R





**Integration HRIS EMPLOYERS - Mon Compte Mobilité**

**Functional and technical specifications**





Versioning

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description |
| 0.1 | 2021-12-29 | Capgemini | First version to evaluate the effort |
| 1.0 | 2022-01-24 | Capgemini | First complete version |
| 1.1 |  | Capgemini | Modify queues names and number of attachment links |
| 1.2 | 2022-05-22 | Capgemini | Add encryption key to incoming flow |
| 1.3 | 2022-06-24 | Capgemini |  |
| 1.4 | 2022-09-05 | Capgemini | Amount field is optional in all cases  Warning on the encoding response for attachments endpoint |

Related documentation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Version | Date | Author | Description |
| 20220221\_Solution\_Chiffrement\_Dossiers\_Souscriptions\_V1.3.pdf | 1.3 | 13/05/2022 | Capgemini | Encryption and decryption specification |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of content

[1. Context and objectives 3](#_Toc106983205)

[1.1. Context 3](#_Toc106983206)

[1.2. Objectives 3](#_Toc106983207)

[2. Functional workflows 4](#_Toc106983208)

[2.1. Incoming Flow 4](#_Toc106983209)

[2.1.1. File attachments / links 5](#_Toc106983210)

[2.2. Outgoing Flow 6](#_Toc106983211)

[2.2.1. Case of validation 6](#_Toc106983212)

[2.2.2. Case of rejection 6](#_Toc106983213)

[2.2.3. Case of error 7](#_Toc106983214)

[2.2.4. Status correspondence HRIS business rules 7](#_Toc106983215)

[3. Error use cases 7](#_Toc106983216)

[4. Technical specifications 8](#_Toc106983217)

[4.1. Requirements 8](#_Toc106983218)

[4.1.1. Authentication 8](#_Toc106983219)

[4.1.2. Storage and resilience 8](#_Toc106983220)

[4.1.3. Protocol access 8](#_Toc106983221)

[4.1.4. Target volumetry 8](#_Toc106983222)

[4.1.5. Confidentiality and encryption 8](#_Toc106983223)

[4.1.6. Traceability 9](#_Toc106983224)

[4.1.7. Extensibility 9](#_Toc106983225)

[4.1.8. Security 9](#_Toc106983226)

[4.2. Solution Architecture 9](#_Toc106983227)

[4.2.1. Message bus software 10](#_Toc106983228)

[4.2.2. Queueing architecture and exchange type 11](#_Toc106983229)

[4.2.3. Supported and preferred protocols 12](#_Toc106983230)

[4.2.4. Other solutions studied 12](#_Toc106983231)

[4.3. Security 12](#_Toc106983232)

[4.3.1. Identity management, identification and authentication 12](#_Toc106983233)

[4.3.2. Authorization and access control 14](#_Toc106983234)

[4.3.3. Integrity 15](#_Toc106983235)

[4.4. Message formats 15](#_Toc106983236)

[4.4.1. Incoming flow 15](#_Toc106983237)

[4.4.2. Outgoing flow 16](#_Toc106983238)

[4.5. Publish a message 16](#_Toc106983239)

[4.5.1. Using AMQ client 17](#_Toc106983240)

[4.5.2. Using HTTP REST API 17](#_Toc106983241)

[4.6. Receive a message 19](#_Toc106983242)

[4.6.1. Using AMQP client 19](#_Toc106983243)

[4.6.2. Using HTTP REST API 20](#_Toc106983244)

[4.7. Download file attachments 21](#_Toc106983245)

[4.8. Environments 21](#_Toc106983246)

[4.8.1. Préproduction 21](#_Toc106983247)

[4.8.2. Production 22](#_Toc106983248)

# Context and objectives

## Context

MOB is an award-winning project that received the support of the French Ministry of Ecological Transition. The project has started in April 2020 and will end in December 2022. It is led by Capgemini and La Fabrique des Mobilités. MoB aims at creating a digital mobility incentive management platform.

During the project phase, MOB is a non-commercial platform that helps to speed up change towards green mobility by easing incentive management and mobility policy management between market players. It also aims at reducing overall C02 footprint emissions by encouraging green mobility incentives subscription.

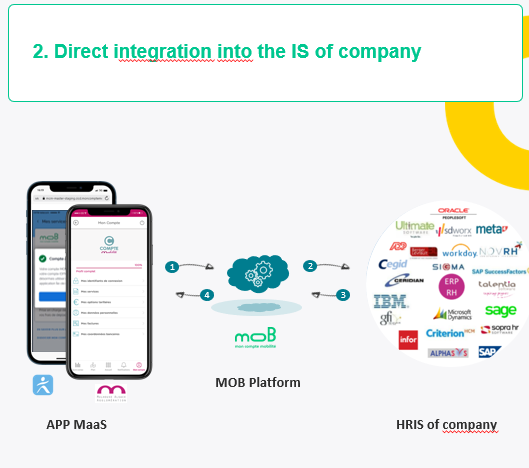
Among these market players, local government authorities and companies set up green mobility incentives for their employees.

MOB is the tool that enables them to:

* Steer the incentive policy by recovering usage data on the most popular incentives and following up mobility budget
* Showcase and manage mobility incentives allocation
* Benefit from MOB interfacing with mobility players for the implementation of mobility policy

The main goal is to simplify the process of validating incentives requests (also called **subscriptions**) thanks to the certification of trips by the direct transmission of purchased certification documents from the transport operator.

In order to manage incentive requests, Capgemini has chosen to integrate MOB directly with the HRIS of the companies to maintain the existing managerial path.



Companies that do not have any HR system in place can use the dedicated MOB interface to manage their incentive requests. Thus, they are not concerned by this specification and don’t use the followed solution.

