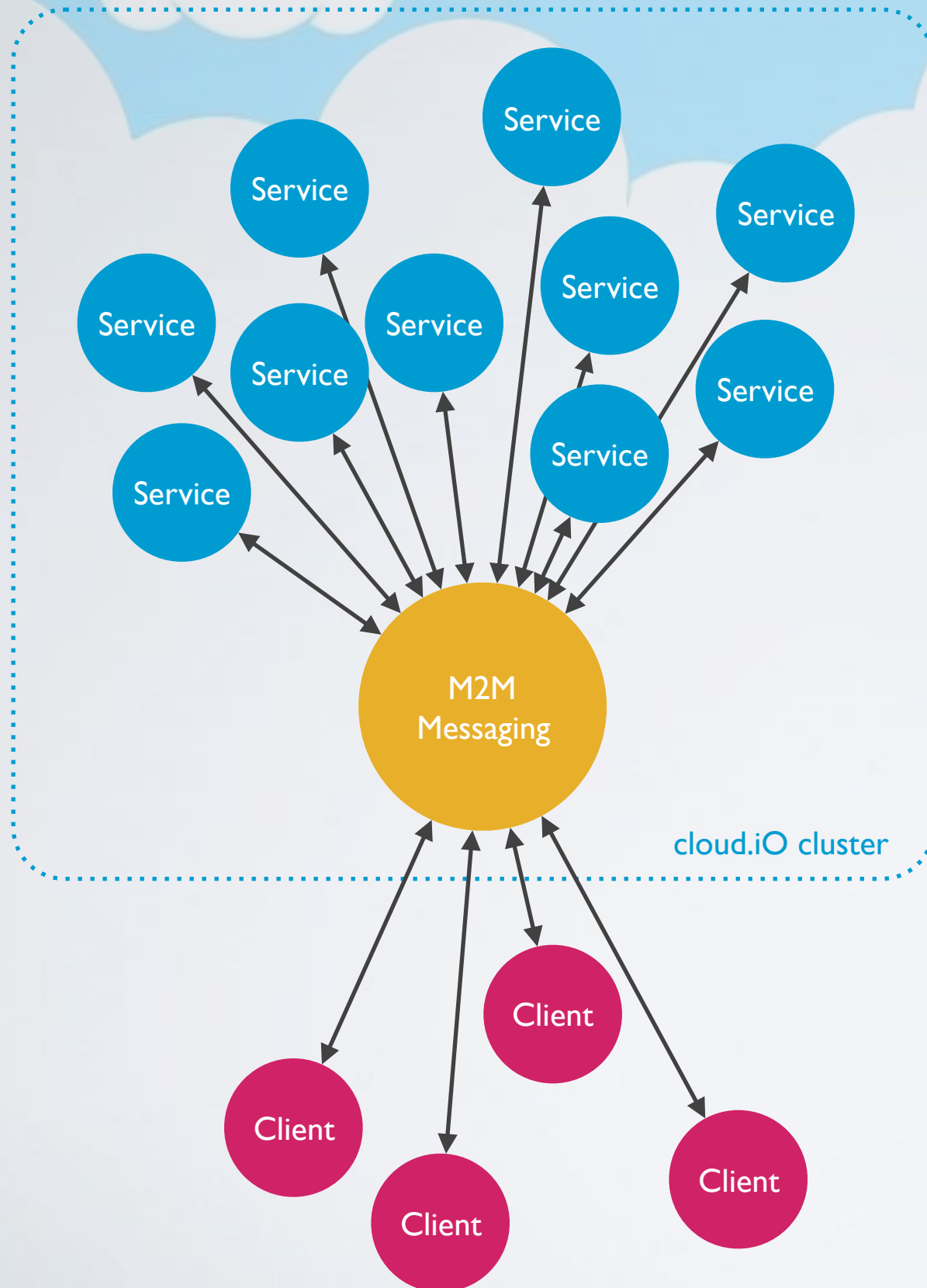
User

- **Endpoint**
 - Publishes it's complete data model.
 - Sends messages when some data of the model changes.
 - Some data can be modified from the outside (control, parameters).
- **Application**
 - Has access to a subset of all „Endpoints“ data models.
 - Can search for actual data using schemes (interfaces, data-classes).
 - Can get actual and historical data for „Endpoints“.
 - Can control set points and parameters of „Endpoints“.
- **User**
 - Owns one or more „Endpoints“.
 - Can give access to Endpoint to other users.
 - Can give access to Endpoint to applications.
 - Can write his own applications.
 - We actually do not distinguish between different kind of users (Simple users, developers, ...)



cloud.iO Components

Message based Microservice Architecture



- **Simplicity**

- A complex system can be composed by a lot of simple services.
- Most services are stateless.

- **Scalability**

- Services can be spread over multiple nodes.
- Services under high load can be dynamically deployed to additional cluster nodes.

- **Extensibility**

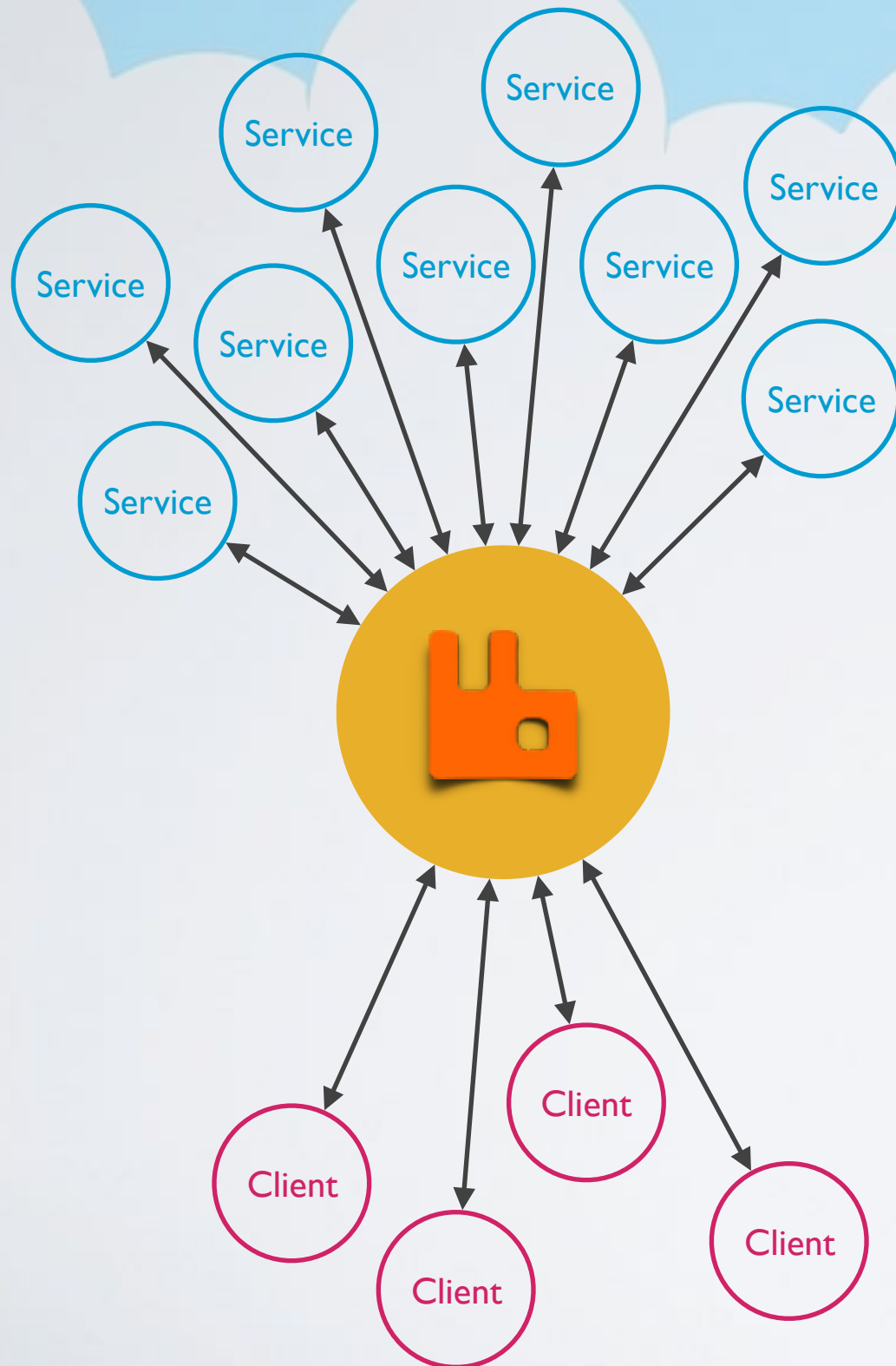
- System can be extended by new services or services can be modified without any downtime of the system.

