
بسم الله الرحمن الرحيم

Dedications

I dedicate this modest work, with all my love and gratitude, to you, for your tenderness, your unwavering support, and the countless sacrifices you have made. No words can fully express the depth of my gratitude.

I also thank from the bottom of my heart my friends and classmates for their kind presence, moral support and constant encouragement which helped me to complete this project.

Thanks

First and foremost, my thanks also go to Mrs. Meriyem Chergui, my academic supervisor, for her availability, her attentive monitoring and her wise advice which allowed me to structure and enhance this work with clarity and academic rigor.

I would like to express my deepest gratitude to Mr. Ismail Chakour, my professional supervisor at HPS, for his trust, his rigorous support, and the valuable responsibilities he entrusted to me throughout this adventure. His technical rigor and strategic vision have been a continuous source of inspiration.

I sincerely thank the members of Ismail's team, particularly Aissam Rhounimi, for their collaboration, availability, and team spirit. A special thought also goes to Kamal Laaroussi, for his valuable technical input, and to Meryem Fouzair, our HR representative, for her professionalism and seamless administrative support throughout my time at HPS.

To my mother, for her tenderness, her patience, and her countless silent sacrifices that allowed me to be the man I am.

To my father, for his strength, his example, and his colossal efforts since my early childhood so that I could surpass myself and strive for excellence. He raised me not to be ordinary, but to be the best.

I would also like to extend my heartfelt thanks to my uncle Khalid, a discreet but decisive pillar of my career. He stood by me from my earliest years until my integration at HPS. His unconditional support, generosity, and presence shaped my career as much as my skills.

I can't end these thanks without a special thought for my sister Asmae. With her zest for life, her spontaneous humor, and her ability to make every moment a burst of laughter, she brought precious lightness to the most intense moments. She transformed the house into a lively, luminous place, filled with love and laughter. Thank you for being this breath of fresh air every day.

To all my friends, near and far, who have provided me with support, motivation, or simply a word at the right time, thank you. Whether through a chat, a joke, a coffee break, or a simple knowing glance, your presence has played an essential role in keeping me grounded during this challenging time. You have each contributed, in your own way, to making this journey more human, more joyful, and infinitely more memorable.

Finally, to my family, to all those who believed in me even when I doubted: this internship, this report, and every line of code written are also yours.

This internship represents much more than just a line on a CV, it is the reflection of a life path, marked by sacrifices and dedication.

Summary :

This report describes my final year internship experience at HPS, during which I worked on a strategic project to migrate the Mastercard CIS Channel from a legacy monolithic C language architecture to a modern V4 microservices architecture, based on Java Spring Boot.

Throughout these months, I was able to develop solid technical skills through handling C legacy code, analyzing critical functions of the CIS protocol and gradually rebuilding these logics in Java, in a microservices-oriented environment, containerized with Docker and orchestrated via Kubernetes. I also participated in setting up the test environment, continuous integration, and developing an SDK dedicated to the CIS protocol, to centralize and reuse the mapping, TLV encoding, and ISO8583 field processing functions.

This project allowed me to deepen my understanding of the digitale banking sector, particularly through Issuing and Acquiring flows, as well as the different message types: Authorization, Reversal, Advice and File Upload. I thus acquired a clear vision of the communication mechanisms between an electronic payment switch and the Mastercard network, and the compliance and security constraints associated with them.

Beyond the technical aspects, this internship also allowed me to strengthen my soft skills:

I learned to organize my work independently, to manage priorities in a complex technical context, to collaborate with senior teams, and to demonstrate rigor and proactivity.

Regular interaction with my professional supervisor, team members, and technical advisors allowed me to evolve in a stimulating and professionally demanding environment.

In summary, this internship provided me with a comprehensive experience, at the intersection of advanced software engineering and digital finance. It allowed me to develop technical, functional, and human skills, while contributing to a real, structuring, and high-impact project at HPS.

Résume :

Ce rapport décrit mon expérience de stage de fin d'études chez HPS, au cours duquel j'ai travaillé sur un projet stratégique visant à migrer le Channel Mastercard CIS d'une architecture monolithique héritée en langage C vers une architecture moderne en microservices V4, basée sur Java Spring Boot.

Tout au long de ces mois, j'ai pu développer des compétences techniques solides en manipulant du code legacy, en analysant des fonctions critiques du protocole CIS, et en reconstruisant progressivement ces logiques en Java, dans un environnement orienté microservices, conteneurisé avec Docker et orchestré via Kubernetes. J'ai également participé à la mise en place de l'environnement de test, de l'intégration continue, ainsi qu'au développement d'un SDK dédié au protocole CIS, afin de centraliser et de réutiliser les fonctions de mapping, de codage TLV et de traitement des champs ISO8583.

Ce projet m'a permis d'approfondir ma compréhension du secteur bancaire, notamment à travers les flux Issuing et Acquiring, ainsi que les différents types de messages : Authorization, Reversal, Advice et File Upload. J'ai ainsi acquis une vision claire des mécanismes de communication entre un switch de paiement électronique et le réseau Mastercard, ainsi que des contraintes de conformité et de sécurité qui y sont associées.

Au-delà des aspects purement techniques, ce stage m'a également permis de renforcer mes soft skills : j'ai appris à organiser mon travail de manière autonome, à gérer les priorités dans un contexte technique complexe, à collaborer avec des équipes seniors et à faire preuve de rigueur et de proactivité. Les échanges réguliers avec mon encadrant professionnel, les membres de l'équipe et les experts techniques m'ont permis d'évoluer dans un environnement stimulant et exigeant sur le plan professionnel.

En résumé, ce stage m'a offert une expérience complète, à l'intersection de l'ingénierie logicielle avancée et de la finance numérique. Il m'a permis de développer des compétences techniques, fonctionnelles et humaines, tout en contribuant à un projet réel, structurant et à fort impact au sein de HPS.

ملخص

يصف هذا التقرير تجربتي في التدريب النهائي في شركة HPS، حيث عملت على مشروع استراتيجي لنقل قناة CIS Mastercard من بنية أحادية قديمة بلغة C إلى بنية حديثة تعتمد على خدمات V4 المصغرة، باستخدام Java Spring Boot.
على مدار هذه الأشهر، تمكنت من تطوير مهارات تقنية قوية من خلال التعامل مع كود C القديم، وتحليل الوظائف الحرجة لبروتوكول CIS وإعادة بناء هذه المنطق تدريجياً في Java، في بيئة موجهة نحو الخدمات المصغرة، محمولة باستخدام Docker ومنسقة عبر Kubernetes. كما شاركت في إعداد بيئة الاختبار، والتكامل المستمر، وتطوير مجموعة أدوات البرمجيات (SDK) المخصصة لبروتوكول CIS، بهدف مركزية وإعادة استخدام وظائف التعيين، وترميز TLV، ومعالجة حقول ISO8583.
سمح لي هذا المشروع بتعميق فهمي لقطاع البنوك، لا سيما من خلال تدفقات الإصدار والاستحواذ، فضلاً عن أنواع الرسائل المختلفة: التفويض، الإلغاء، الإشعار وتحميل الملفات. وبذلك اكتسبت رؤية واضحة لآليات الاتصال بين مفتاح الدفع الإلكتروني وشبكة Mastercard، والقيود المتعلقة بالامتثال والأمان المرتبطة بها.
بصرف النظر عن الجوانب التقنية، فقد أتاح لي هذا التدريب أيضًا تعزيز مهاراتي الشخصية: تعلمت كيفية تنظيم عملي بشكل مستقل، وإدارة الأولويات في سياق تقني معقد، والتعاون مع الفرق العليا، وإظهار الدقة والاستباقية.
التفاعل المنتظم مع مشرفي المهني، وأعضاء الفريق، والمستشارين الفنيين سمح لي بالتطور في بيئة تحفيزية ومهنية تتطلب الكثير.
باختصار، زودني هذا التدريب بتجربة شاملة، عند تقاطع هندسة البرمجيات المتقدمة والتمويل الرقمي. سمح لي ذلك بتطوير المهارات التقنية والوظيفية والإنسانية، بينما كنت أساهم في مشروع حقيقي وبنيوي وعالي التأثير في HPS.

Acronyms

Acronyms	Meaning
HPS	High-tech Payment System
HSM	Hardware (or Host) Security Module
PWC	PowerCard
CIS	Customer Interface Specifications
NW	Network
Acq	Acquiring
ISS	Issuing
ONL	Online
SDK	Software Development Kit
ZKP	PIN Key Zone
AWK	Acquire Working Key
TMK	Terminal Master Key
REP	Response
REQ	Request
Auth	Authentication
EMV	Europay, Mastercard, Visa
TTL	Time To Live

Contents

List of Figures

General introduction

The electronic payments sector is undergoing a profound transformation, driven by evolving technologies, security requirements, and growing expectations for scalability, traceability, and interoperability. In this context, financial institutions are turning to more modern and resilient architectures to meet new international standards. This project, carried out within HPS, is part of this modernization process.
The project aims to completely migrate the Mastercard CIS Channel, a strategic component of the PowerCARD payment system, from a legacy architecture in C language (V4) to a new modular version in Java Spring Boot (V4). The CIS Channel plays a fundamental role in the processing of payment messages, ensuring the mapping and bidirectional conversion between the internal ISO8583 PowerCARD format and the external Mastercard CIS protocol.
The old system, while robust, now has limitations in terms of maintainability, scalability, and integration with modern tools such as Kafka, Docker, or Kubernetes. The new architecture therefore adopts a microservices approach, splitting responsibilities between two main components:
Protocol: responsible for business processing and message mapping.
Connector: responsible for network communication via TCP/IP with the Mastercard system.
This report focuses on presenting this migration in detail, addressing the technical choices made, the advantages of the new architecture, as well as the different stages of design, implementation and testing. We will also analyze the transaction flows (Authorization, Reversal, Advice, File Upload), the processing logic on the Issuing and Acquiring sides, as well as the benefits brought by the adoption of modern technologies (Spring Boot, Kafka, OpenTelemetry, etc.).
The overall objective of this project is to strengthen the robustness and flexibility of the electronic payment system, while ensuring compliance with Mastercard standards and ensuring seamless interoperability with the PowerCARD V4 platform. This strategic migration represents a key step towards the modernization of electronic payment systems at HPS, paving the way for the seamless integration of future protocols (Base1, MDS, etc.).