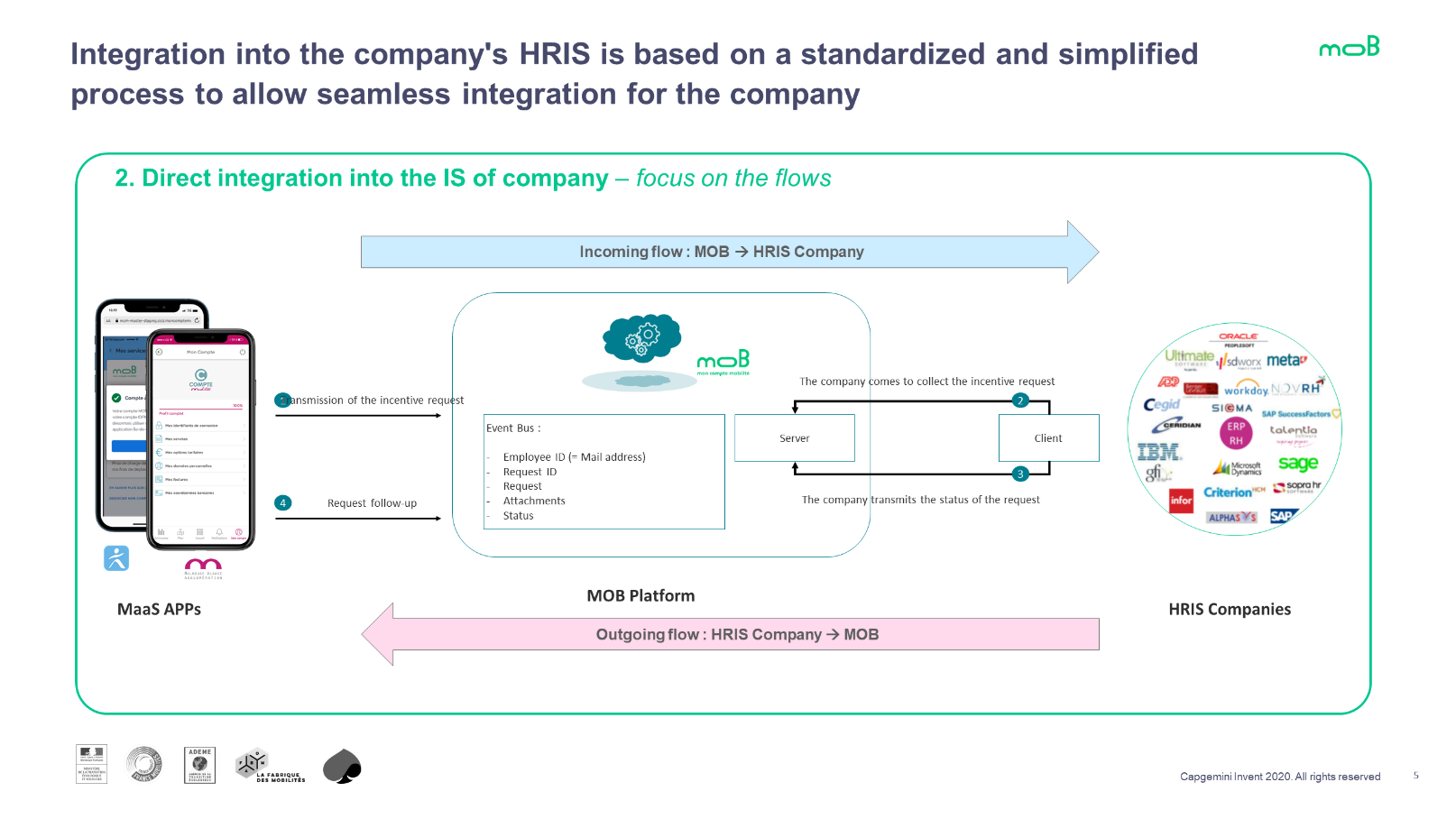
## Objectives

The purpose of this document is to provide both functional and technical description related to the Message Bus (Event Bus), which will be implemented from MOB’s side in order to allow HRIS to receive their incentive request’s information as incoming Flow and give the possibility to the HRIS managers to send back their response as an outgoing Flow.

The implementation of an event bus from MOB side that is based on a standardized and simplified process that solves the problem of how microservices communicates with each other, which will allow a smooth and easier integration with a company’s HRIS.

# Functional workflows

The goal of this chapter is to give a functional definition of the overall data that will be sent through the Event Bus after the user subscribes to an incentive. 2 flows are necessary to cover the business needs.



The event bus server is on MOB platform. On client side, we have:

* A client on the backend MOB application
  + It publishes the details of the incentive request
  + It collects the status updates for all incentive requests and updates the database with these informations
* A client per HRIS linked with MOB
  + It collects new incentive requests for her company and integrates them into his HR process
  + It publishes the details of the HR feedback about incentive requests received

## Incoming Flow

After issuing the incentive request from the user, MOB publishes a message containing all the request’s information on a communication channel of the event bus:

|  |  |  |
| --- | --- | --- |
| Inputs | Mandatory | Format |
| Last Name | **Yes** | String |
| First Name | **Yes** | String |
| Date of birth | **Yes** | Datetime  Sample: 1993-06-25T00:00:00.000Z |
| Citizen ID | **Yes** | String |
| Incentive ID | **Yes** | String |
| Subscription ID | **Yes** | String |
| Professional Email | **Yes** | Standard email Format  Domain name to indicate in advance by the company (@company.com, @domain.com...)   Email inserted by the employee while subscripting and verified by the validation link sent. |
| Community Name | **No** | Name of the citizen’s community (only if configured for the funder). It can permit to HRIS to route correctly the request. |
| Attachments | **Yes** | Array of secured URL links.  **One link per subscription attachment, to download it.**  **Each attachment is encrypted**. It is only decryptable with the funder’s private key.  Maximum number of attachments: 10 Format: PDF, JPEG, JPG, PNG |
| Status | **Yes** | Status “A TRAITER” or “ERREUR” |
| Specific fields | **Yes** | A JSON object with fields and values filled by the citizen.  The incentives are recreated on MOB the same way they were created on the HRIS side. With the same fields declared  (**Same Name, Same Format**)  In case of an unknown field by the HRIS, it should be inserted by the HRIS into a free text field. |
| Encrypted AES Key | **Yes** | String Base64  The encrypted symmetric key used to encrypt the attachment.  Essential to decrypt with the funder's private key |
| Encrypted Initialization Vector | **Yes** | String Base64  The encypted [initialization vector](https://en.wikipedia.org/wiki/Initialization_vector) used to encrypt the attachment.  Essential to decrypt with the funder's private key |

### File attachments / links

Please note, there is no direct sending of supporting documents in the event bus. Secure URL links are provided into the incentive request flow and allows HRIS to download encrypted attachments.