Microservice Platform – **Spring.io**



- Lightweight framework to build Java Enterprise applications.
- Inversion of Control.
- Dependency injection (Decoupling components).
- A ton of libraries to facilitate the developers work.
- Excellent AMQP support.
- Excellent MongoDB support.
- Open source.

Message Broker – RabbitMQ



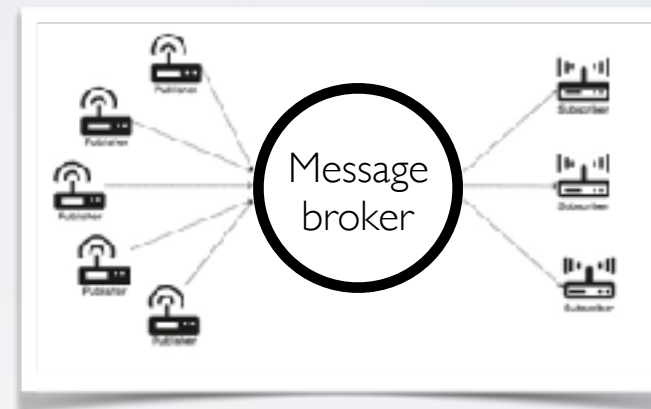
- Very powerful **AMQP** & **MQTT** (among others) message broker.
- Stable and broadly used in production (Instagram as an example).
- Flexible message routing.
- Scalable (high availability cluster).
- Open source.
- User authentication and access rights possible using multiple backends.

<http://www.rabbitmq.com>

Endpoint connectivity – MQTT

Message **Q**ueue **T**elemetry **T**ransport

- Simple message based protocol.
- Builds on top of TCP (SSL).
- Publish-Subscribe architecture.

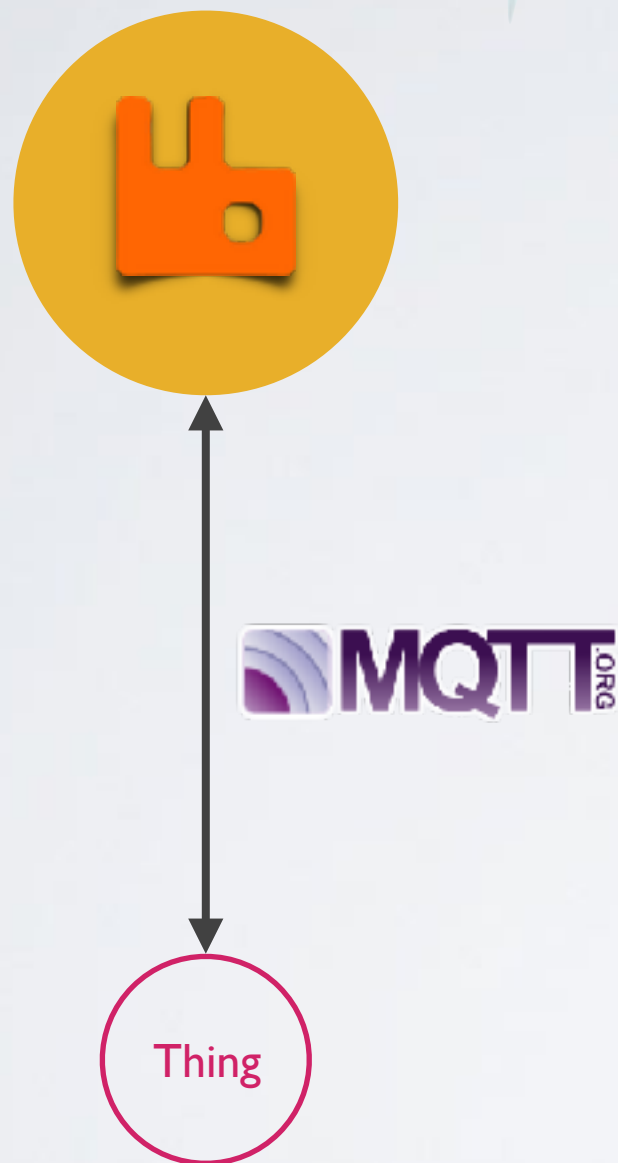


- Binary protocol with minimal overhead.
- Designed for low bandwidth and unreliable networks.
- Data agnostic.
- Open source.

<http://mqtt.org>



Endpoint connectivity – MQTT (2)



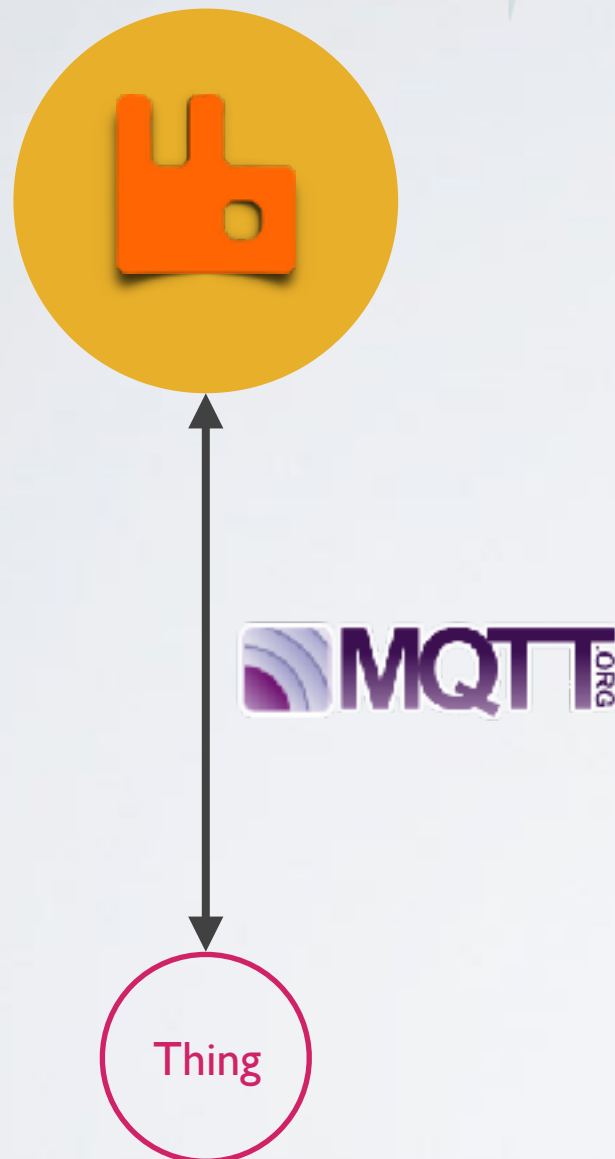
• Use Cases

- Push instead of poll.
- Bandwidth is a premium.
- Possible to interact with enterprise applications.
- Reliable delivery of messages over unreliable networks.
- Constrained devices.
- Low latency.

• Features

- Topic Wildcards.
- 3 Quality of service levels:
- Retained messages.
- Last will and testament.
- Persistent sessions.

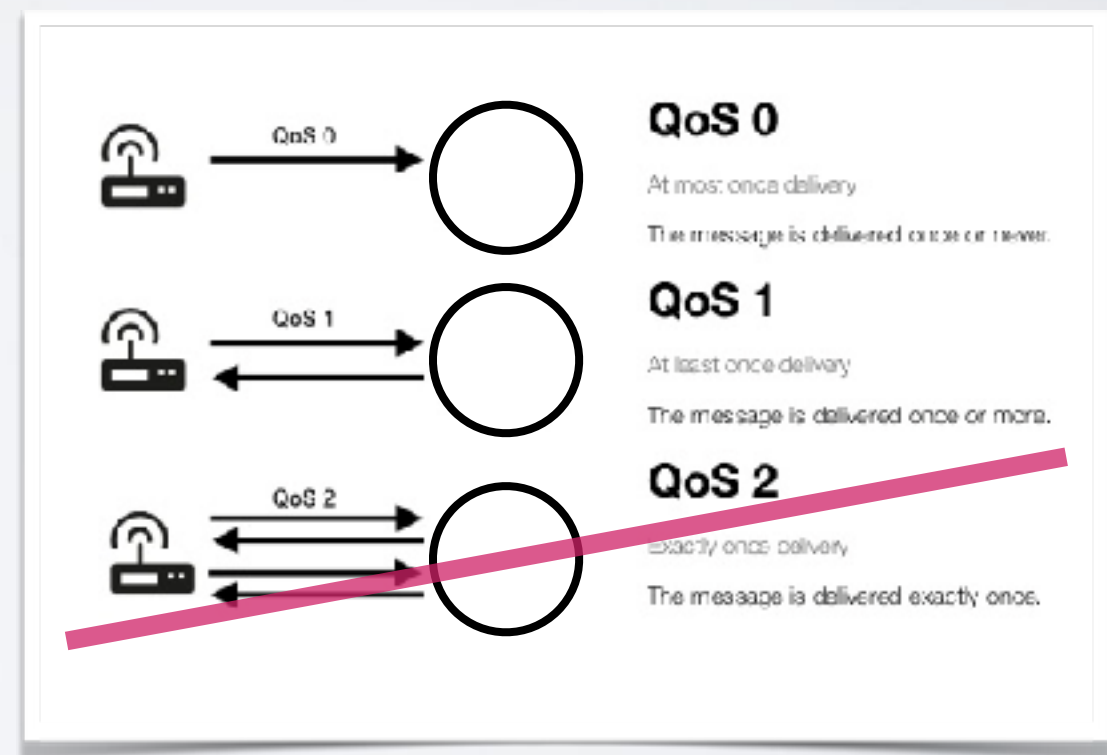
Endpoint connectivity - MQTT (3)