Chapter 1

Internship Environment

Introduction

In this section, I will introduce the host company by detailing its organizational chart, its activities, its fundamental values, with particular emphasis on its subsidiary in Morocco.

1. Presentation of the host organization:

HPS is an internationally renowned Moroccan company specializing in electronic payment solutions. Founded in 1995, HPS has established itself as a leader in the field of payment systems, offering innovative and secure solutions tailored to the needs of financial institutions and large corporations around the world.

Figure 1.1: HPS Morocco identity card

Since its inception, HPS has continued to grow and diversify. The company began by providing credit and debit card processing solutions and gradually expanded its portfolio

to include mobile payment, online payment, and risk management solutions. HPS has also expanded its international presence, with offices and clients in over 85 countries.

2. HPS History:

   HPS was born from its founders' vision to create innovative and secure electronic payment solutions to meet the needs of financial institutions. From the very beginning, HPS focused on developing cutting-edge technologies and quickly gained recognition in the national market.

   In 1999, HPS marked a milestone with the launch of PowerCard, an integrated payment platform that transformed the management of bank cards and payment transactions. This flagship product enabled HPS to expand its operations internationally, attracting customers in Africa, the Middle East, Europe, and Asia.

   Over the years, HPS has continued to innovate and diversify its offerings, consolidating its leadership position in the payment technology sector. In 2010, the acquisition of ICPS (Indian Ocean Card Processing Services) strengthened its presence in the international market and expanded its client portfolio.

   Today, HPS is recognized as a major player in the field of electronic payment solutions, with a presence in more than 90 countries and a diverse customer base including banks, financial institutions, and payment operators.

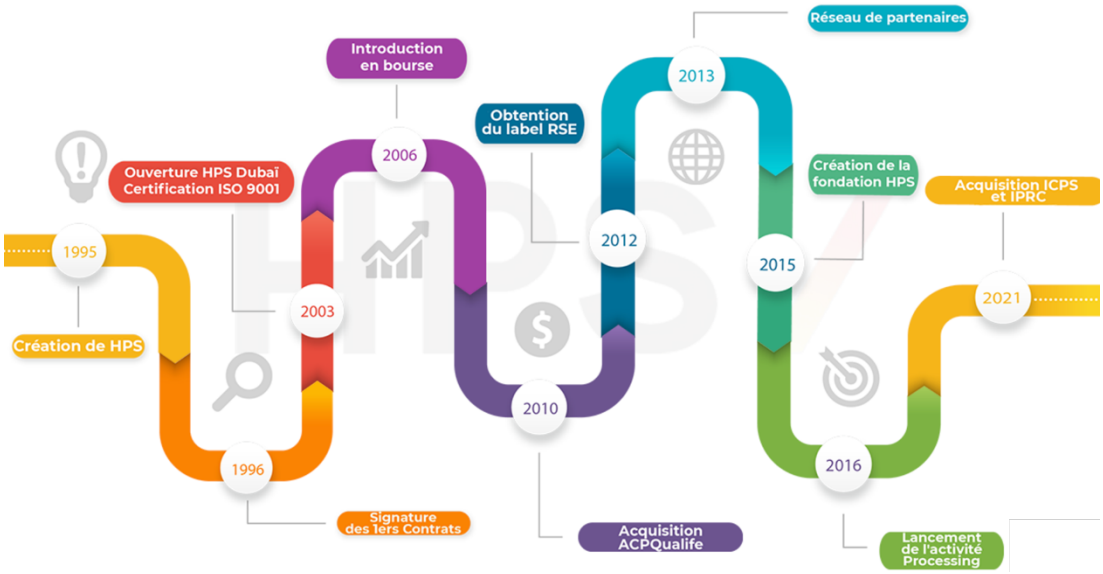


Figure 1.2: HPS History

3. Main missions:

   HPS is the invisible technology that makes simple, transparent, and secure payments possible, allowing people to create, share, and live.

   HPS enables the delivery of high-value solutions and services and ensures the smooth and secure execution of transactions across all possible payment channels and across all business sectors.

   It is constantly working to enhance its suite of POWERCARD solutions for omnichannel electronic payment, which it deploys to its customers around the world.

4. Main activities:

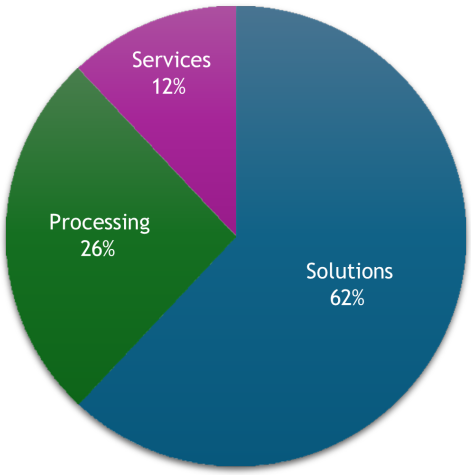


Figure 1.3: HPS activities

Solutions: Omnichannel management of electronic payments in "on-premise" mode through PowerCard Solutions.

Processing: Management of national switching activities in Morocco and processing of transactions through the POWERCARD platform in SaaS mode.

Services: Testing, software qualification and project management assistance for IT transformation projects.

5. Main products:

PowerCard: It is a payment card and mobile payment management platform. PowerCard enables financial institutions to efficiently manage financial transactions, issue payment cards, and provide mobile payment services.

SmartSwitch: SmartSwitch is a payment switching solution that provides secure connectivity between financial sector stakeholders. It facilitates the routing of transactions between issuers, acquirers, and payment networks.

PowerCard Connect: This Payments Platform as a Service (PaaS) enables businesses to quickly launch customized payment services, reducing time to market.

ePower: ePower is a mobile payment solution developed by HPS to enable secure, contactless payments via smartphones.

Switchware: Switchware is a comprehensive product suite dedicated to payment management for banks and financial institutions.

6. International presence:

With offices in Africa, Europe, the Middle East, Asia, and North America, HPS is a truly global company. This international presence allows HPS to understand local specificities and offer solutions tailored to each market.

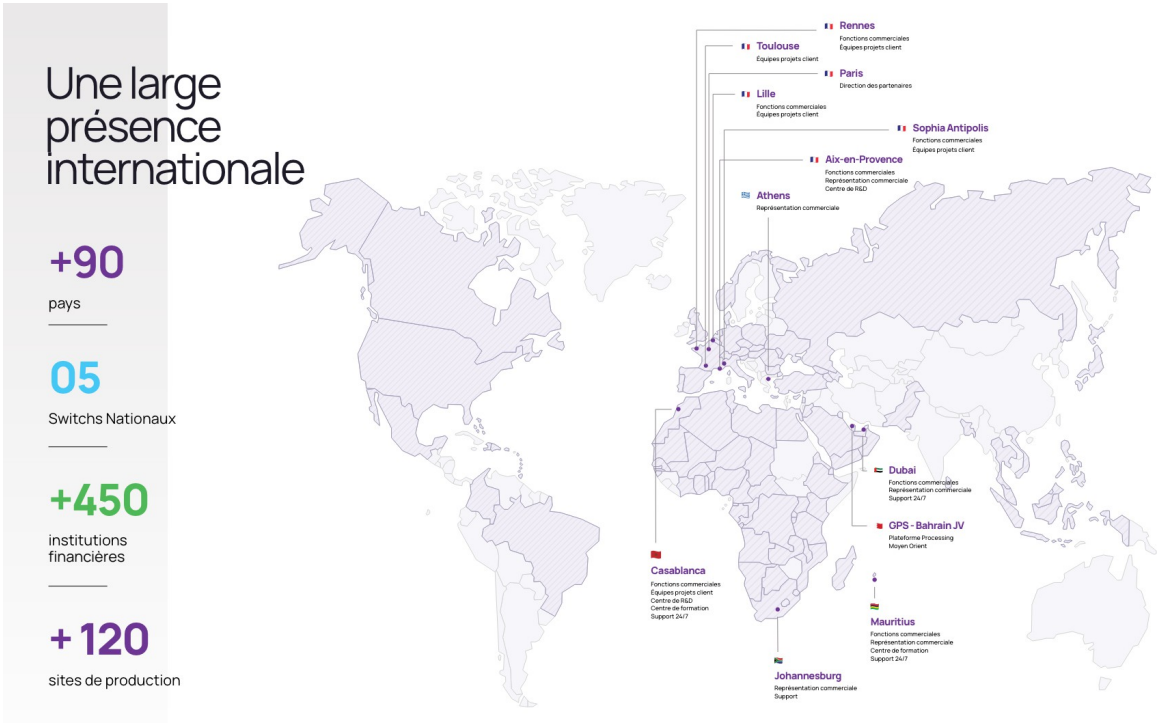


Figure 1.4: HPS International Presence

7. Company organization chart:

   In 2021, the HPS Group adopted a new organization that takes into account its new dimension and its international development. This organization gives the various entities of the Group increasing autonomy and guarantees the achievement of several levels of synergies between them. It also highlights two essential aspects for the future evolution of the HPS Group: customer support and sales development through the Market entity, and management of the integration and development of PowerCard solutions in its different versions and according to the two models under the responsibility of the Software Factory entity.