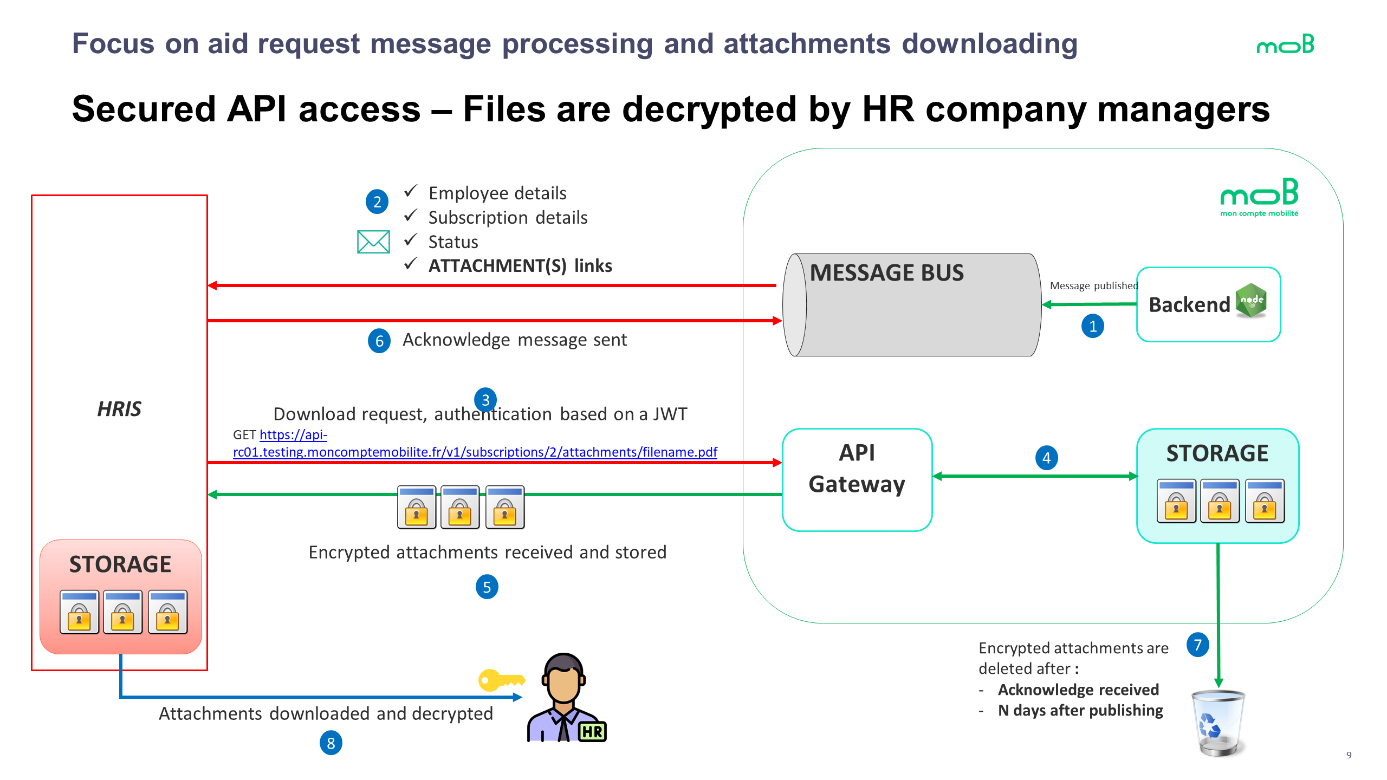
In fact, attachments can be sensitive documents like identity card scan for example and it is necessary to secure its.

Attachments can be decrypted only by HRIS managers. They must have the private key essential to open and read attachments in clear.

The HRIS must download all files using the links provided, at the same time just after getting the incentive request message.

The HRIS sends an acknowledge to MOB to confirm the message processing and the correct download of the attachments. It must store them on his storage and attach them to the request created in his case management system.

Once the message is acknowledged by HRIS or after a predefined period (to be defined), MOB considers that the processing of the incentive request is under the HRIS responsibility. MOB can delete quickly then all files for data regulation rules.



## Outgoing Flow

The goal of this service is to allow the HRIS managers to send a response through the bus as a message containing all information related to their decision after processing the incentive request, whether if request is validated or rejected.

### Case of validation

In case of the validation of the request, the following information are sent as a response:

|  |  |  |
| --- | --- | --- |
| Input | Mandatory | Format |
| Citizen ID | **Yes** | String. MOB internal ID given in the incoming flow. |
| Subscription ID | **Yes** | String. MOB internal ID given in the incoming flow. |
| Status | **Yes** | String “**VALIDEE**” |
| Payment Method (mode) | **Yes** | String 3 possible choices   * No payment ‘**aucun**’ (if no information available, use it) * Single payments ‘**unique**’ * Multiple payment ‘**multiple**’ |
| Comments | **No** | Text with a maximum limit of 4000 characters |
| Amount | No | Number. |
| Frequency | No | String “mensuelle” or “trimestrielle” |
| Last Payment | No | Date “2022-01-30” |

### Case of rejection

In case of an incentive request’s rejection, the following information are sent as a response:

|  |  |  |
| --- | --- | --- |
| Input | Mandatory | Format |
| Citizen ID | **Yes** | String. MOB internal ID given in the incoming flow. |
| Subscription ID | **Yes** | String. MOB internal ID given in the incoming flow. |
| Status | **Yes** | STATUS «REJETEE » |
| Rejection type | **Yes** | String  4 possible choices :   * “ConditionsNonRespectees” * “JustificatifManquant” * “JustificatifInvalide” * “Autre”   If there is no cause of rejection field available in the HRIS, so as of now there will be no correspondence with MOB. Thus, consider using the case “Autre” and fill the “other” field. |
| Comments | No | Text with a maximum limit of 4000 characters |
| Other Comment | No | Mandatory if the rejection type is ”Autre”  HRIS can transmit the reason with this field |

### Case of error

For various reasons, the processing of the incentive request may be in error on HRIS side.

In this scenario, the following information are sent as a response:

|  |  |  |
| --- | --- | --- |
| Input | Mandatory | Format |
| Citizen ID | **Yes** | String. MOB internal ID given in the incoming flow. |
| Subscription ID | **Yes** | String. MOB internal ID given in the incoming flow. |
| Status | **Yes** | STATUS «ERREUR » |

The citizen will then be informed that his request could not be processed by the funder. He may be asked to renew his request.

### Status correspondence HRIS business rules

|  |  |
| --- | --- |
| **BR1**: | In case of a request in the status “awaiting manager decision” there will be no communication with MOB platform |
| **BR2:** | The status « Cancelled by employee» is not possible. |
| **BR3:** | The status “To be completed by the employee” is not possible, as the user cannot complete or update his request through MOB |
| **BR4:** | Adding an “Origin” flag to the incentive request received by the HRIS, in order to specify that this request is coming from MOB |
| **BR5:** | Do not display the button “To be completed by the employee” when the request is coming from MOB |

# Error use cases

In the table below, a list of few error cases which the user/system might face, and the solution suggested accordingly:

|  |  |
| --- | --- |
| Description | Proposition |
| Case 1: Unknown Employee by the HRIS | Rejecting the request with a specific error message  « Unknown Employee » |
| Case 2: A Known Employee, but outside of the experimentation’s scope | The case can still be managed by the company, without impact on the employee. |
| Case 3: Communication Problems | Treatment to be defined during the tests |
| Case 4: Unknown status for MOB sent by the company | Keep the request open, to be manually processed by the manager on HRIS |

# Technical specifications

## Requirements

### Authentication

All HRIS are authenticated in the MOB IAM solution.

### Storage and resilience

No loss of message is tolerated, the messages are persisted until the messages are consumed by the receiver. The system must be redundant.

### Protocol access

The solution must allow access to as many HRIS as possible, by offering widespread and state-of-the-art protocols. It must be easy to implement a client to interface with the solution, with most languages on the market.

### Target volumetry

Period: **monthly**

|  |  |  |  |
| --- | --- | --- | --- |
| PARAMETERS | MVP (Phase 1) | Low target (Phase 2) | High target (Phase 2) |
| Total number of employees | 20 000 | 100 000 | 1 000 000 |
| Number of incentive requests per employee | 1 | 1 | 1 |
| Total number of requests | 20 000 | 100 000 | 1 000 000 |
| Total number of messages | 40 000 | 200 000 | 2 000 000 |

### Confidentiality and encryption

All external exchanges are encrypted using SSL / TLS, HTTPS.

All bus access are secured by a signed JWT Token.

The messages are siled by company, on dedicated queues.

The links provided are secure and give access to encrypted files (only decryptable by HR Managers).

### Traceability

All actions on the solution must be traced (sending or receiving a message, updating a configuration).