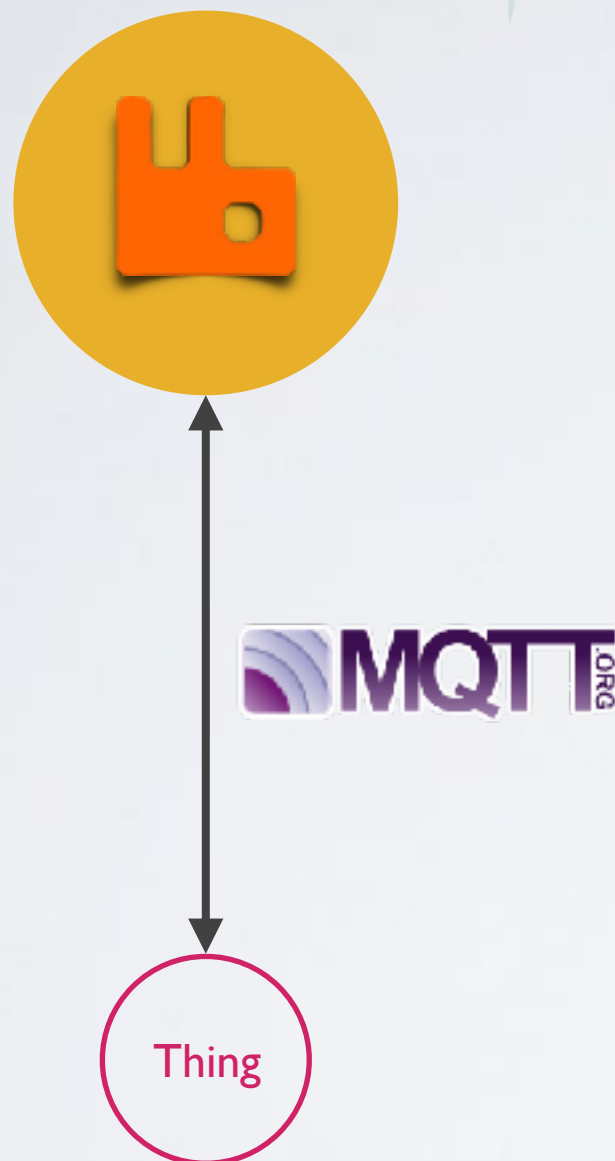
- **Security**

- Username | Password authentication.
- x509 certificate based authentication using TLS.
- Payload encryption using TLS.

- **Quality of Service**



Endpoint connectivity – MQTT (4)



- **Last will and testament**

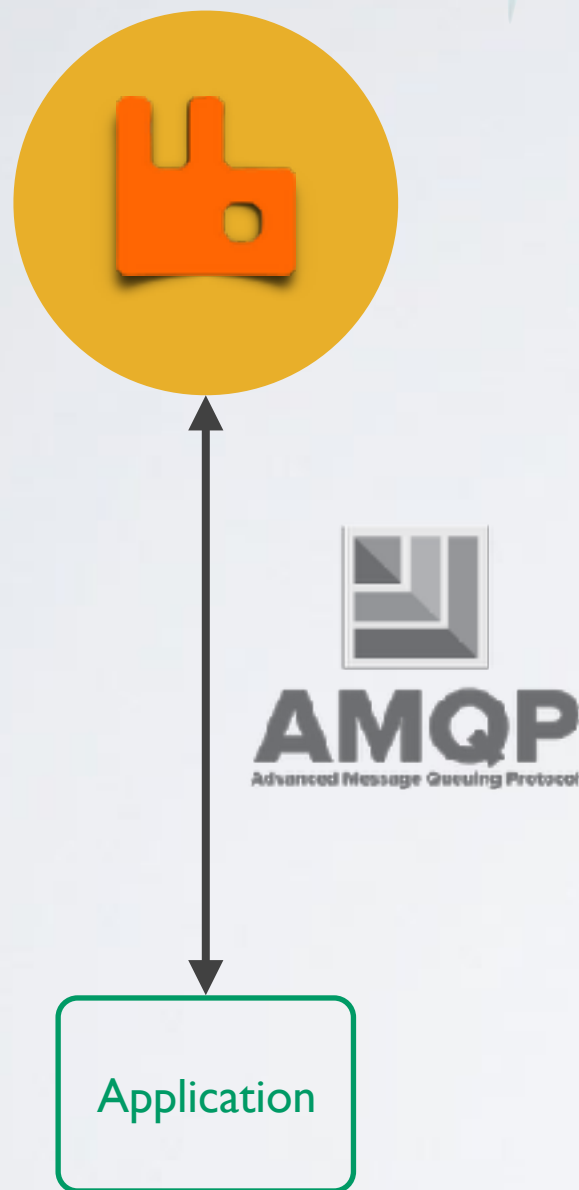
- Clients can specify a Last will and testament.
- Broker publishes the LWT message on behalf of the client on „death“.
- Useful for reporting problems.
- Real push on device „death“.
- Mostly used for reporting the connection status of a publisher.

- **Client libraries**



- paho project
 - C, C++, Go, Java, Javascript, Lua, Python
- Other languages
 - ActionScript, Clojure, Dart, Delphi, Erlang, Node.js, LotusScript, .NET, Objective-C, Perl, PHP, REXX, Ruby, Tcl
- Devices
 - Arduino, mbed, Nonode, Netduino