   Furthermore, the Group's new organization reflects the consolidation of the two activities previously grouped within Processing (Payment & Switching), following their sustained development over the past few years, both organically and through acquisitions. HPS has thus set up the Payments Services entity, responsible for processing payment transactions and services related to its operations, and has given the Switching activity a dedicated entity to support the development of electronic payment at the national level. The Group has also strengthened the prerogatives of the Corporate Services entity, which has a central and transversal role, enriched by the new functions linked to external growth and the management of the Group's CSR issues. The operational deployment of the Group's strategic orientations and the monitoring of its performance are ensured by the Executive Committee under the supervision of the General Management, which also relies on the Transformation Committee, which oversees the implementation and supervision of the Group's transformation strategy.

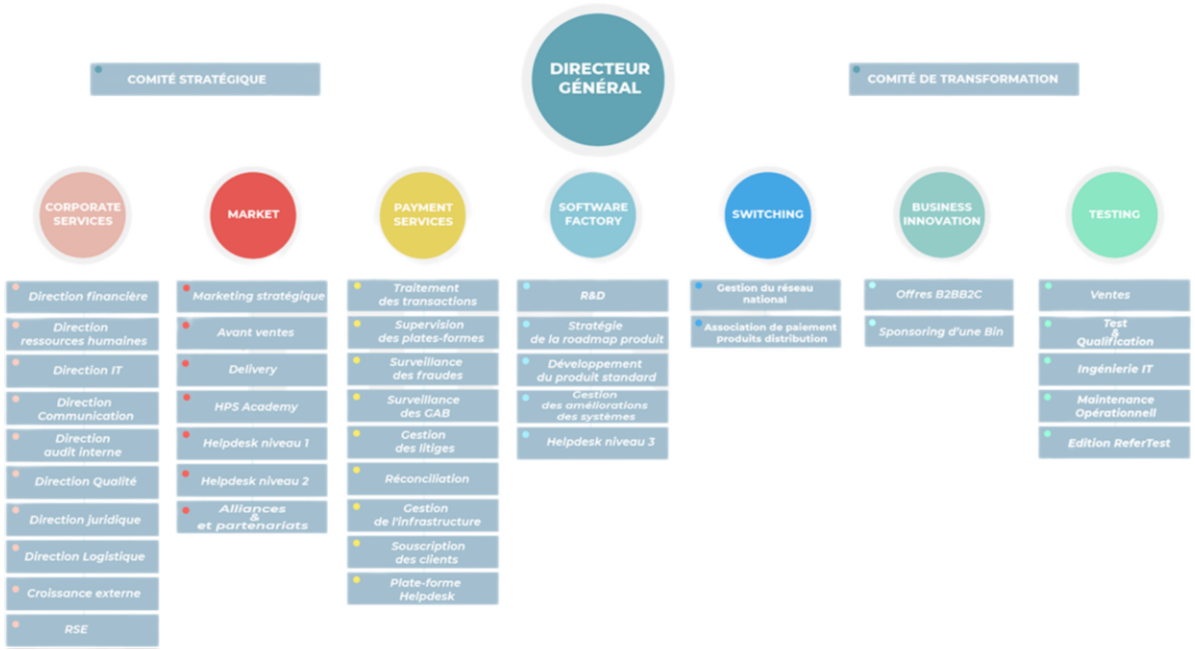


Figure 1.5: HPS Organization Chart

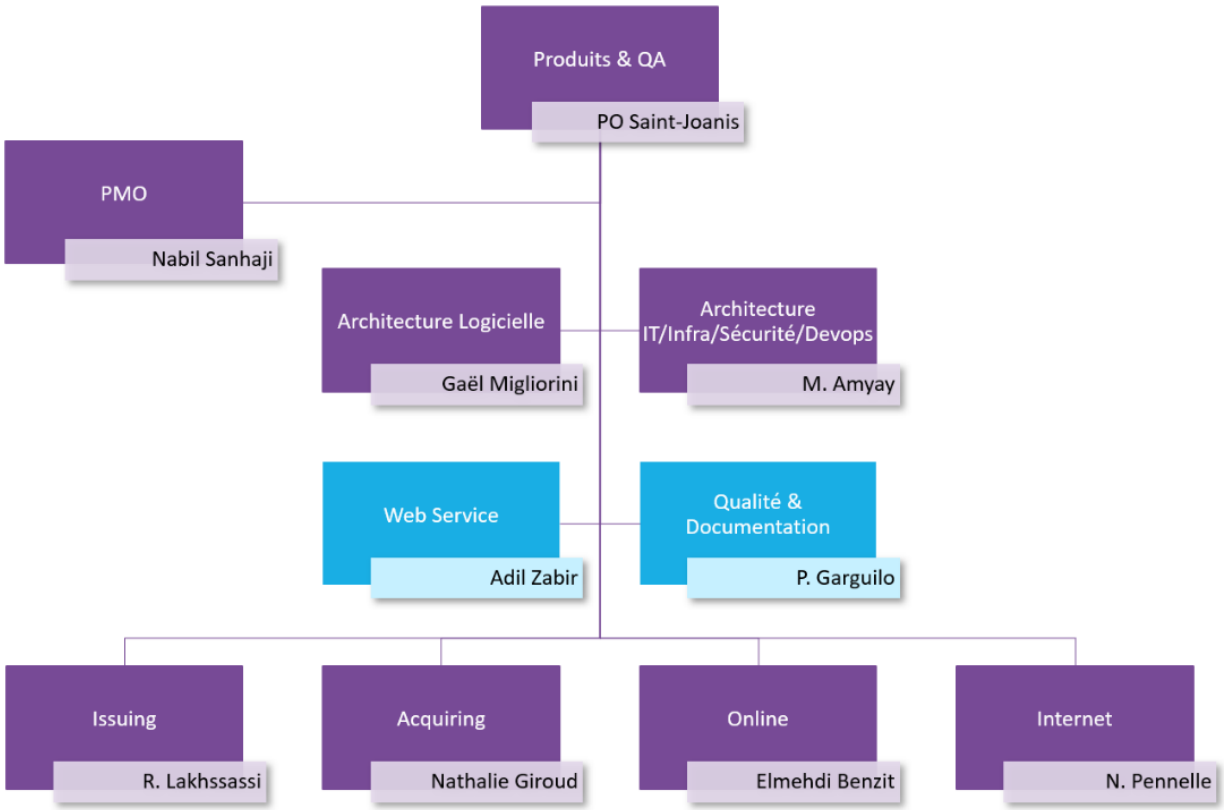


Figure 1.6: HPS Organization Chart 2

8. Presentation of the training:

   From our first day at HPS, the company offers us training on the principles and values of HPS, as well as self-training via the HPS-EACADEMY platform.

Regarding these training courses:

- Introduction to the Payments Industry
- Integration into the HPS Group
- PowerCard Solutions
- PCI DSS
- Norms and standards

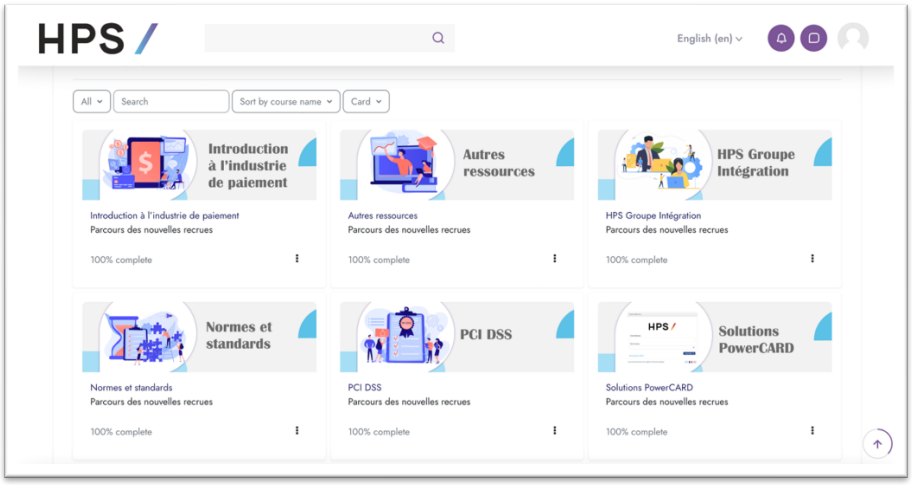


Figure 1.7: HPS Training

1.0.1 Conclusion:

This chapter provided a comprehensive overview of the internship environment, introducing the host organization, its history, missions, core activities, and international presence. These elements helped to understand the framework in which the internship took place and the key aspects of the HPS company.

Chapter 2: General context of the project

1.1.1 Introduction

The specification is the first step in a project. This step is crucial for the smooth running of the project. It consists of knowing and understanding the work required from an organizational and structural point of view. In the first part, we will begin by presenting the general context of the application by discussing the problem, objectives, and scope. In the second part, we will present the work methodology and planning.

1.1.2 Fundamental concepts:

Cis (Customer Specification Interface):

This is Mastercard's implementation of the ISO 8583-1987 standard for processing authorization information using the Mastercard Dual Message System authorization platform. It provides Mastercard customers with the information needed to develop a software interface between Customer Processing Systems (CPS) and Mastercard's Dual Message System.

What is CIS Interface?

It is a Powercard interface designed to communicate with the Mastercard network for online authorization management.

Where can the CIS interface be configured?

We can configure the CIS interface in PowerCard from the following screen: Settings > PowerCARD-Switch > Switch Settings > Resources > Definition > Resource Settings

What is the mapping between CIS protocol and Powercard internal protocol?

Here we find the mapping between the CIS protocol and the Powercard protocol.