### Extensibility

It must be possible to quickly add new partners, by simple configuration.

### Security

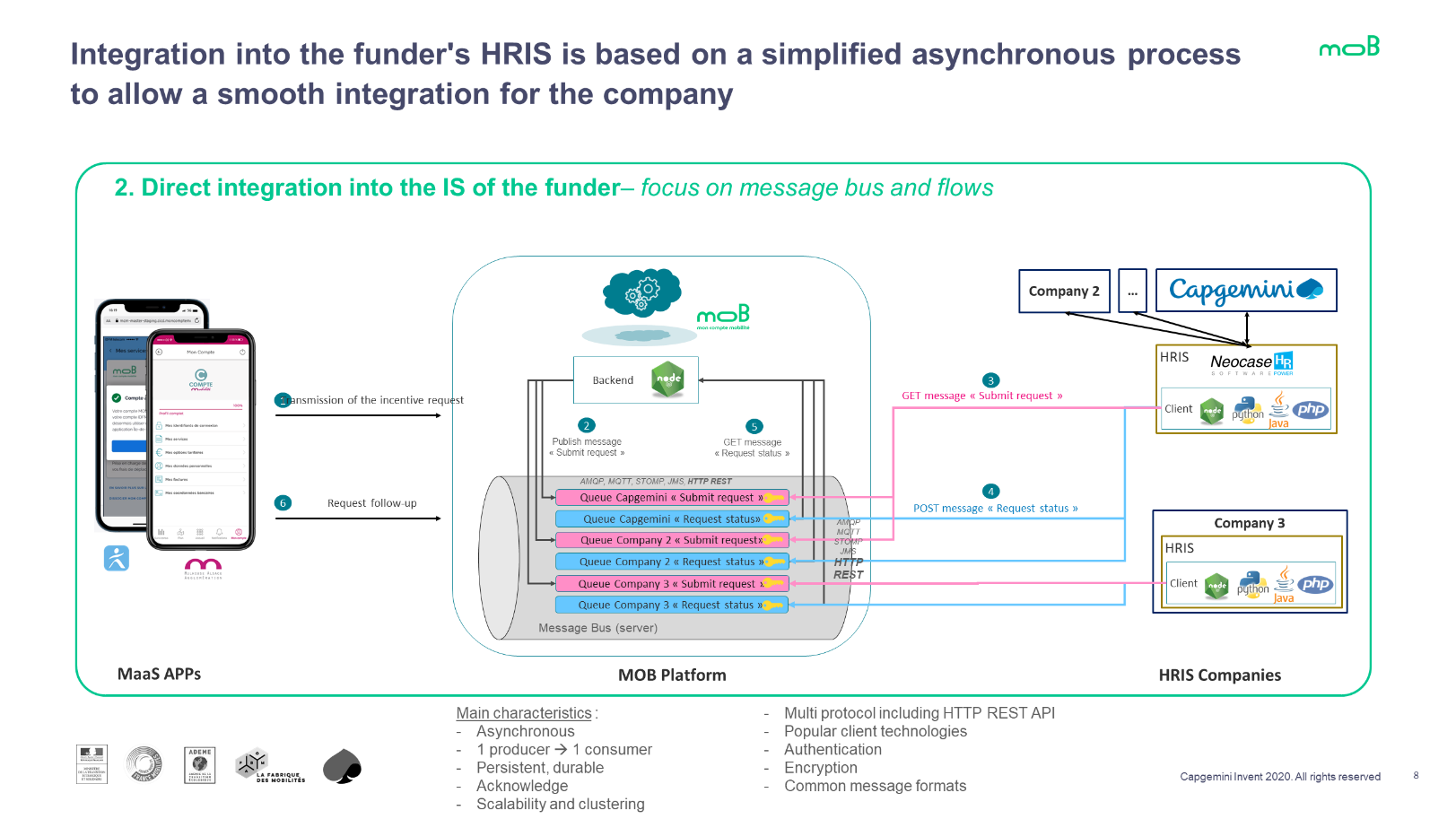
Avoid solutions requiring other systems to open an access to there is.

## Solution Architecture

Integration into the HRIS companies is based on a simplified asynchronous process to allow a smooth integration for the company.

The main characteristics of this solution are:

* Asynchronous
* 1 producer à 1 consumer
* Persistent, durable
* Acknowledgement
* Scalability and clustering
* Multi protocols including HTTP REST
* Popular client technologies
* Authentication
* Encryption



The advantages are:

* MOB system is notified directly, in both directions
* The company don’t need deploy a server to expose an API
* The company don’t need open his IS to the outside
* All companies can benefit from it, because they can connect with MOB, without having to deploy an event bus at home
* Only need one client (not a server), it is lightweight to implement
* Functional simplicity, for MOB and the companies. It is possible to make generic if companies agree.

In contrast, the companies must adapt to the generic MOB message formats.

### Message bus software

The message bus technology used for MOB is the solution [RabbitMQ](https://www.rabbitmq.com/).

It is the most widely deployed open-source message broker.

RabbitMQ is lightweight and easy to deploy on premises and in the cloud. It supports multiple messaging protocols. RabbitMQ can be deployed in distributed and federated configurations to meet high-scale, high-availability requirements.

RabbitMQ runs on many operating systems and cloud environments, and provides a wide range of developer tools for most popular languages.

Its purpose is to transmit a message received from any source to another source as soon as it is their turn. In other words, all transactions can be listed in a queue until the source to be transmitted gets up. RabbitMQ’s support for multiple operating systems and open-source code is one of the most preferred reasons.

#### Concepts

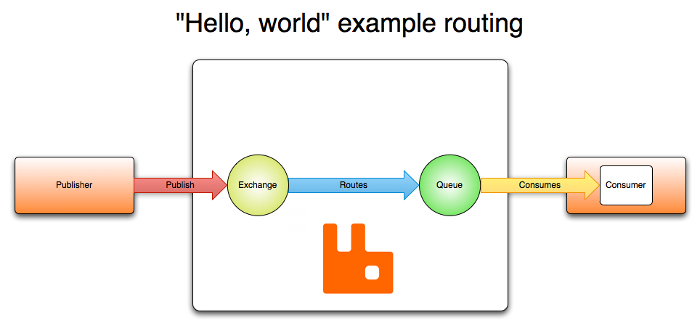


Figure 1: "Hello world" example routing in RabbitMQ

**Producer**: The source of the message is the application. It creates and sends the message.

**Queue**: Where messages are stored. The sent messages are put in a queue before they are received. All incoming messages are stored in Queue, that is memory.

**Consumer**: It is the server that meets the sent message. It is the application that will receive and process the message on the queue.

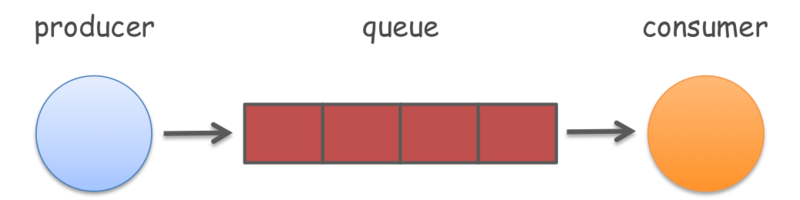
**Message**: The data we send on the queue. It is composed of 2 parts, headers (key-value pair) and body (the message).