Application connectivity – **AMQP**

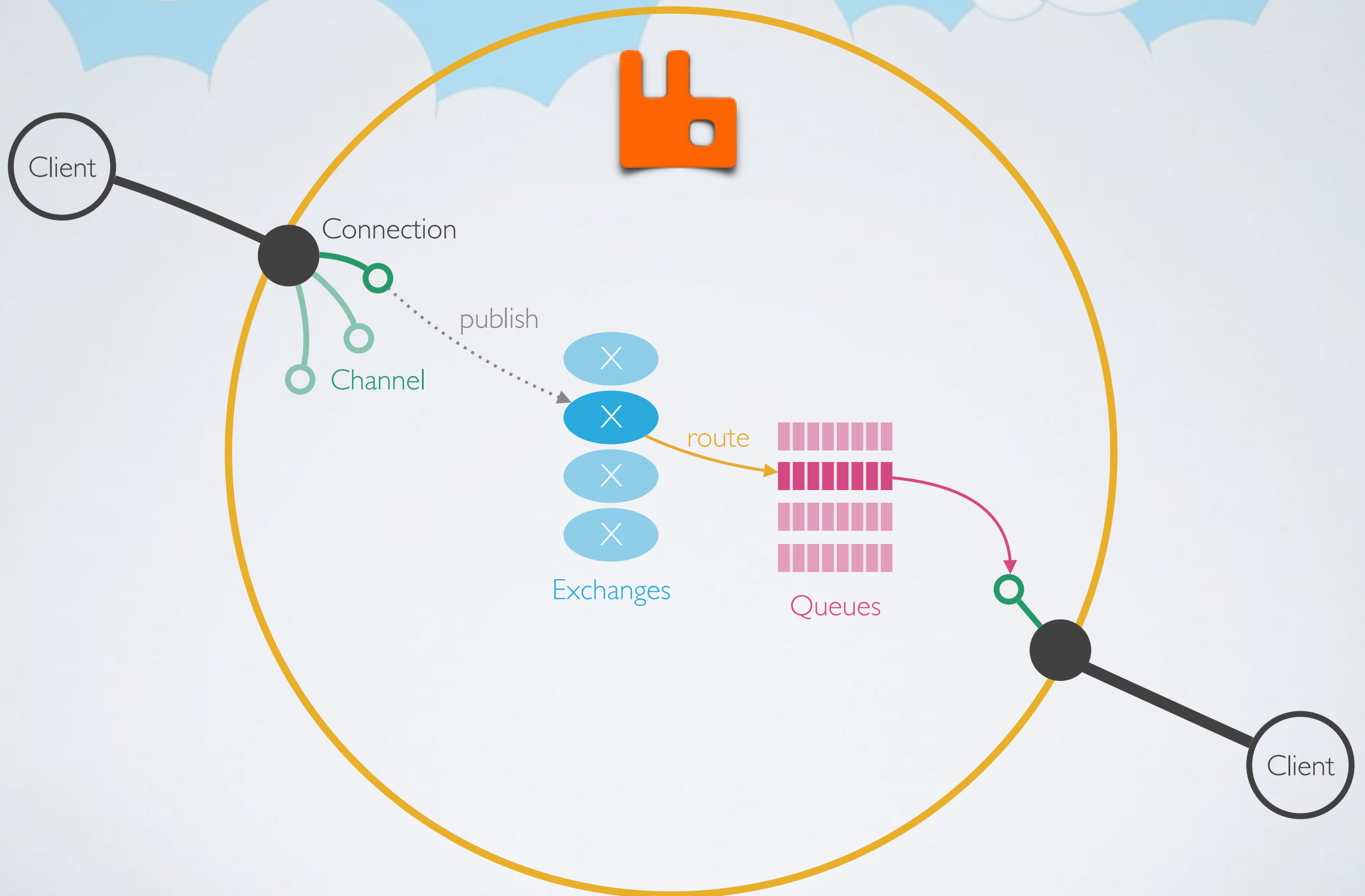


Advanced **M**essaging **Q**ueue **P**rotocol

- Open standard for passing business messages between different applications / processes.
- Can be used to scale the system (multi-process, multi-host).
- Very flexible message routing and queuing.
- Queuing of messages can be used to avoid the need for buffer DB's for load peaks.
- Mixing of different programming languages possible in order to be able to use an adapted language for a given job.
- **RabbitMQ** uses AMQP at its core.

<http://www.amqp.org>

Application connectivity - **AMQP** (2)



Topic Syntax – **AMQP** vs. **MQTT**



toto.lala.doe

toto/lala/doe

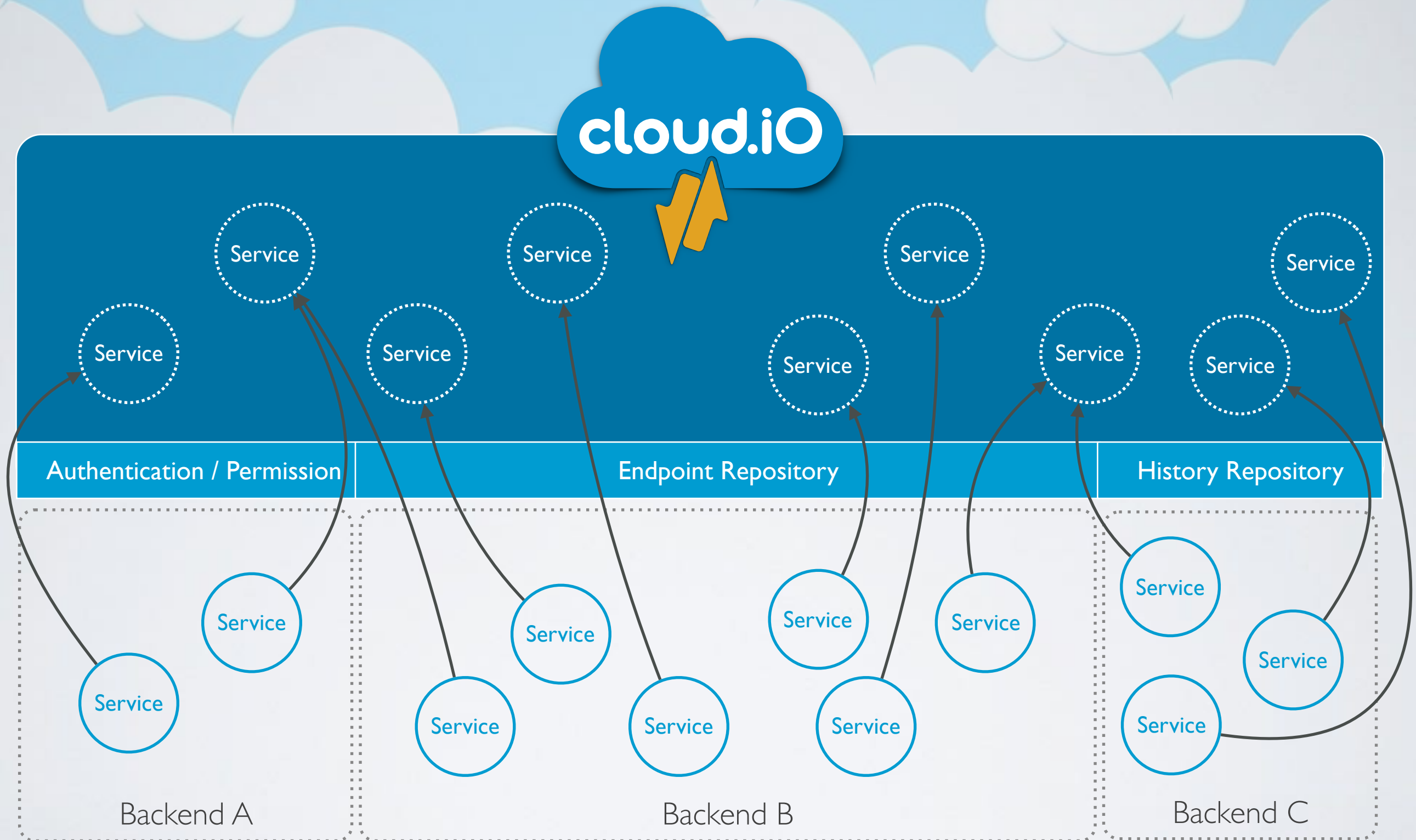
toto.*.doe

toto/+ /doe

toto.#

toto/#

Backends



Backends (2)