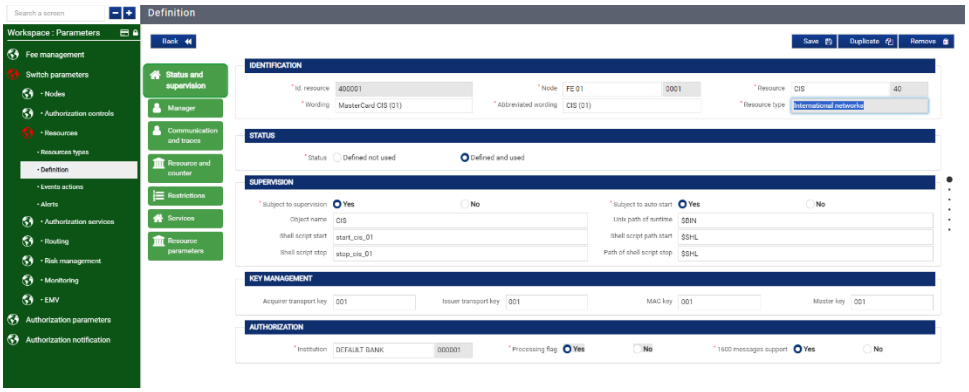


Figure 8 :CIS configuration interface

Issuer background

As part of issuing an authorization request, Powercard uses the CIS interface to capture and process incoming messages from Visa, performing the necessary security checks associated with the issuing party and the mapping between the CIS protocol and the Powercard Switch protocol...

Acquirer background

As part of acquiring an authorization request, Powercard uses the base1 interface to process incoming messages from Visa, performing the necessary security checks associated with the acquiring party and the mapping between the Powercard Switch protocol and the CIS protocol...

Issuing and acquiring:

Issuing: This is the bank that issued the customer's credit card. It is responsible for authorizing or declining transactions.

Acquiring : This is the merchant's bank, which accepts card payments and relays authorization requests to the banking network.

Simplified flow diagram :

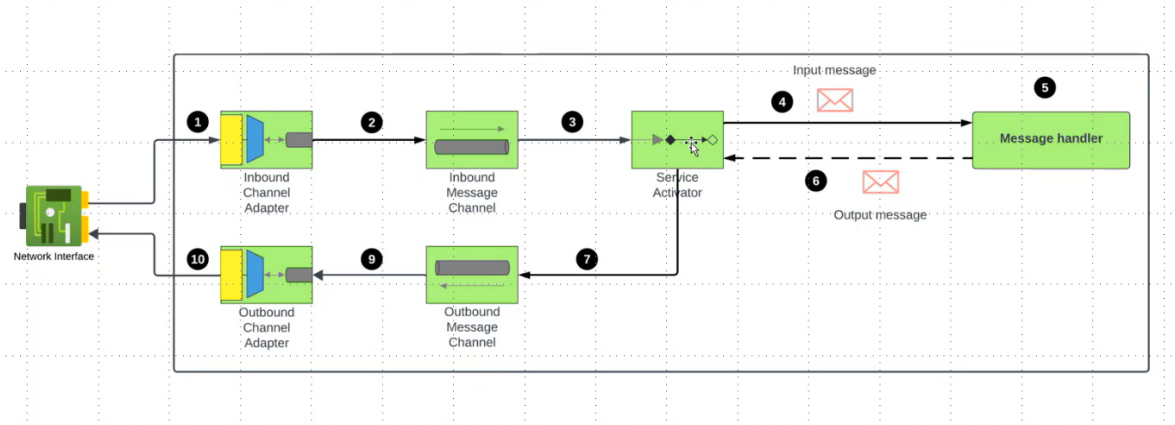


Figure 9 : Explanatory diagram of the entry and exit of the message in tcp/ip

Each received message starts with a structured header: the first 4 bytes indicate the message length, followed by the protocol identifier (ISO), the PowerCARD header, then the message type (MTI) and the primary/secondary bitmaps that indicate the fields present. The data fields follow this format: for example, field 2 (card number) is of type LLVAR, field 3 (processing code) is fixed, field 4 (transaction amount) is 12 digits. Some fields like field 48 (Private Additional Data) use a TLV (Tag-Length-Value) encoding. The bitmap allows to know which fields are present: if the i-th position is equal to 1, the i field is included. This structure guarantees a clear and adaptable reading of ISO8583 messages within PowerCARD.

Figure 10 : Explanatory diagram of the two message flows

This figure illustrates the two possible communication flows between the Mastercard network and the Switch via Channel V4:

- **In acquiring**: it is the Switch that initiates the communication. It generates a transaction message (Authorization, Reversal, Advice, etc.), which passes through Kafka to the Protocol service (where the ISO ↔ CIS mapping is performed), then passes through the Connector which sends the message to the Mastercard network. The response follows the reverse path.
- **In issuing** : here, it is the Mastercard network that initiates the communication. The incoming message first arrives at the Connector, is passed to the Protocol for processing and mapping, and then injected into Kafka to be consumed by the Switch.

Kafka serves as an asynchronous messaging layer enabling resilience and traceability of flows between components.

What is digital banking ?

Digital electronic banking refers to all the technologies that enable secure and traceable electronic payments. It includes bank cards, payment terminals, electronic payment switches, networks (Visa, Mastercard), and internal banking systems.

The main actors in the world of digital banking :

The electronic money system is based on several actors who interact to enable a secure transaction to be carried out:

- **The cardholder (the customer)**: it initiates the card transaction.
- **The merchant**: he receives payment via a payment terminal (TPE or ATM).
- **The acquiring bank**: it provides the payment infrastructure to the merchant.
- **The issuing bank (*issuer*)**: she issued the customer's bank card.

- **The international network (***MasterCard***)**: it acts as a link between the two banks.

The bank switch (PowerCARD at HPS): it plays the role of intermediary

How a transaction flows:

When a customer makes a card payment:

- The terminal sends an authorization request (Request) to the acquiring bank.
- This request goes through the electronic money switch, which translates it into the correct format according to the network protocol (e.g.: CIS for Mastercard).
- The network (Mastercard) then transmits the message to the issuing bank (issuer).
- The issuer validates or rejects the transaction, then returns a response.
- This response is sent back to the merchant.

Position of my solution in this ecosystem:

The project I worked on aims to reimplement this channel in the form of two modern microservices:

- *Protocol*: for the ISO ↔ CIS transformation logic
- *Connector*: for TCP/IP communication with the Mastercard network

My role was to design and develop this smart gateway, capable of processing end-to-end flows:

- **Authorization**
- **Reversal**
- **Advice**
- **File Upload**

In other words, my solution acts as a banking interpreter, ensuring that every message transmitted between actors follows the correct rules, formats and protocols, while ensuring security and compliance.

The ISO 8583 protocol:

- **Definition**

ISO 8583 is a standard communication protocol for exchanging financial messages. It is used by most international networks (Visa, Mastercard, Amex) for authorization requests.

- **Structure of a message**

An ISO 8583 message is composed of three main parts:

MTI (Message Type Indicator): a 4-digit numeric field indicating the general function of the message (authorization request, response, etc.)

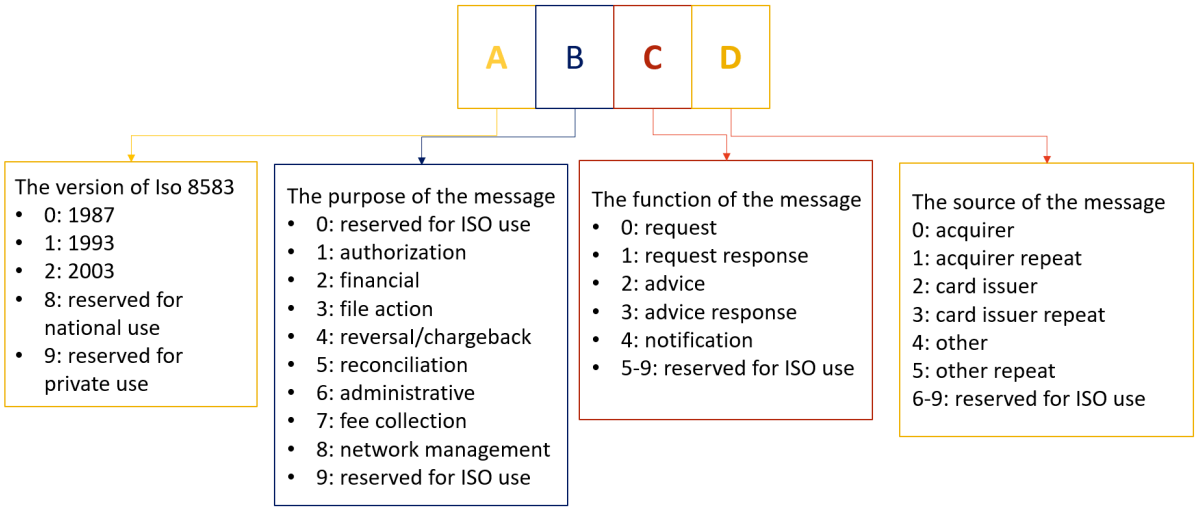


Figure 11 : Explanatory diagram of MTI

Bitmap: A bitmap is used to indicate the data elements present in a message.

The main bitmap, consisting of 64 binary positions (16 in hexadecimal), indicates whether a specific element is present (1) or absent (0).

There can be up to 192 data elements.

Data Elements: The fields containing the actual transactional data, such as amount, PAN, response code, etc.

- **ISO-8583 Field Encoding :**

Undefined: When an ISO-8583 standard field is not used in the interface protocol, its type is defined as Undefined. Even if the field is unused, all its properties must be declared so that, if it is received in the authorization message, the parsing process knows to ignore it and move on to the next field.