**Exchange**: It is the structure that decides which queues to send the messages. It makes the decision according to routing keys.

**Binding**: The link between exchange and queue.

**FIFO**: The order of processing of outgoing messages in RabbitMQ is first in first out.

#### Basic cinematic



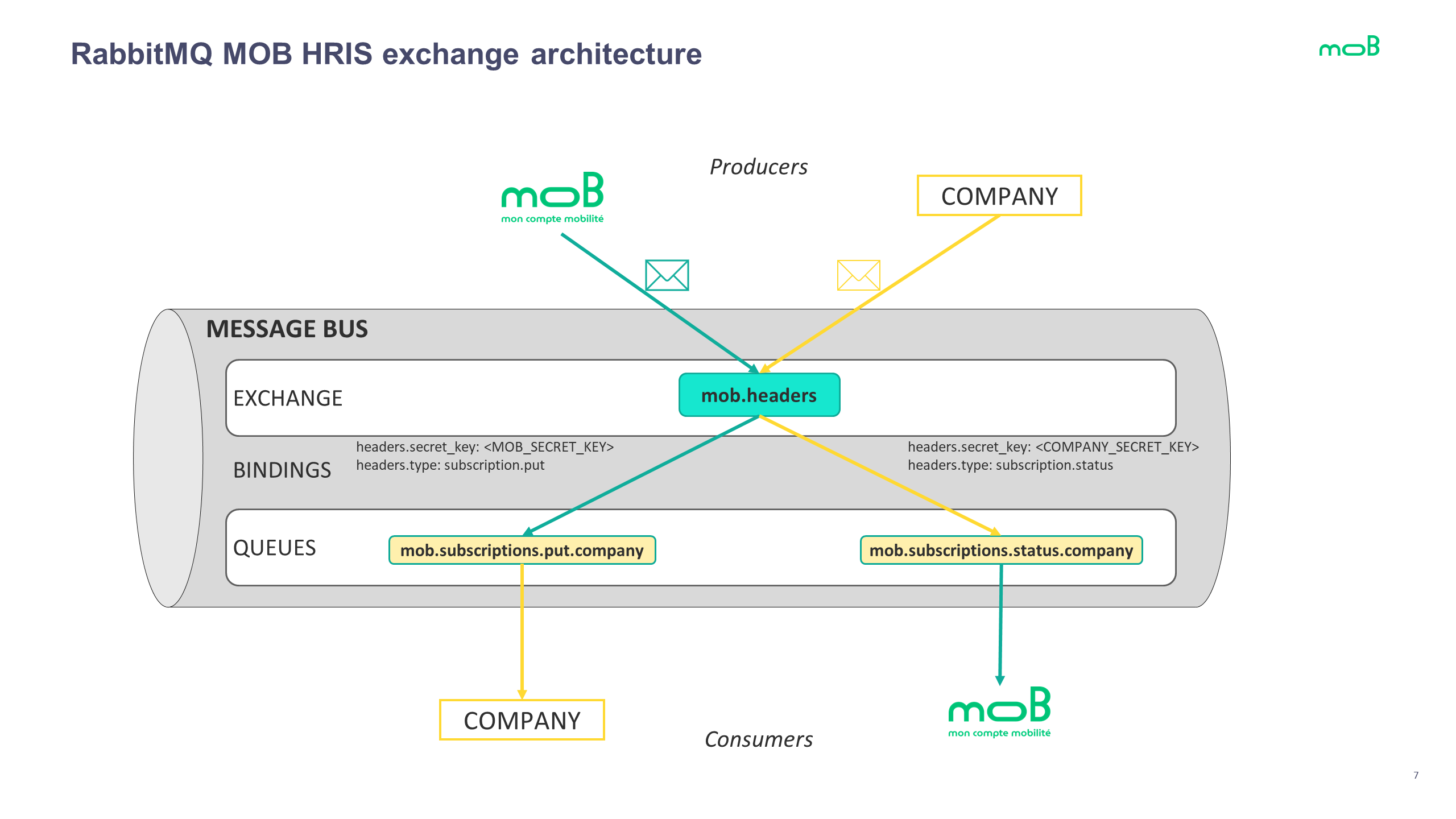
The Producer project sends a message to be queued. The message is received by the Exchange interface and redirects to one or more queues according to various rules.

### Queueing architecture and exchange type

The producer must address an exchange to write a message on the bus.

The exchange type selected is an exchange based on the headers. Thus, the routing is not based on the routing key but on the headers. 2 headers are sufficient for our use case.

The exchange routes the message in the right queue according to these properties. If one of these properties is not the value expected, the message is not routed and is lost.



On other side, the consumer address directly the queue which it wants to read.

### Supported and preferred protocols

RabbitMQ is multi-protocol, it supports the following protocols, but they are not all suitable for our use cases.

* **AMQP**, which is the core protocol supported by the RabbitMQ. For clients it's a reasonably easy protocol to implement, and as such there are [a large number of client libraries](https://www.rabbitmq.com/devtools.html) available for many different programming languages and environments.
* **JMS 1.1**, which is the is an application programming interface (API) by Sun Microsystems that functions as Java message-oriented middleware. It is designed for an exchange of loosely coupled, reliable and asynchronous messages between different software application components (called clients), which are based on the Java 2 Platform, Enterprise Edition (J2EE).
* MQTT (over HTTP using WebSockets or not), which is a binary protocol emphasising lightweight publish / subscribe messaging, targetted towards clients in constrained devices. It has well defined messaging semantics for publish / subscribe, but not for other messaging idioms.
* STOMP (over HTTP using WebSockets or not), which is a text-based messaging protocol emphasising (protocol) simplicity. It defines little in the way of messaging semantics, but is easy to implement and very easy to implement partially (it's the only protocol that can be used by hand over telnet).
* **HTTP REST API**, which is a simple API to send and receive messages. This is primarily intended for diagnostic purposes but can be used for low volume messaging.

|  |  |
| --- | --- |
| Protocol | Recommended for this use case |
| AMQP 0.9.1 | Yes |
| JMS 1.1 | Yes |
| MQTT | No |
| STOMP | No |
| HTTP REST API | Yes, only if JMS/AMQP are not possible. |

### Other solutions studied

Other solutions were studied but did not meet the functional and technical requirements

|  |  |  |
| --- | --- | --- |
| 1. Automatic email sending | Need to carry out developments & configuration for each new company  Solutions not technically state of the art  Complicated or impossible management of return flows | |
| 1. Sending flat files |
| 1. API implementations | *On Company side* :  Security constraints for IS companies that must open outwards  Complicated management of return flows | *On MOB side :*  Difficulty of scalability with many companies calling the API  Complicated management of return flows |