cloud.iO



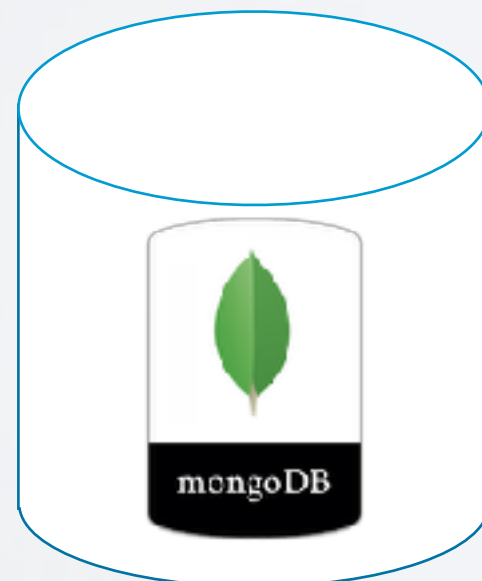
Authentication / Permission

Endpoint Repository

History Repository

MongoDB
Backend

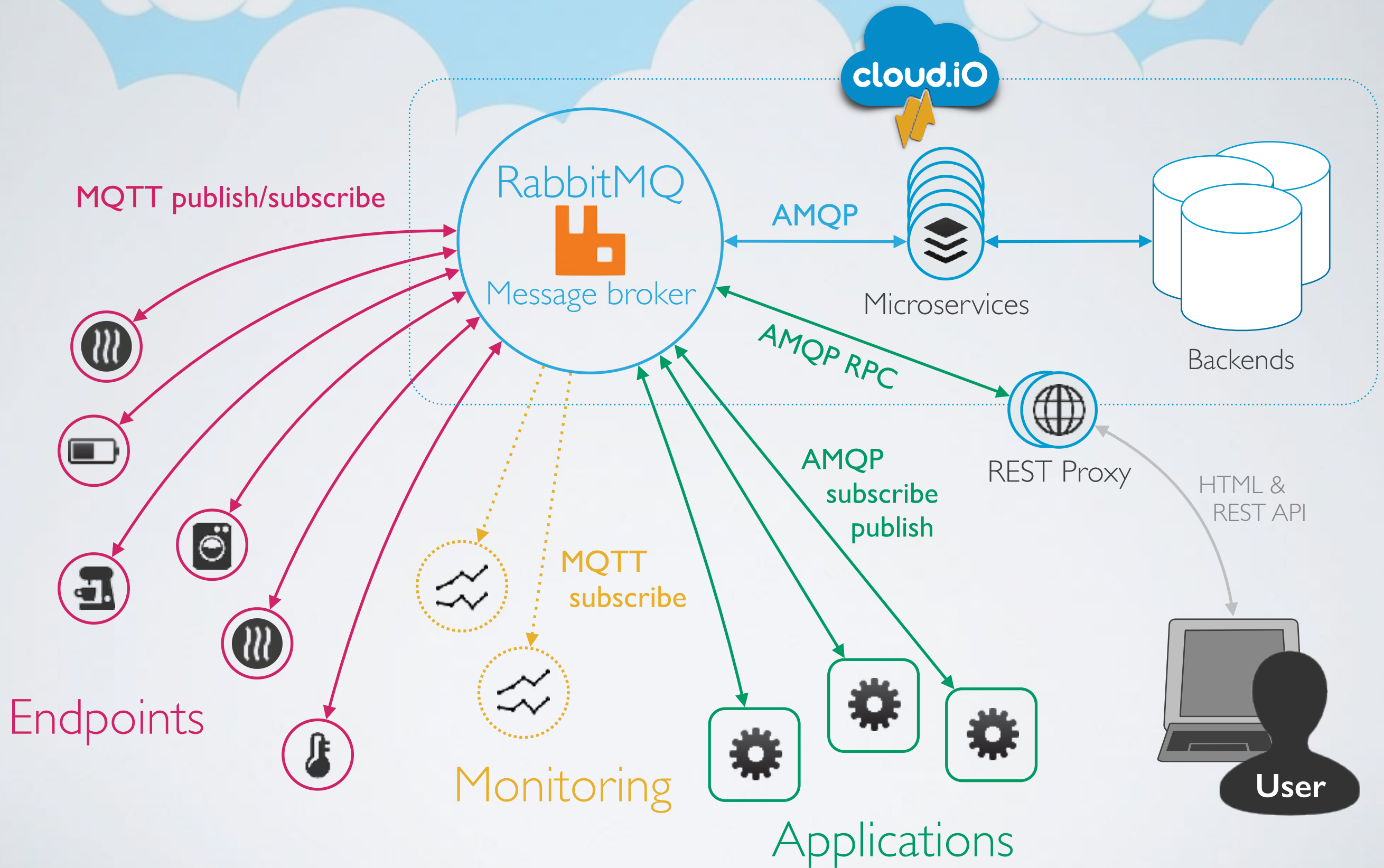
InfluxDB
Backend





cloud.iO Architecture

Overview



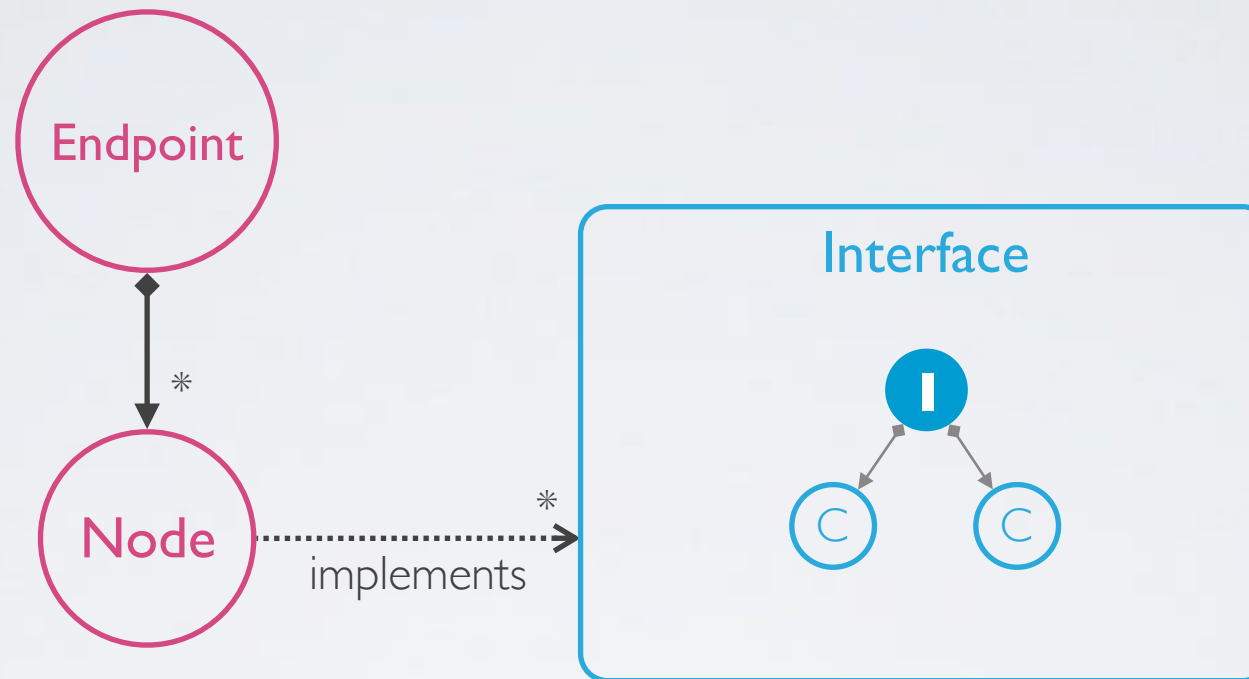


Endpoint structure



Node interfaces

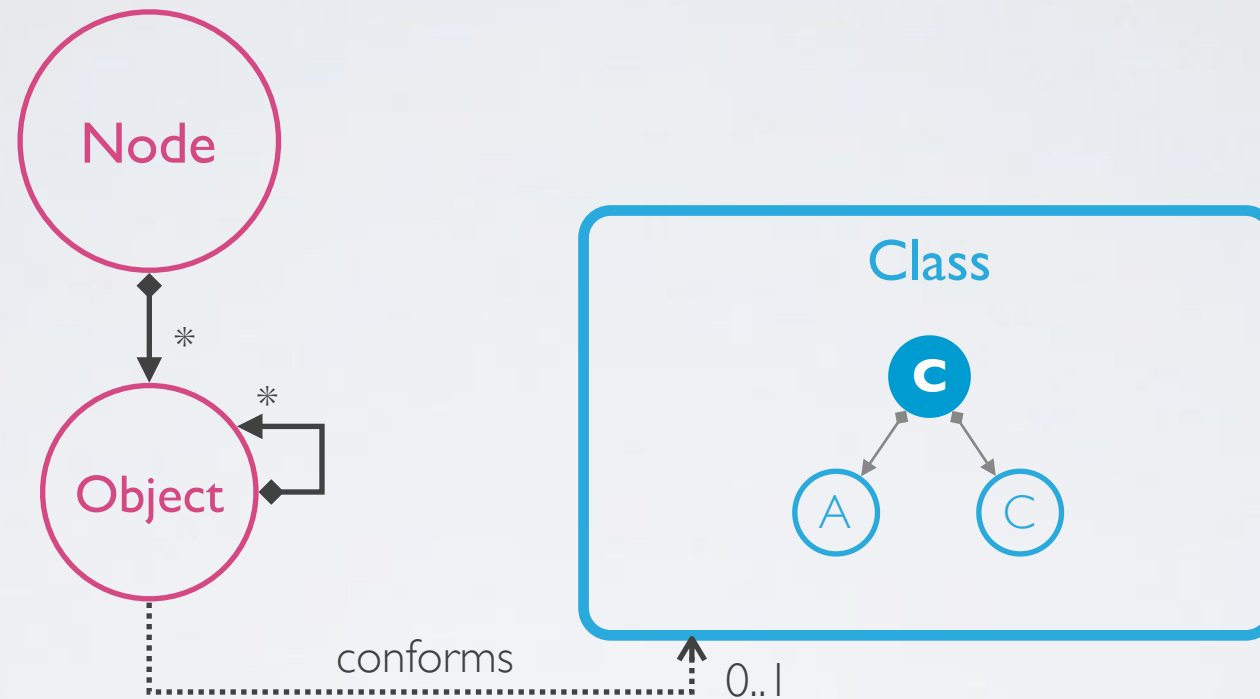
- Node structure is theoretically completely free, classification can be done using **interface** definitions:



- An **interface** defines which **object classes** a given node has to offer among others.
- A node can implement any number of **interfaces**.
- The actual structure of a node is fix and can not be changed once the node has been added to the Endpoint.