TLV (type-length-value or tag-length-value):

The TLV-encoded data stream contains a code identifying the data element, followed by the length of the record value, and then the value itself. The tag and length are fixed-size (usually 1 to 4 bytes), and the value field is variable-size. These fields are used as follows:

Tag: Binary or alphanumeric code identifying the field.

Length: Size of the field value (usually in bytes).

Value: A series of variable-sized bytes containing the data for this part of the message.

BER (Basic Coding Rules):

Is a set of rules specifying a standardized way to represent complex data structures, such as integers, strings, and sequences, in binary format.

TLV encoding can be considered as a specific application of BER, where fixed sizes are assigned to the tag and length.

BER elements, such as field 55, use a TLV structure. Each BER element consists of one or more bytes indicating the data type (tag) of the element, one or more bytes indicating the length of the value, and the encoded value itself.

Bitmapped:

Data encoding: Some fields use bitmap encoding to specify the sub-elements present in the field. Each field using bitmap encoding begins with a bitmap, with each bit representing a potential sub-element. A bit set to ”1” indicates that the corresponding sub-element is included in the field; a ”0” indicates its absence.

After the bitmap, the field contains the actual data for each sub-element identified as present by the bitmap.

The main properties of bitmap-encoded fields are:

- Number of Bitmaps: Indicates the number of bitmap sequences used.
- Bitmap Length: Indicates the total number of bytes in each bitmap.
- Bitmap Format: Indicates whether the bitmap is encoded in ASCII or binary.

- **Channel Architecture V4: Abstract Layers**

Each V4 channel is based on three layers:

- **Abstract Layer**: analysis, logging, communication
- **Custom Layer**: ISO ↔ CIS mapping specific to each network
- **Configuration Layer**: security, addresses, formats

- **Internal Channel Services (Protocol/Connector)**
- **Connector:**

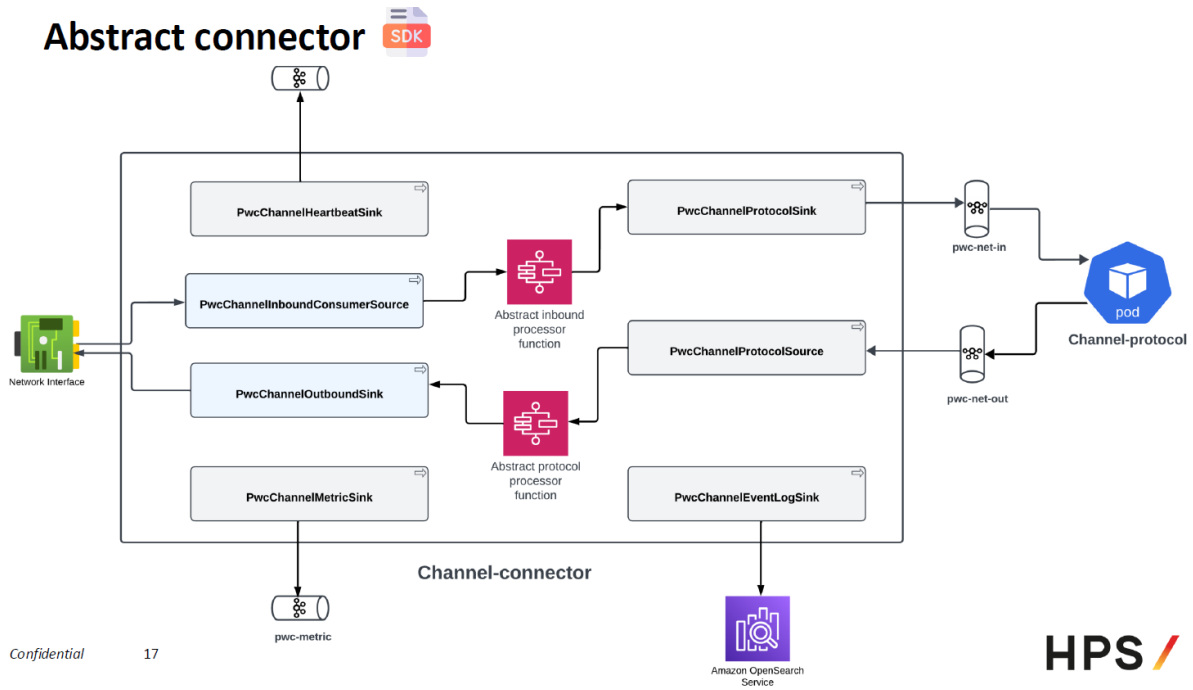


Figure 12 : Diagram of the structure of the abstract connector

- Source: PwcChannelConnectorSource

- Processor: AbstractConnectorProcessorFunction
- Sink: PwcChannelPartitionerSink

Main services:

The connector relies on several essential components that ensure the reliability, monitoring and observability of the service. The PwcChannelHeartbeatSink component sends periodic signals (heartbeat) to check the connector status (online / offline). The PwcChannelEventLogSink logs service lifecycle events (start, sign-on, stop, etc.). The PwcChannelMetricSink keeps track of important metrics, such as message processing times. The PwcChannelContext stores configuration information, the current state of the service and associated metadata.
The embedded RocksDB database allows message and processing context persistence, with TTL (time-to-live) based management. The OpenTelemetry module provides advanced system traceability and observability capabilities. Finally, the ChannelAdapter offers utilities for sending network responses and updating connection status.

- **Protocol**

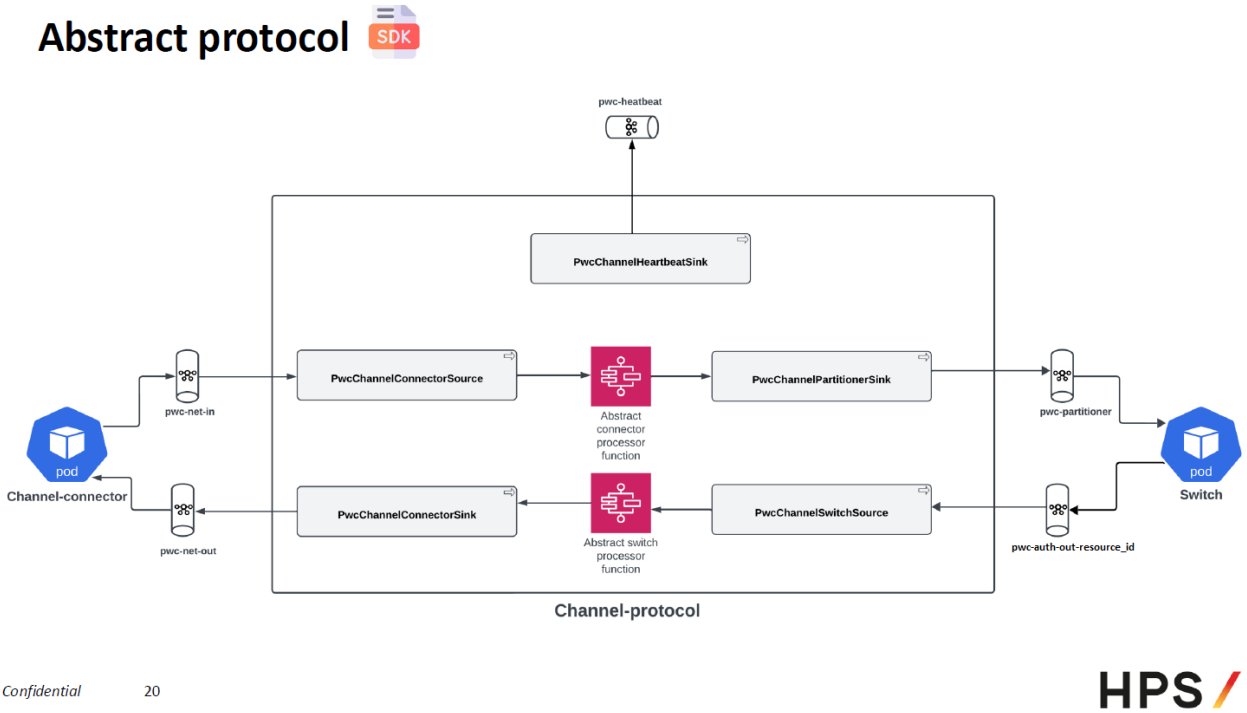


Figure 13 : Diagram of the structure of the abstract protocol

- Source: PwcChannelSwitch Source
- Processor: AbstractSwitchProcessorFunction
- Sink: PwcChannelConnectorSink

Main services

- **PwcChannelConnectorSource**: Receives authorization requests.

- **AbstractConnectorProcessorFunction**: Processes messages before sending them to the switch. This function is specific to each Channel. Developers must implement message type processing and protocol mapping by overloading the methods of the extended class.
- **PwcChannelPartitionerSink**: Sends messages to the appropriate partition of the switch.
- **PwcChannelSwitchSource**: Receives responses from the switch.
- **AbstractSwitchProcessorFunction**: Processes responses received from the switch. This function is specific to each Channel. Developers must implement message type processing and protocol mapping by overloading the methods of the extended class.
- **PwcChannelConnectorSink**: Sends the final response to the connector.
- **PwcChannel Heartbeat Sink**: Monitors the health of the protocol service.

Features

- Heartbeat and readiness check.
- TTL storage with RocksDB for message tracking.
- Propagation of channel context between connector, protocol and switch.
- Full integration with Kafka (connector ↔ protocol ↔ switch).
- Support for administration commands via gRPC.
- Connectivity with HSM module.
- Observability thanks to OpenTelemetry.