## Security

### Identity management, identification and authentication

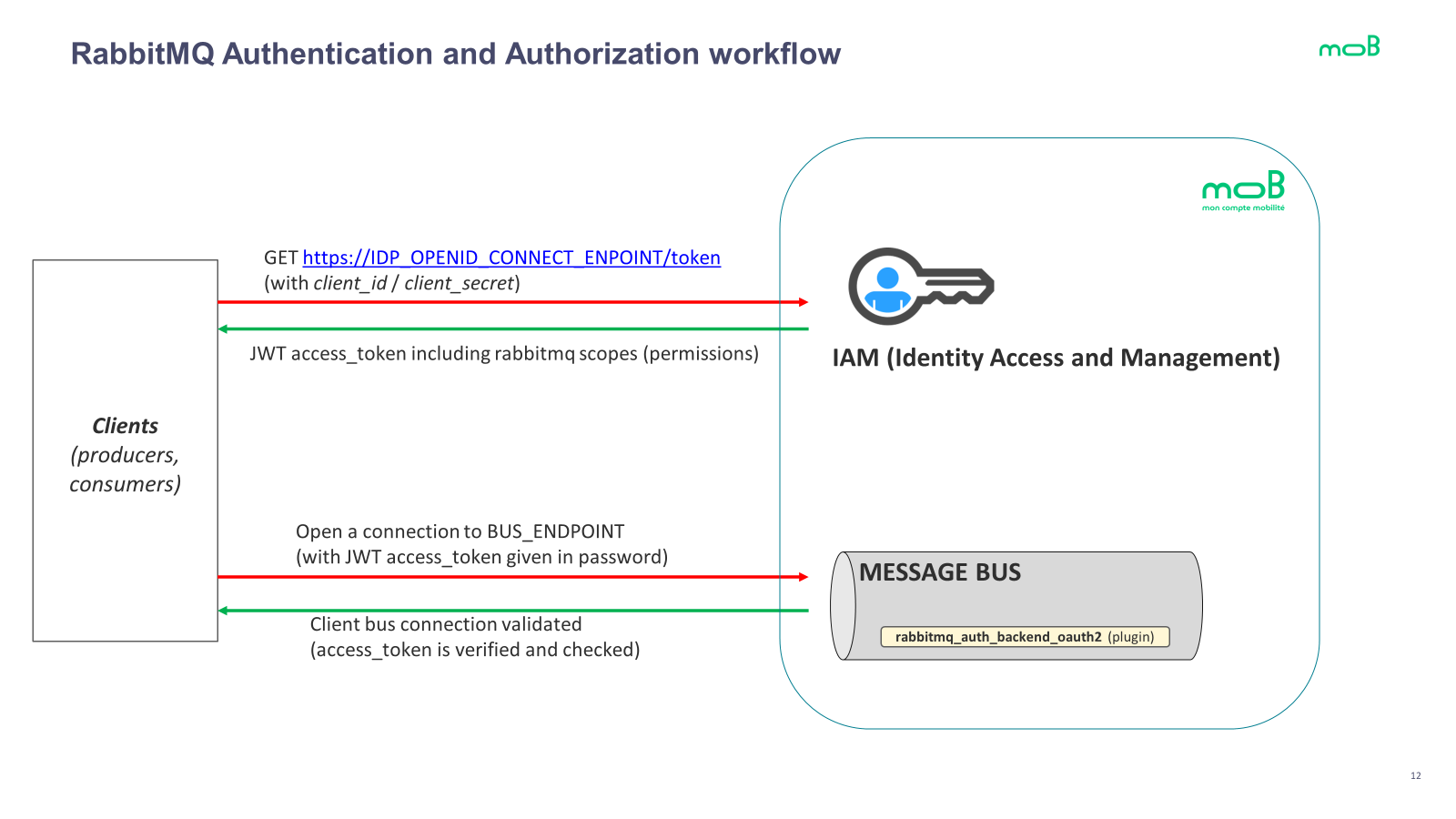
Message bus solutions and their protocols natively only support password authentication. The client connecting to the bus must provide a username and a password.

However, MOB is based on an identity access and management solution, which is compliant with OAuth 2 and OpenID Connect protocols.

We use the plugin [rabbitmq\_auth\_backend\_oauth2](https://github.com/rabbitmq/rabbitmq-server/tree/master/deps/rabbitmq_auth_backend_oauth2) to integrate bus clients with the IAM, instead of using the internal user management system.

One client per HRIS must be registered in IAM solution, as service accounts. A pair of secrets is generated for each and transmitted securely to HRIS.

The password authentication is disabled but it could be activated during a testing phase.



A communication between the bus client and the bus server must be initiated according the following 2 steps :

1. The client get an access token from the IDP (Identity Provider) server. The token obtained is composed of these fields in particular :
   1. « aud », which equals to the resource\_server\_id configured in the plugin on server side
   2. « scopes », which defines the permissions given to the client
2. The client open a connection with bus server by giving this access token in the password field parameter. The username field is « ignore » in this case.

#### Getting an access token

You must make a HTTP POST request on IDP token endpoint :

[https://idp-<version>.<env>.moncomptemobilite.fr/auth/realms/mcm/protocol/openid-connect/token](https://idp-rc01.testing.moncomptemobilite.fr/auth/realms/mcm/protocol/openid-connect/token)

Your request must contains the following key/values in the body (x-www-form-urlencoded):

|  |  |
| --- | --- |
| client\_id | *<HRIS\_client\_id>* |
| client\_secret | *<HRIS\_client\_secret>* |
| scope | openid |
| grant\_type | client\_credentials |

The response is given in JSON format by the IDP contains the main following fields:

|  |  |
| --- | --- |
| access\_token | JWT with “aud” and “scope” fields |
| expires\_in | 300 |
| refresh\_expires\_in | 1800 |
| refresh\_token | JWT |
| token\_type | bearer |
| id\_token | JWT |

### Authorization and access control

The permissions are based on object names in RabbitMQ.

The write operation is checked at the exchange level whereas the read operation is checked at queue level. We secure the message routing with headers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Tags | Configure regexp | Write regexp | Read regexp |
| user | administrator | *.\**  (all) | .\*  (all) | .\*  (all) |
| mob | management | ^$  (none) | ^mob.\*  (mob exchange) | ^mob\.subscriptions\.status\.\*  (only all queues « status ») |
| company | management | *^$*  (none) | *^mob.\**  (mob exchange) | *^mob\.*subscriptions*\.put\.company\.\**  (only company queue « put ») |

The authentication workflow defined in the previous chapter describe a token including permission definitions, contained in “scope” field.

The mapping between internal RabbitMQ permissions and the scopes is declared in the IDP, as clients and client scopes.

All permissions must be prefixed with the resource server id, set in the audience mapper (“aud” field in the JWT) and must be correspond to the value in server configuration.

The permissions are stored in int as client scopes. For example, a HRIS client must have the following scopes:

Tableau 1 : OAuth2 plugin permissions for a company HRIS

|  |  |
| --- | --- |
| Scope | Type |
| rabbitmq.tag:management | Tag |
| rabbitmq.configure:\*/%5E%24 | Permission |
| rabbitmq.write:\*/mob.\* | Permission |
| rabbitmq.read:\*/mob.subscriptions.status.company | Permission |

The client just needs to get the access token with the scope list and transmit the token as a password parameter during the connect operation.