Object classes

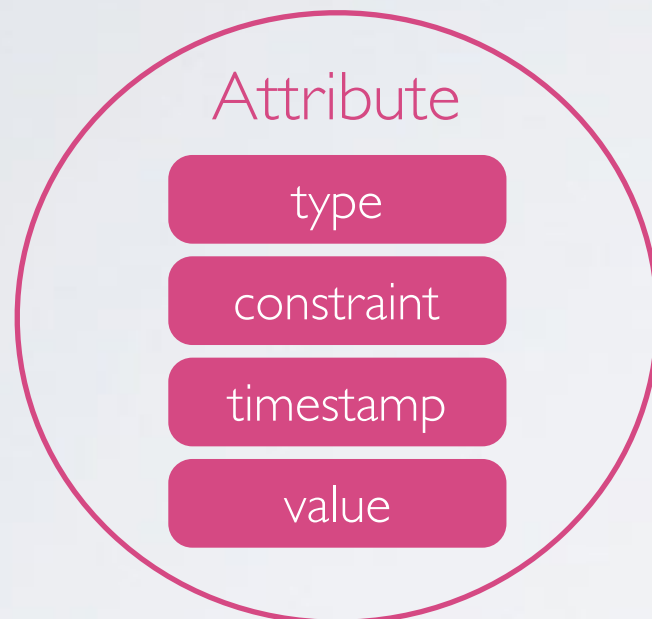
- In order to enable the reuse of object definitions, **classes** for common object structures can be defined:



- A **class** defines the exact structure of child **object classes** and **attributes** a given object has to be composed of.
- An object can only conform to no or exactly one **class**.
- Classes can contain other classes, this enables hierarchical structures.

Attributes

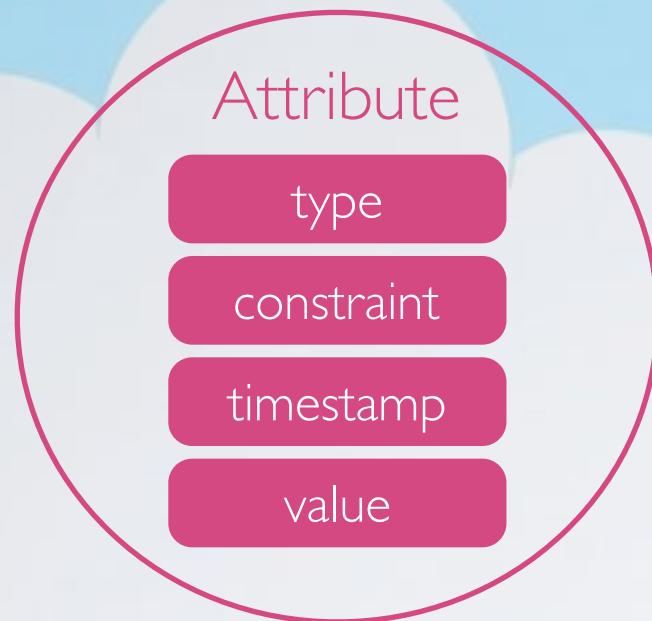
- Attributes are the very atomic elements objects are made of.
- Attributes always have four fields:



```
{  
  „type“: „Number“,  
  „constraint“: „Measure“,  
  „timestamp“: 45878945.457,  
  „value“: 19.2  
}
```

- The constraint defines the actual kind of an attribute:
 - **Static**: Static value, never changes, read-only. For example manufacturer, model, serial number...
 - **Status**: Read-only value representing a state or a calculated value.
 - **Measure**: Read-only value representing an actual measure made by the device.
 - **Parameter**: Parameter that can be written to, changes the configuration of the device.
 - **SetPoint**: An output or a set point of a local regulation. Temperature threshold for example.

Attributes (2)



```
{  
  „type“: „Number“,  
  „constraint“: „Measure“,  
  „timestamp“: 45878945.457,  
  „value“: 19.2  
}
```

- The timestamp is in seconds since epoch
 - In order to support sub-second resolution of time, the timestamp can be optionally of floating point type.
- The value has to be of a given type:
 - Basic Types: **Bool**, **Integer**, **Number**, **String**
 - Complex Types: **Bool[]**, **Integer[]**, **Number[]**, **String[]**

Topic / Content relation

Topic:

@UPDATE/D640cdd70-1bdb-11e4-8c21-0800200c9a

```
{
  „uuid“: „640cdd70-1bdb-11e4-8c21-0800200c9a66“,
  „nodes“: {
    „Meteo“: {
      „implements“: [„MeteoStation“],
      „objects“: {
        „temperatures“: {
          „conforms“: null,
          „objects“: {
            „inside“: {
              „conforms“: „TemperatureMeasure“,
              „attributes“: {
                „unit“: {
                  „constraint“: „Static“,
                  „timestamp“: null,
                  „value“: „K“
                },
                „temperature“: {
                  „type“: „Number“,
                  „constraint“: „Measure“,
                  „timestamp“: 45878945.457,
                  „value“: 19.2
                }
              }
            }
          }
        }
      }
    }
  }
}
```

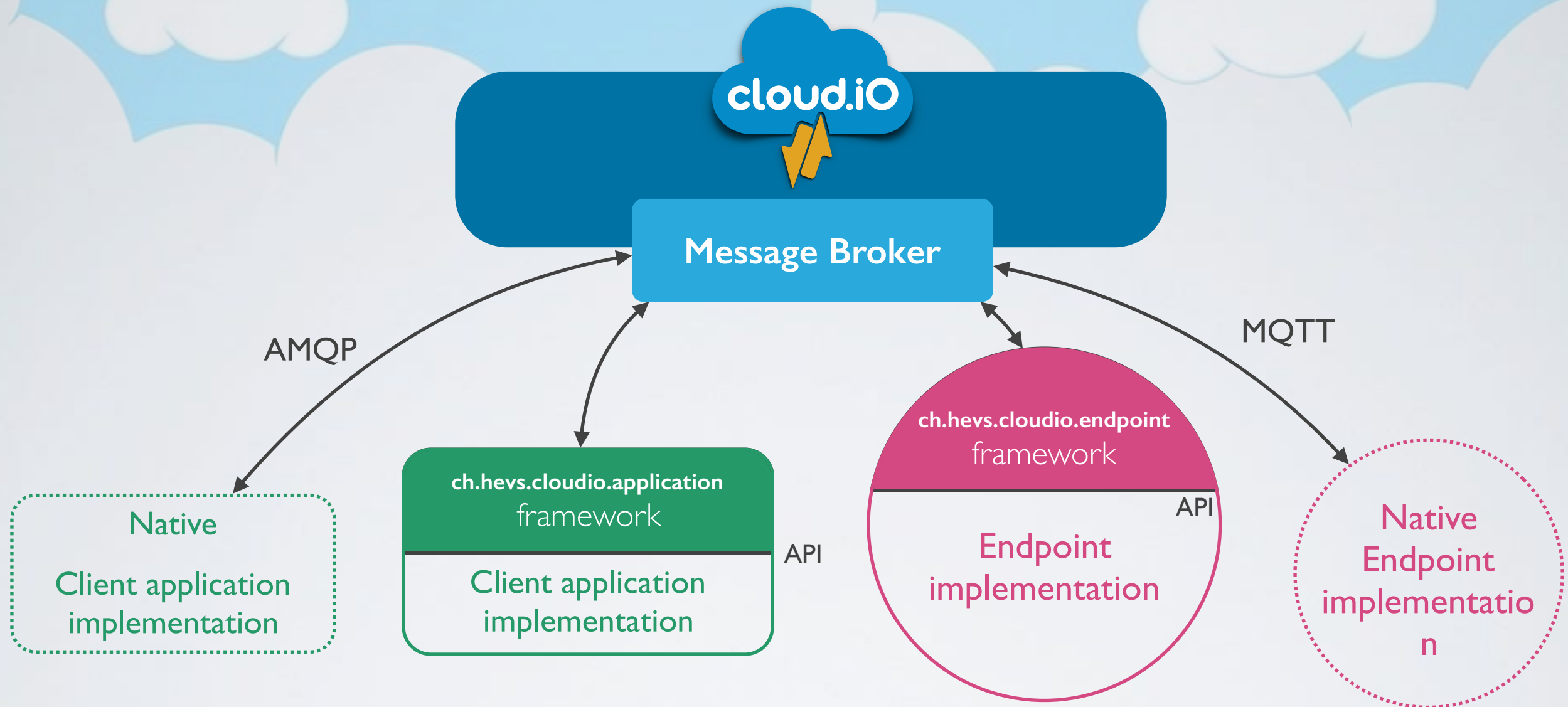
Topic:

@UPDATE/D640cdd70-1bdb-11e4-8c21-0800200c9a/
nodes/Meteo/objects/temperatures/objects/inside/
attributes/temperature

```
{
  „TYPE“: „Number“,
  „constraint“: „Measure“,
  „timestamp“: 45878945.457,
  „value“: 19.2
}
```

- It is possible to send just a subset of a device data model
- The topic reflects the path to the content in the complete device data model.