Other components:

- Heartbeat, Metrics, RocksDB, OpenTelemetry, HSM
- **The properties of a CIS message:**

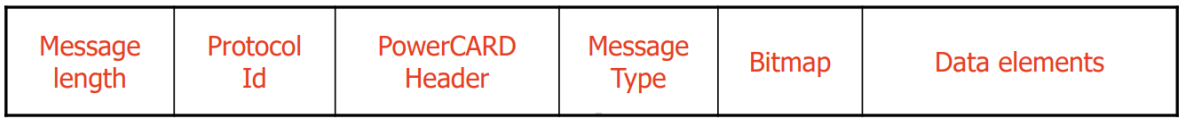


Figure 14 : Diagram of the CIS message structure

This diagram describes the structure of a message as it is processed in the Mastercard CIS Channel. Each message begins with the total message length (Message length), followed by the Protocol ID, and then the PowerCARD header, which contains routing information.

Next is the Message Type (MTI), which defines the nature of the transaction (authorization, reversal, file upload, advice, etc.). The Bitmap specifies which data fields will be included in the message.

Finally, the Data Elements section contains all transactional information (card number, dates, response codes, etc.). This structure is essential to ensure the correct interpretation and compatibility of messages between PowerCARD and the Mastercard network.

1.1.3 Project context:

Problematic :

Migrating the PowerCARD Switch CIS channel is a strategic imperative to modernize the infrastructure and optimize communication with the Mastercard network. The legacy implementation, developed in C, poses major challenges in terms of security, maintainability, and integration with the PowerCARD Switch V4 microservices architecture. Replacing C with Java provides improved security and memory management, increased developer productivity, simplified maintenance, and optimal platform compatibility.

The existing C code, processing messages following the ISO 8583 standard with Mastercard-specific variations, suffers from several critical limitations: increased vulnerability due to manual memory management, a lack of integration with modern microservice technologies (Kafka, gRPC), and high maintenance costs of Docker images due to the need for frequent updates to fix security vulnerabilities and ensure library compatibility. In addition, the scarcity of C++ developers familiar with these modern technologies makes maintaining and evolving this code particularly challenging.

As part of the migration of the PowerCARD Channel solution from version 4 implemented in C to a version 4 rewritten in Java, the integration of the CIS channel is therefore essential. The move to Java/Spring Boot will provide automatic memory management, a rich ecosystem of libraries and tools, better exception handling, and increased portability thanks to the JVM. This migration is crucial to ensure compliance with the latest Mastercard standards, reduce security risks, and enable the rapid introduction of new financial services.

The Switch team, in charge of this development, is facing two major technical difficulties:

The complexity of processing encoded fields: Many message fields use encoding formats such as TLV (Tag-Length-Value) or BER (Basic Encoding Rules). To facilitate the identification of these formats, the team designed a solution based on an XML file, which acts as a data dictionary, describing all the message fields. This XML file does not automatically process the encoding; it provides the information necessary to determine the format (TLV, BER, etc.) and apply the appropriate decoding routines. The combination of this XML file, the detailed documentation (Confluence) and CIS protocol documentation, and the developers' expertise in translating business logic from C code to Java, will allow for an efficient and accurate migration.

The complexity of CIS protocol specifications: The CIS channel reference documents are dense, sometimes unclear, and difficult to use directly. Optimal use of the specifications requires in-depth knowledge of existing code and accurate interpretation of the technical documents.

The central challenge of this report is therefore to migrate the CIS channel to a modern, secure, and maintainable version, taking full advantage of the benefits of Java and microservices architecture, while overcoming the challenges related to the complexity of encoding formats and CIS protocol specifications. The success of this migration is essential to ensure fast, reliable, and secure communication with the Mastercard network and to ensure the future evolution of the PowerCARD Switch platform.

Project objective:

The PowerCARD Switch Customer Interface Specifications (CIS) channel migration project from its current C language implementation to a new Java version has the following objectives:

- *Modernize the technical infrastructure*: Replacing monolithic C code with a Java/Spring Boot microservices architecture to improve the flexibility, scalability and resilience of the PowerCARD Switch platform.
- *Improve code maintainability and security*: Benefit from Java's advantages in terms of automatic memory management, exception handling, and a rich ecosystem of libraries and development tools, thus reducing the risk of vulnerabilities and facilitating long-term maintenance.
- *Ensure seamless integration with existing architecture*: Ensure optimal interoperability between the CIS channel and other Java/Spring Boot microservices of PowerCARD Switch V4, by adopting consistent communication and integration standards.
- *Simplify the processing of ISO 8583 messages*: Implement tools and processes to facilitate the identification and processing of encoded fields (TLV, BER, etc.), by using the XML field description file and capitalizing on the team's expertise.
- *Optimizing communication with the Mastercard network*: Ensure fast, reliable and secure communication with the Mastercard network, in compliance with the latest CIS protocol standards and specifications.
- *Reduce maintenance and operating costs*: Reduce costs associated with maintaining Docker images and resolving security vulnerabilities, through the use of Java and modern development practices.

Achieving these objectives will ensure the sustainability of the PowerCARD Switch platform, improve its operational efficiency and ensure its ability to meet the future needs of the electronic payments market.

Study of the existing situation

-

The current Mastercard CIS message processing system is based on a monolithic architecture developed in C language, integrated into the PowerCARD V4 ecosystem. This historical solution provided the mapping between PowerCARD's internal ISO 8583 format and Mastercard's CIS protocol, used primarily in the processing of authorization, confirmation, file and reconciliation messages.

Although functional, this implementation currently suffers from several structural limitations. The rigid architecture makes maintenance, scalability, and integration with modern components such as Kafka, Docker, or Kubernetes difficult. Additionally, message processing is tightly coupled to the core of the system, making it difficult to add new business rules or third-party protocols.

The environment does not allow for fine-grained transaction traceability or smooth error management, which limits the system's resilience, particularly in the event of load peaks or partial failures.

•

The main shortcomings of the V4 architecture are:

- **Strong coupling of business and network code**: ISO ↔ CIS processing is mixed with TCP communication logic, making the code difficult to modify or test independently.
- **Low scalability**: Monolithic architecture limits the ability to process large volumes or adapt to load peaks.
- **Obsolete technology**: The C language does not allow for smooth integration with modern tools such as Kafka, OpenTelemetry, or cloud monitoring solutions.
- **Lack of modularity**: Every change in the protocol requires recompiling and redeploying the entire binary, which slows down responsiveness.
- **Complex maintenance**: Requires C-skilled developers with good knowledge of legacy code.

Proposed solution
Criticism of the existing

The solution is to migrate the Mastercard CIS channel to a microservices architecture in Java Spring Boot, compatible with the PowerCARD V4 version.

This new architecture is based on two independent microservices:

- **Protocol**: responsible for CIS ↔ ISO 8583 mapping and business logic.
- **Connector**: in charge of TCP/IP network communication with Mastercard.

The advantages of this solution are numerous:

- **Separation of responsibilities**: Business processing is completely separated from network logic.
- **Native integration with Kafka**: For asynchronous, traceable and resilient communication between services.
- **Flexible deployment via Docker/Kubernetes**: For cloud-native management and horizontal scalability.

- **Reusability via SDK**: Critical functions (TLV, TranslatePIN, etc.) are centralized in maintainable SDKs.
- **Modern observability**: Integration with OpenTelemetry, centralized logs, and real-time monitoring.

  This migration represents a fundamental architectural change, making the system more flexible, scalable and ready for future protocols such as BASE1 or MDS.

Indicative planning

The Mastercard CIS Channel migration project to a Java Spring Boot architecture followed a progressive and iterative development cycle, structured around learning the legacy system in C, implementing new Java components, integration testing and stabilization.

The project officially started on February 24, 2025. I took over the code on February 25, followed by an in-depth functional and technical analysis phase, during which I explored the mechanisms of the C code, in particular the functions that manage requests and responses in broadcast mode.

Following this phase, I began implementing the Connector microservice, then quickly implementing the Protocol, after clarifying the use of the TLV format. The development of the Issuing flow (request and response) was carried out in this first step.

During May, I started processing the Acquiring flow, developing the function to convert request message from internal iso to external iso and same function but for response conversion.

In parallel, the test environment was set up with Docker and Kubernetes, allowing the reception of the Country and AcquirerID and Mcc tables, and the execution of integration tests on the implemented flows.

As the academic defense approaches at the end of July, the Authorization flows have been stabilized, and development has been focused on the remaining flows: Reversal, Advice and File Upload, the full implementation of which is planned after the defense.