Please note the access token expires after 5 minutes.

For information, the MOB client must have the following permissions:

|  |  |
| --- | --- |
| Scope | Type |
| rabbitmq.tag:management | Tag |
| rabbitmq.configure:\*/%5E%24 | Permission |
| rabbitmq.write:\*/mob.\* | Permission |
| rabbitmq.read:\*/mob.subscriptions.status.\* | Permission |

### Integrity

The integrity and encryption of exchanges is ensured by protocols HTTPS and SSL/TLS.

The server certificate subject presented is *<env>.moncomptemobilite.fr*.

## Message formats

The message must be transmitted in an escaped JSON string, with UTF-8 encoding.

A message is composed of the following parts:

* Properties (including headers)
* Payload (message body)

### Incoming flow

#### Properties

|  |  |  |
| --- | --- | --- |
| content\_type | application/json | |
| content\_encoding | UTF-8 | |
| delivery\_mode | 2 (persistent) | |
| headers | secret\_key | <MOB\_SECRET\_KEY> |
| message\_type | subscriptions.put.<company> |

The *secret\_key* property is used to ensure that the message from MOB backend. It must be populated with an agreed secret value only known by MOB.

#### Payload

{

"lastName":"Dupont",

"firstName":"Jacques",

"birthDate":" 1990-12-12T00:00:00.000Z",

"citizenId":"xxxxx-xxxxx-xxxxxx",

"incentiveId":"xxxxxxxxxxx",

"subscriptionId":"xxxxxxxx",

"email":"jacques.dupont@capgemini.com",

"communityName":"Managers",

"status":"A TRAITER",

"attachments":[

"https://api-v1\_1.preprod.moncomptemobilite.fr/api/v1/maas/subscriptions/xxxxxxxx/attachments/justif\_domicile.pdf",

"https://api-v1\_1.preprod.moncomptemobilite.fr/api/v1/maas/subscriptions/xxxxxxxx/attachments/facture.pdf"

],

"specificFields":{

"Nombre de kilomètres":"60",

"Montant":"35"

},

"encryptdAESKey": "base64string",

"encryptedIV": "base64string"

}

### Outgoing flow

#### Properties

|  |  |  |
| --- | --- | --- |
| content\_type | application/json | |
| content\_encoding | UTF-8 | |
| delivery\_mode | 2 | |
| headers | secret\_key | <COMPANY\_SECRET\_KEY> |
| message\_type | subscriptions.status.<company> |

The *secret\_key* property is used to ensure that the message from a specific company and goes to the right queue.

It must be populated with the client\_secret key obtained from MOB Identity Provider.

#### Validation payload

{

"citizenId":"xxxxxx",

"subscriptionId":"xxxxxxx",

"status":"VALIDEE",

"mode": "aucun",

"comments":"demande complète et valide"

}

#### Rejection payload

{

"citizenId":"xxxxxx",

"subscriptionId":"xxxxxxx",

"status":"REJETEE",

"type":"Autre",

"other":"justificatif non validé, la date ne correspond pas à la demande"

}

#### Error payload

{

"citizenId":"xxxxxx",

"subscriptionId":"xxxxxxx",

"status":"ERREUR"

}

## Publish a message

Two actors need publish messages on the bus:

* MOB platform, write messages about incentive requests to submit to HRIS
* HRIS, write status messages about incentive requests processed

These clients must open a connection to the bus to do this operation after [getting an access token](#_Getting_an_access).

The same exchange is targeted for all flows: **mob.headers**

The technical details depend on the [protocol and client](#_Protocol_access) used by the actor to interact with the IDP and the bus.

### Using AMQ client

Using this mode, you have to do :

1. Make an initial HTTP POST request to [get an access token](#_Getting_an_access) from IDP
2. Open a long-lived connection then a communication channel with the bus

|  |  |
| --- | --- |
| host | *bus-<version>.<env>.moncomptemobilite.fr* |
| port | *5672* |
| username | ignore |
| password | <JWT access token> |

1. Publish a message (basic\_publish) by defining the fields below

|  |  |
| --- | --- |
| exchange | *mob.headers* |
| routing\_key | *‘’ (empty)* |
| properties | *<properties and headers>* |
| body | <payload> |

It is optimal for large throughput.

The authentication mode required is based on a token, filled in the password parameter during the connecting phase.

The publish function defines all message parts: properties including headers, and the payload.

The flow is asynchronous.

On HRIS side, it is preferred to wait a confirm from message from bus server to validate that it has wrote the message correctly on disk.

However, HRIS must not wait a positive acknowledge from MOB processing because MOB can process messages according to a different time.

If the volumetry is low, it is possible to open the connection periodically and not keep it active always.

### Using HTTP REST API

Using this mode, you have to do one HTTP POST request by message to publish.

The prerequisite is to get an access token with client credentials.

It is not optimal for high performance, but it can be sufficient for certain HRIS, according to the number of processing incentive requests a day.

The authentication mode required is based on a token, given in the HTTP headers.

The url contains the targeted exchange “**mob.headers**”, for example :

[https://admin-bus-<version>.<env>.moncomptemobilite.fr/api/exchanges/%2F/**mob.headers**/publish](https://admin-bus-rc01.testing.moncomptemobilite.fr/api/exchanges/%2F/mob.headers/publish)

The HTTP request body contains all message parts: properties including headers, and the payload.

No acknowledge mechanism is possible, the flow is synchronous.

#### Example HTTP API publish request body

{

    "properties": {

        "content\_type": "application/json",

        "content\_encoding": "UTF-8",

        "delivery\_mode" : 2,

        "headers": {

            "secret\_key": "xxxxxxxxxxxxx",

"message\_type": "subscriptions.put.capgemini"

        }

    },

    "routing\_key": "",

    "payload": " {\"lastName\":\"lognaise\",\"firstName\":\"thibault\",\"birthdate\":\"1970-01-01T00:00:00.000Z\",\"citizenId\":\"ef3e1796-833c-45f9-92a4-30244dd17da0\",\"incentiveId\":\"62b5bfc119f0ddfeb707593d\",\"subscriptionId\":\"62b5c01919f0ddc41c07593e\",\"email\":\"test@exemple.com\",\"status\":\"A\_TRAITER\",\"communityName\":\"\",\"specificFields\":\"{\\\"Spec\_test\\\":\\\"free text\\\"}\",\"attachments\":[]}

",

    "payload\_encoding": "string"

}

#### Example HTTP API publish response body OK

If the message is published successfully, the response will look like as below.

{"routed": true}

#### Example HTTP API publish response body KO

If the message is no published and no destination queue matches, the response will look like as below.