Endpoint & Application frameworks



- The messaging API between the actors in IoT cloud can be completely hidden by the application frameworks, however the messaging API is defined too in order to allow „native“ endpoints and applications.
- The application or endpoint developer does not necessary need to have any knowledge of the messaging system behind the scenes,.
- The messaging system can be changed at any time without the need to rewrite the application and endpoint software if the frameworks are used.



cloud.iO Status & Roadmap

Open source



<http://claudio.hevs.ch>

<https://github.com/claudio-project>



claudio-rabbitmq



claudio-cloud-microservice



claudio-cloud-backend-mongo-influx



claudio-endpoint-java



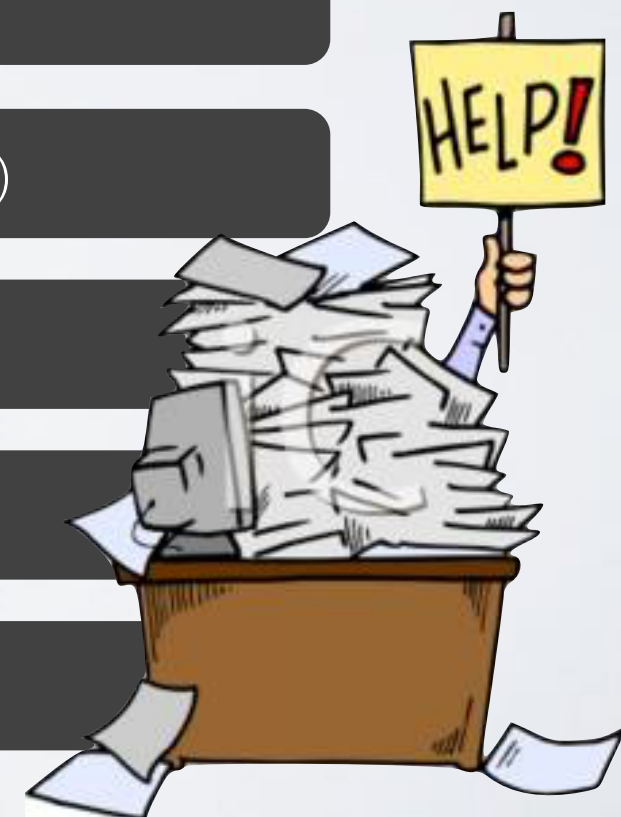
claudio-endpoint-osgi



claudio-endpoint-python

Status & Roadmap

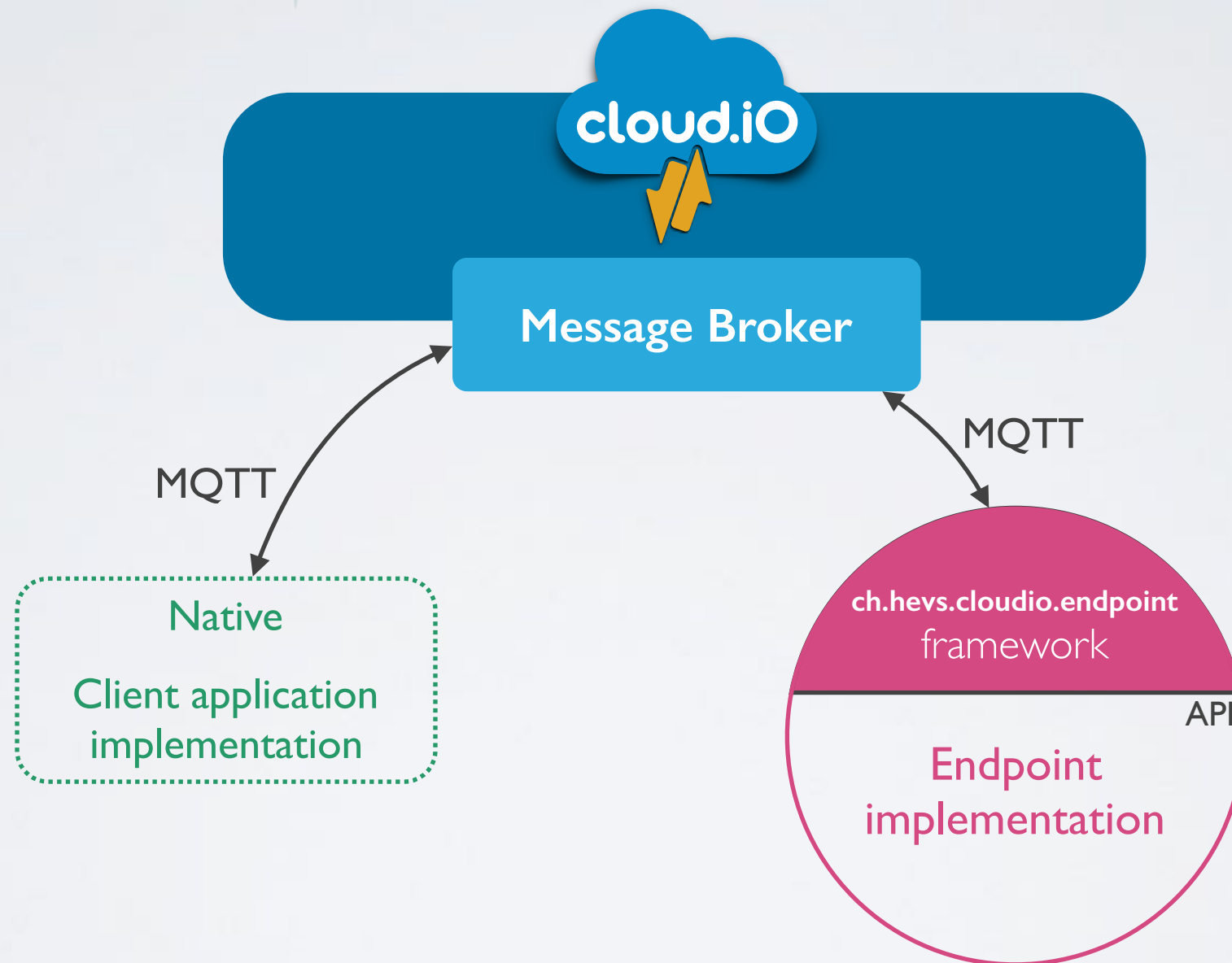
- ✓ Security/Privacy related modifications to RabbitMQ
- ✓ Proof of concept Cloud implementation
- ✓ Endpoint API (Java)
- ✓ Cloud implementation including default data backends
- ✗ Formal topic and message specification
- ✗ Application & Monitoring API (Java)
- API in other Languages (C, C++, [Python](#))
- ✗ Management REST API
- ✗ Common data classes library (Java)
- ✗ Public documentation and examples





cloud.iO Example

Example



Development infrastructure

- Single node cloud.iO stack using docker.io
 - RabbitMQ based message broker
 - MongoDB NoSQL database
 - InfluxDB Time series database
 - cloud.iO MongoDB/InfluxDB backend

```
# wget -q -O - http://bit.ly/cloudio-sn | bash
```

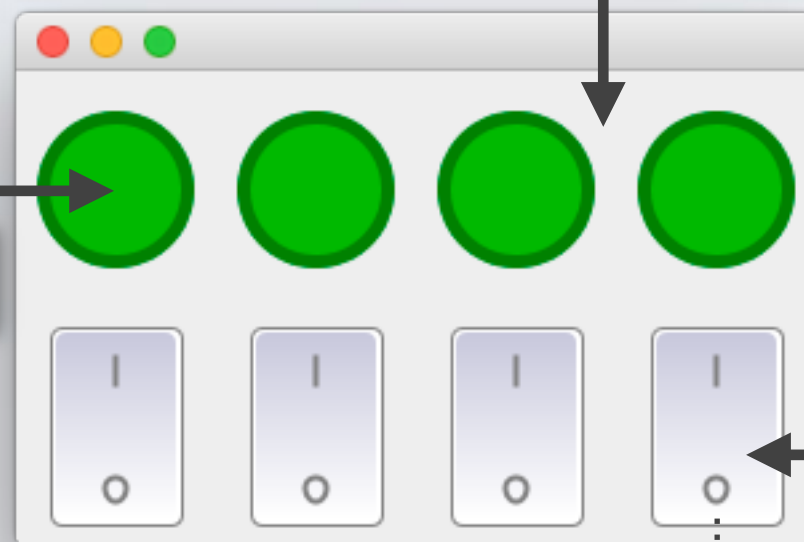
Sample Application

JIoBoxFrame

```
public JIoBoxFrame(final int ioCount)
public JLed getLed(int index)
public JSwitch getSwitch(int index)
```

JLed

```
public void setState(boolean state)
```



JSwitch

```
public void setListener(JSwitchListener listener)
```

JSwitchListener

```
public interface JSwitchListener {
    void stateChanged(JSwitch jswitch);
}
```