Diagramme de Gantt

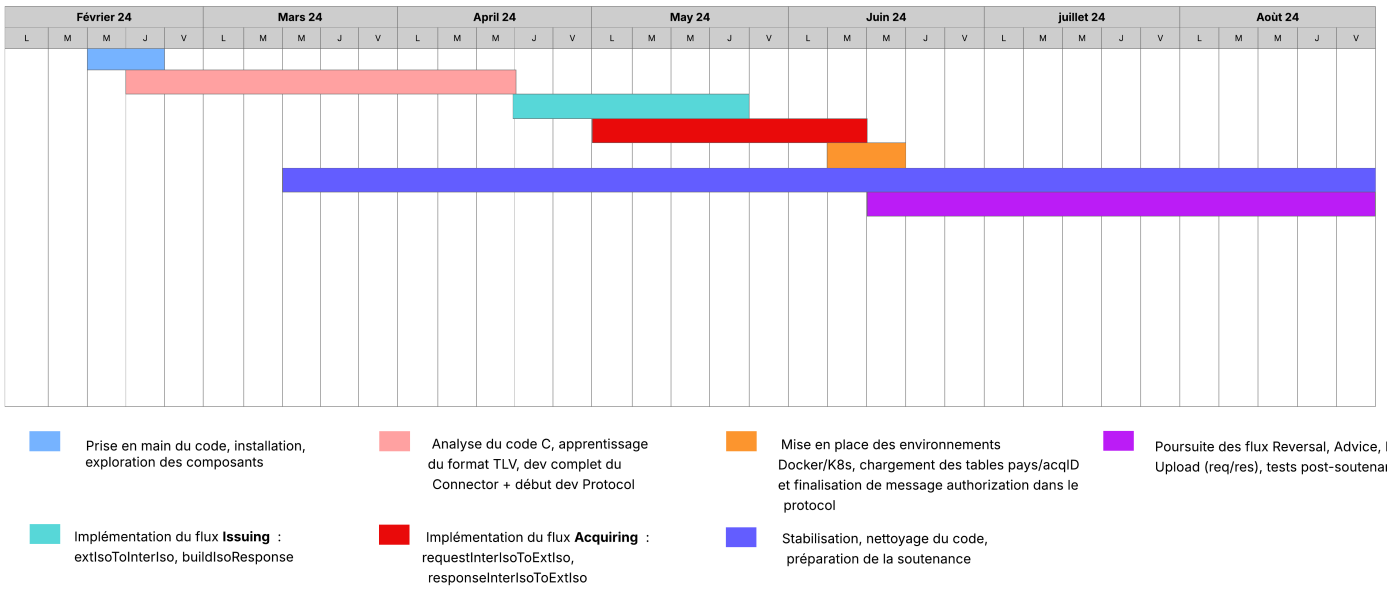


Figure 15 : Gantt Chart Diagram

1.1.4 Conclusion

This second chapter laid the conceptual and technical foundations for the Mastercard CIS Channel migration project to a modern Java Spring Boot microservices architecture. It detailed the challenges of the existing system, the critical limitations of the C implementation, and the functional, technological, and strategic challenges of this transformation.

The study of the existing system highlighted the obstacles related to maintenance, scalability, and integration, which are hindering the system's evolution. In response, the proposed solution based on the separation of responsibilities via Protocol and Connector microservices, the integration of Kafka, and the implementation of a reusable CIS SDK allows for the establishment of a more modular, scalable architecture that complies with modern standards.

The technological choices and the adopted development model guarantee an efficient and sustainable migration, capable of meeting business requirements while preparing the PowerCARD platform for the integration of future protocols.

This chapter thus constitutes the theoretical basis on which the implementation phase, presented in the following chapter, was based.

Chapter 3: Project Analysis and Design

1.2.1 Introduction

This chapter presents the functional analysis and business design of the Mastercard CIS Channel Migration Solution. Its objective is to describe the role of the solution in the overall flow of transactions, the positioning of the main components (Protocol and Connector), as well as the circulation of messages between the different stakeholders of the system (Switch, Mastercard network, issuing banks and acquirers).

The chapter is based on flowcharts and abstract diagrams to explain the expected operation of the solution, without addressing here the technological choices or aspects related to implementation patterns.

1.2.2 Analysis :

1.2.3 Functional requirements:

This section describes the functional requirements of the new PowerCARD Switch Customer Interface Specifications (CIS) channel, focusing on the key components: Message Management, Connector, Protocol, and CIS SDK.

Message management:

The system must support the processing of all message types, in both streams (acquisition and emission), whether requests or responses. The main message types include:

- *Authorisation*: Request for authorization for a transaction (payment, withdrawal, etc.).

- *Advice*: Notification without prior authorization.
- *Reversal*: Cancellation of a previously granted authorization.
- *File Upload*: Deferred transmission of batches of transactions.

Connector:

The Connector is responsible for communication with the Mastercard network and must ensure the following functionalities:

- *Network Communication*: Ensure communication with the Mastercard payment system.
- *Timeout Management*: Manage network message timeouts using a deferred queue.
- *Initial Connection*: Send an initial sign-on message.
- *Heartbeat Management*: Manage heartbeats for liveness and readiness monitoring.
- *Echo Tests*: Manage echo tests using a scheduler.
- *Connection Tracking*TCP/IP: Monitor TCP/IP connections.
- *Channel Event Management*: Manage channel events (start, stop, etc.).
- *Channel Metrics Management*: Collect channel metrics, including response time.
- *Observability*: Integrate OpenTelemetry for observability.
- *Access to Configuration*: Provide a channel context giving access to the channel configuration.
- *Channel Adapter*: Provide a channel adapter providing access to main functions.
- Context Persistence: Ensure context persistence using RocksDB to save CID (Channel Identifier) and OpenTelemetry data to correlate requests and responses.
- *Kafka Communication*: Ensure communication with the Protocol component via Kafka.
- *Administration Order Management*: Manage administrative commands using gRPC communication.
- *TCP/IP Event Interceptor*: Intercept TCP/IP events.

Protocol:

The Protocol is responsible for processing messages and communicating with the Switch and must provide the following functionalities:

- *Managing Original Messages*: Manage incoming and outgoing original messages using RocksDB-based Time-To-Live (TTL) storage.
- *Heartbeat Management*: Manage heartbeats for liveness and readiness monitoring.

- *Context Propagation*: Propagate the channel context between the Connector, the Protocol and the Switch.
- *Observability*: Integrate OpenTelemetry for observability.
- *Access to Configuration*: Provide a channel context giving access to the channel configuration.
- *Channel Adapter*: Provide a channel adapter providing access to main functions.
- *Kafka communication (Connector)*: Ensure communication with the Connector component via Kafka.
- *Kafka Communication (Switch)*: Ensure communication with the Switch via Kafka.
- *HSM Connectivity*: Ensure connectivity with a Hardware Security Module (HSM).
- Administration Command Management: Manage administration commands using gRPC communication.
- *Field Mapping*: Perform mapping of all message fields.

CIS SDK:

The CIS SDK provides tools and functions to facilitate the development and integration of the CIS channel and must ensure the following functionalities:

- *Field Mapping:*
  - Manage message field mapping.
  - Process files requiring specific processing (via Protocol).
- *Unimplemented Functions*: Handle unsupported cases (unimplemented function).
- *Special Cases:*Handle exceptions in fields.
- *Resources*: Provide enumerations and configurations (CIS fields, network messages, special param loader, ...).

1.2.4 Non-functional requirements:

This section describes the non-functional requirements for the new PowerCARD Switch CIS channel. These requirements define the expected qualities and characteristics of the system, independent of its specific functions.

Maintainability: The system must be easy to maintain, modify and evolve, with clear, modular and well-documented code, thus minimizing long-term maintenance costs.

Security: The system must ensure the security of data and transactions, protecting against unauthorized access, vulnerabilities, and potential attacks. This involves robust authentication mechanisms, encryption of sensitive data, and access management.

Testability: The system must be easily testable at all levels (unit, integration, system), with mechanisms to automate tests and ensure code quality.

Reliability: The system must be reliable and available, minimizing downtime and ensuring service continuity. This involves error management, redundancy, and disaster recovery mechanisms.

Productivity: The system must enable high developer productivity, by offering efficient tools and frameworks, as well as clear and complete documentation.

Integrability: The system must integrate easily with other components of the PowerCARD Switch architecture, respecting integration standards and protocols.

Scalability: The system must be scalable, that is, capable of handling an increasing workload without performance degradation. This requires an architecture capable of adapting to variations in traffic and resources.

Observability: The system must be observable, that is, capable of providing detailed information about its state and operation, thus allowing for rapid diagnosis of problems and optimization of performance. This involves the collection of metrics, logs, and traces.

1.2.5 Use case diagram:

•

Actor

As part of the migration of the Mastercard CIS Channel to the V4 architecture, the main players identified are:

- **Mastercard Network**: the external network that exchanges messages with our system via the CIS protocol, using TCP/IP communication.
- **Switch (PowerCARD)**: the internal microservice that receives or sends ISO 8583 messages to financial institutions (issuing or acquiring banks).

These two actors interact directly with the Mastercard CIS Channel, which acts as a translator and router between the two message formats.

•

Main use cases

The main use cases that structure the operation of the solution are:

- **Process an incoming message from the Mastercard network**
  - Receive a CIS message via the Connector
  - Map the CIS message to an internal ISO 8583 format via the Protocol
  - Transmit the mapped message to the Switch
- **Process an outgoing message from the Switch**

- Receive an internal ISO 8583 message via the Protocol
- Map ISO 8583 message to CIS format
- Send the mapped message to the Mastercard network via the Connector

These use cases cover all expected functional flows, whether authorization, reversal, advice or file upload messages.

•

Use Case Diagram:

The diagram above illustrates the main use cases of the Mastercard CIS Channel (V4) in the context of the migration carried out.

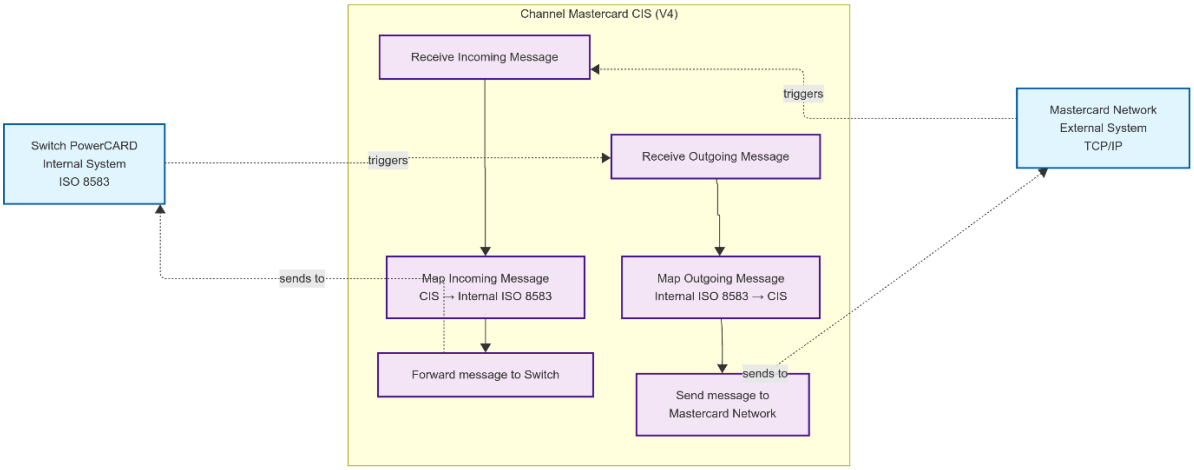


Figure 16 : Use Case Diagram Schema

The system is positioned between two main actors: on one side, the Mastercard network (external actor) which communicates via a CIS protocol encapsulated on TCP/IP; on the other, the PowerCARD Switch (internal actor) which manages exchanges with financial institutions in ISO 8583 format.