{"routed": false}

#### Example HTTP API rejection request body

{

    "properties": {

        "content\_type": "application/json",

        "content\_encoding": "UTF-8",

        "delivery\_mode" : 2,

        "headers": {

            "secret\_key": "xxxxxxxxx",

"message\_type": "subscriptions.status.capgemini"

        }

    },

    "routing\_key": "",

    "payload":  " {\r\n\t\"citizenId\":\"31542651-24de-4d7b-9e56-487a54df65dc\",\r\n\t\"subscriptionId\":\"624d5d9498792515f0324bab\",\r\n\t\"status\":\"REJETEE\",\r\n\t\"type\":\"Autre \",\r\n\t\"other\":\"reason why\"\r\n}

",

    "payload\_encoding": "string"

}

## Receive a message

Two actors need receive messages on the bus:

* MOB platform, get messages about incentive requests status from HRIS
* HRIS, get incentive requests messages about incentive requests submitted by employees from MOB

These clients must open a connection to the bus to do this operation after [getting an access token](#_Getting_an_access).

The operation targets directly the queue desired:

* **mob.subscriptions.status.<company>**
* **mob.subscriptions.put.<company>**

The technical details depend on the [protocol and client](#_Protocol_access) used by the actor to interact with the IDP and the bus.

### Using AMQP client

Using this mode, you have to do:

1. Make an initial HTTP POST request to [get an access token](#_Getting_an_access) from IDP
2. Open a long-lived connection then a communication channel with the bus

|  |  |
| --- | --- |
| host | *bus-<version>.<env>.moncomptemobilite.fr* |
| port | *5672* |
| username | ignore |
| password | <JWT access token> |

1. Define a callback function which will be triggered on each message received in order to get all message parts below and sending an acknowledgement (basic\_ack):

|  |  |
| --- | --- |
| method | *<method>* |
| properties | *<properties and headers>* |
| body | <payload> |

1. Start a consumer (basic\_consume) on the target queue by defining the target queue and the callback function

|  |  |
| --- | --- |
| queue | *mob.subscriptions.<type>.<company>* |
| on\_message\_callback | *<callback function>* |

It is optimal for large throughput.

The authentication mode required is based on a token, filled in the password parameter during the connecting phase.

The publish function defines all message parts: properties including headers, and the payload.

The flow is asynchronous.

It is preferred to wait a confirm from message from bus server to validate that it has wrote the message correctly.

HRIS must not wait an acknowledge from MOB processing because MOB can process messages according to a different time.

If the volumetry is low, it is possible to open the connection periodically and not keep it active always.

### Using HTTP REST API

Using this mode, you have to do one HTTP POST request by message or group of messages.

The prerequisite is to get an access token with client credentials.

It is not optimal for high performance, but it can be sufficient for certain HRIS, according to the number of processing incentive requests a day.

The authentication mode required is based on a token, given in the HTTP headers.

The url contains the targeted queue, for example:

[https://admin-bus-<version>.<env>.moncomptemobilite.fr/api/queues/%2F/mob.subscriptions.status.<company>/get](https://admin-bus-arnaud.preview.moncomptemobilite.fr/api/queues/%2F/mob.subscriptions.status.capgemini/get)

The HTTP request body contains few fields to customize the receiving:

|  |  |
| --- | --- |
| count | *maximum number of messages to get* |
| ackmode | *ack\_requeue\_false* |
| encoding | auto |

No real acknowledge mechanism is possible, the flow is synchronous.

The ackmode parameter indicates to the bus server to discard the message after the getting.

#### Example HTTP API get request body

{

    "count": 5,

    "ackmode": "ack\_requeue\_false",

    "encoding": "auto"

}

#### Example HTTP API get response body

[

    {

        "payload\_bytes": 440,

        "redelivered": **false**,

        "exchange": "mob.headers",

        "routing\_key": "",

        "message\_count": 0,

        "properties": {

            "delivery\_mode": 2,

            "headers": {

                "message\_type": "subscriptions.put.company",

                "secret\_key": "xxxxxxxxxxxx",

                "x-delivery-count": 0

            },

            "content\_encoding": "UTF-8",

            "content\_type": "application/json"

        },

        "payload": " {\"lastName\":\"lognaise\",\"firstName\":\"thibault\",\"birthdate\":\"1970-01-01T00:00:00.000Z\",\"citizenId\":\"ef3e1796-833c-45f9-92a4-30244dd17da0\",\"incentiveId\":\"62b5bfc119f0ddfeb707593d\",\"subscriptionId\":\"62b5c01919f0ddc41c07593e\",\"email\":\"test@exemple.com\",\"status\":\"A\_TRAITER\",\"communityName\":\"\",\"specificFields\":\"{\\\"Spec\_test\\\":\\\"free text\\\"}\",\"attachments\":[]}

",

        "payload\_encoding": "string"

    }

]

## Download file attachments

The incoming flow contains secure link(s) to download files that can be sensitive.

The links provided are like the following example:

[[https://api-<version>.<env>.moncomptemobilite.fr/v1/subscriptions/{subscriptionId}/attachments/{filename}](https://api-rc01.testing.moncomptemobilite.fr/v1/subscriptions/%7bidSubscription%7d/attachments/%7bfilename%7d)](https://api-rc01.testing.moncomptemobilite.fr/v1/demandes/2/justificatifs)

These links are secured by client credentials authentication. HRIS must use the [access token](#_Getting_an_access) obtained at starting.

These links are valid for a defined period.

**The filed are encrypted using a symmetric key that is encrypted with the funder public key. All the details about the attachment encryption/decryption are available in the** [**specific documentation**](#_top)**.**

Caution: the response obtained from this endpoint is an UTF-8 buffer in Unicode (add one zero for each character).

The files are purged on MOB after this delay and cannot be retrieved after.

Only the metadata of the incentive request are kept and available on MOB. The details of the request including attachments are only available in HRIS.

## Environments

The essential endpoints to know are the token IDP endpoint and the bus protocols url to open a connection.

### Préproduction

|  |  |  |
| --- | --- | --- |
| Address | Usage | Authentication mode |
| https://idp-v1\_1.preprod.moncomptemobilite.fr/auth/realms/mcm/protocol/openid-connect/token | Get an access\_token | Client secrets |
| amqp://ignore:$token@bus-v1\_1.preprod.moncomptemobilite.fr/%2F | Initiate a connection to the message bus in order to publish a message or to create a consumer to get messages | JWT |
| https://admin-bus-v1\_1.preprod.moncomptemobilite.fr/api/exchanges/%2F/mob.headers/publish | Publish a message from HTTP Management API (not ideal but possible) | JWT |
| https://admin-bus-v1\_1.preprod.moncomptemobilite.fr/api/queues/%2F/mob.subscriptions.put.<company>/get | Get messages “subscriptions.put” from HTTP Management API (not ideal but possible) | JWT |

### Production

|  |  |  |
| --- | --- | --- |
| Address | Usage | Authentication mode |
| https://idp.moncomptemobilite.fr/auth/realms/mcm/protocol/openid-connect/token | Get an access\_token | Client secrets |
| amqp://ignore:$token@bus.moncomptemobilite.fr/%2F | Initiate a connection to the message bus in order to publish a message or to create a consumer to get messages | JWT |
| https://admin-bus.moncomptemobilite.fr/api/exchanges/%2F/mob.headers/publish | Publish a message from HTTP Management API (not ideal but possible) | JWT |
| https://admin-bus.moncomptemobilite.fr/api/queues/%2F/mob.subscriptions.put.<company>/get | Get messages “subscriptions.put” from HTTP Management API (not ideal but possible) | JWT |