Endpoint API Library



- One single artifact (Jar file)
- Very few dependencies
 - Paho
 - Jackson-core
 - Log4j



```
dependencies {  
    compile 'ch.hevs.cloudio:cloudio-endpoint-java:0.1.1'  
}
```



Settings

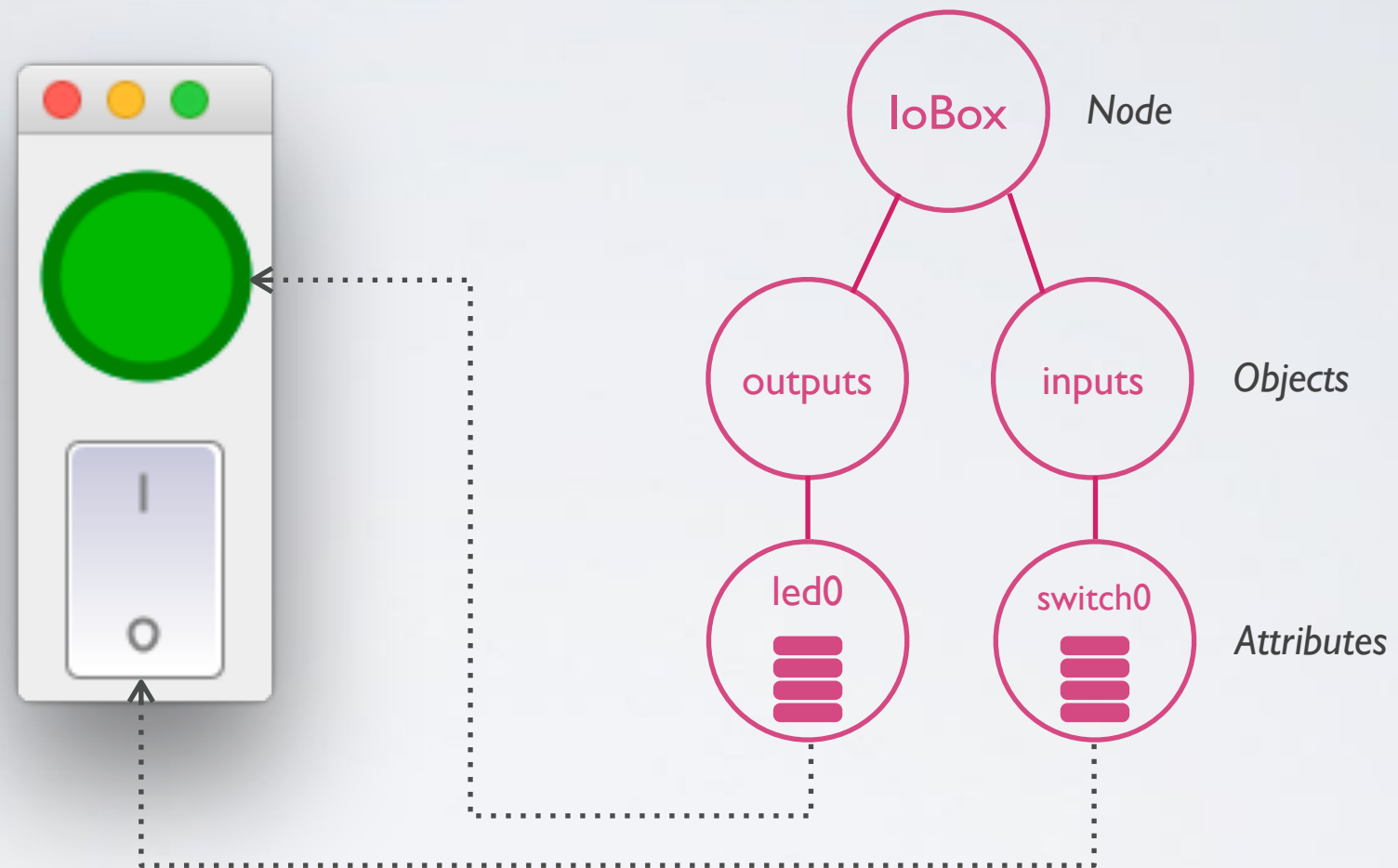


- **Loads configuration from Java Properties file**
 - ~/.config/cloud.io/{Endpoint UUID}.properties
 - /etc/cloud.io/{Endpoint UUID}.properties
 - cloud.io/{Endpoint UUID}.properties inside the application bundle

```
ch.hevs.cloudio.endpoint.hostUri = ssl://213.221.143.116:8883  
ch.hevs.cloudio.endpoint.ssl.authorityPassword = cloudio
```

- **Endpoint certificate is searched here:**
 - ~/.config/cloud.io/{Endpoint UUID}.p12
 - /etc/cloud.io/{Endpoint UUID}.p12
 - cloud.io/{Endpoint UUID}.p12 inside the application bundle
- **Certificate authority certificate is searched here:**
 - ~/.config/cloud.io/authority.jks
 - /etc/cloud.io/authority.jks
 - cloud.io/authority.jks inside the application bundle

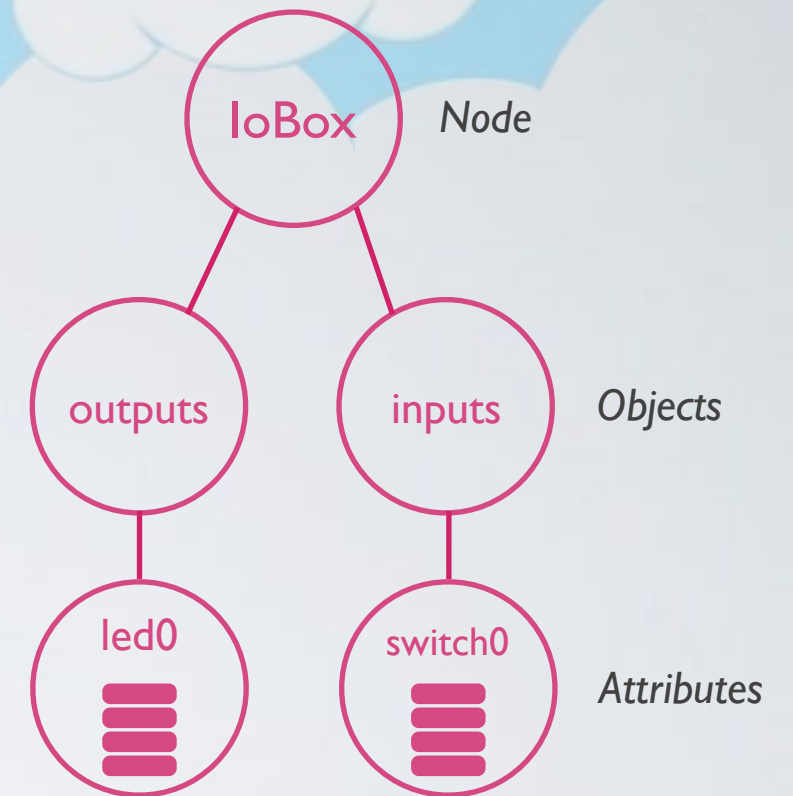
Data Model



Data Model (2)



```
@Implements(value = {"MyInterface"})
public class IoBox extends CloudioNode {
    public Inputs inputs;
    public Outputs outputs;
}
```



```
@Conforms("SingleLedOutput")
public class Outputs extends CloudioObject {
    @SetPoint
    public CloudioAttribute<Boolean> led0;
}
```

```
@Conforms("SingleSwitchInput")
public class Inputs extends CloudioObject {
    @Measure
    public CloudioAttribute<Boolean> switch0;
}
```


Application

```
public class IoBoxExample {
    public static void main(String... args) {
        try {
            final CloudioEndpoint endpoint = new CloudioEndpoint("test");
            final IoBox box = endpoint.addNode("io", IoBox.class);

            final JIoBoxFrame frame = new JIoBoxFrame(1);

            frame.getSwitch(0).setListener(new JSwitchListener() {
                @Override
                public void stateChanged(JSwitch jswitch) {
                    try {
                        box.inputs.switch0.setValue(jswitch.getState());
                    } catch (CloudioAttributeConstraintException e) {
                        e.printStackTrace();
                    }
                }
            });

            box.outputs.led0.addListener(new CloudioAttributeListener<Boolean>() {
                @Override
                public void attributeHasChanged(CloudioAttribute<Boolean> attribute) {
                    frame.getLed(0).setState(attribute.getValue());
                }
            });

            frame.setVisible(true);

        } catch (InvalidUuidException | InvalidPropertyException |
            CloudioEndpointInitializationException | DuplicateNamedItemException e) {
            e.printStackTrace();
        }
    }
}
```



Questions?