Two major functional scenarios are represented:

- **Incoming flow (Acquiring)** : The Mastercard network initiates communication by sending a CIS message. The Connector receives it (Receive Incoming Message), then the Protocol performs a detailed mapping (Map Incoming Message CIS → Internal ISO 8583), taking into account the business rules and the specificities of the internal protocol. The message thus transformed is then transferred to the Switch (Forward message to Switch), which processes it or relays it to the relevant bank.
- **Outgoing flow (Issuing)** : Here, the PowerCARD Switch initiates the request (authorization, reversal, advice, etc.). The ISO message is sent to the Protocol (Receive Outgoing Message), where it is converted to the CIS format expected by Mastercard (Map Outgoing Message Internal ISO 8583 → CIS). The Connector then ensures the message is sent to the Mastercard network (Send message to Mastercard Network).

This diagram highlights the bidirectional nature of the Channel, which acts as a translator between two heterogeneous protocol worlds, while ensuring the conformity of the exchanged messages. It also illustrates the logical independence of the Protocol and Connector components, each assuming a specific role in the processing chain.

1.2.6 Sequence diagram

•

General acquiring case:

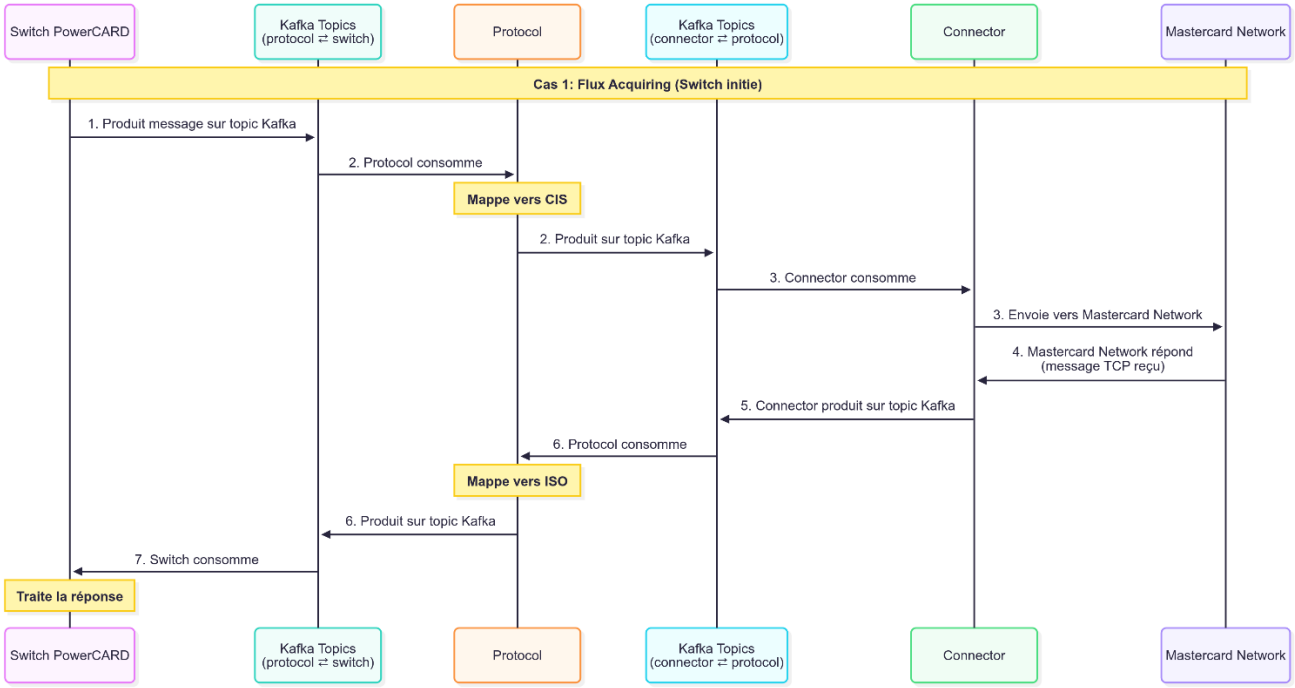


Figure 17 : General Sequence Diagram

The presented diagram illustrates the complete flow of an Acquiring authorization message in the migrated PowerCARD to Mastercard CIS architecture, initiated by the Switch.

This case corresponds to a purchase transaction carried out by a cardholder with a merchant (POS, e-commerce), where the PowerCARD Switch initiates the flow to the Mastercard network.

Step 1: Generate the authorization message

The PowerCARD Switch generates an internal ISO 8583 authorization message, corresponding to a transaction processing request (purchase, authorization).

This message is published on a specific Kafka topic (protocol ↔ switch topic), facilitating communication between the Switch and the protocol module (Protocol).

Step 2: Consumption and transformation to CIS

The Protocol consumes this message and transforms it into the CIS format used by the Mastercard network.

This mapping is achieved through a complex business logic BuildAutReqToNw inspired by C code migrated to Java.

The produced CIS message is then published to a second Kafka topic (connector ↔ protocol) to allow the Connector to use it.

Step 3: Network Transmission

The Connector consumes this CIS message and physically transmits it to the Mastercard network via TCP communication (socket).

This layer provides encapsulation, session management, timeouts, and network compliance.

Step 4: Mastercard Network Response

The Mastercard network processes the request and returns a response (TCP message received by the Connector), which will be consumed and transformed into a return.

Step 5: Publication of the CIS response

The Connector produces a CIS response message on the Kafka topic (connector ↔ protocol).

Step 6: Remapping to ISO

The Protocol consumes the CIS response and transforms it back into the internal ISO 8583 format, understandable by the Switch.

Step 7: Processing by the Switch

The PowerCARD Switch consumes this ISO response and processes it (validation, database update, return to the POS or e-commerce).

-

General issuing case:

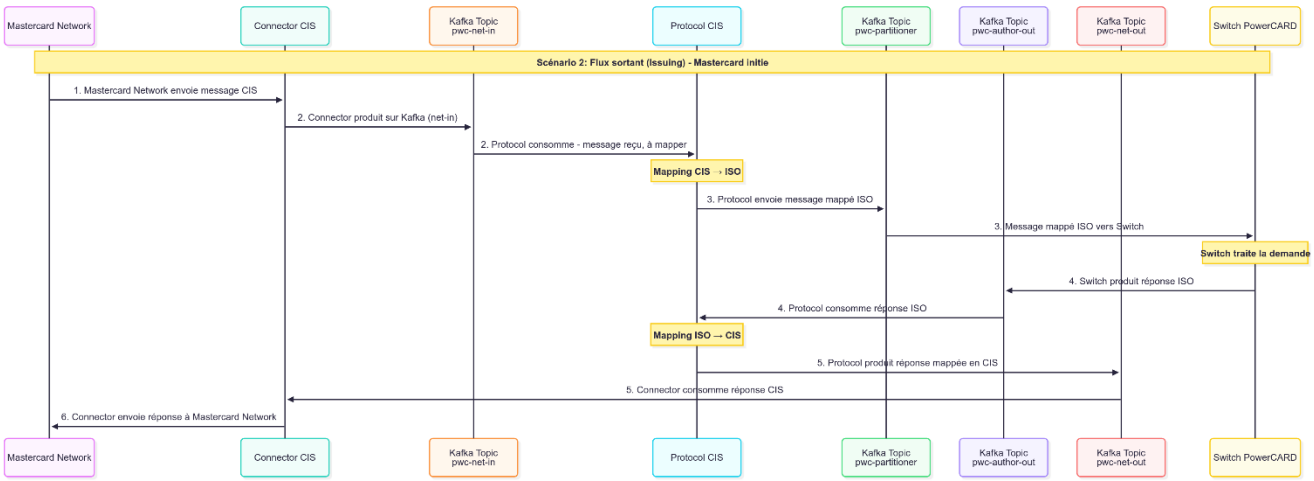


Figure 18 : Sequence diagram of emission case

This diagram illustrates the flow of a request initiated by the Mastercard network to PowerCARD in Issuing mode (Switch acting as the card issuer).

In this case, it is the Mastercard network that sends an incoming message to the PowerCARD Switch (for example an Advice 0120 or a Reversal 0420).

Detailed procedure:

Step 1: Receiving the network message

The Mastercard Network sends a CIS message (Mastercard format) in TCP mode to the CIS Connector.

Step 2: Publish to Kafka

The Connector receives this message, transforms it into a Kafka message and publishes it to the pwc-net-in Kafka topic.

Step 3: Consumption by Protocol

The CIS Protocol consumes this Kafka pwc-net-in message.

It performs a CIS → ISO mapping (eg: BuildAutReqFromNw or ReversalReqFromNw) to convert the CIS message into the internal ISO 8583 format, usable by PowerCARD.

The ISO mapped message is then published to the pwc-author-out topic.

Step 4: Switch consumes the request

The PowerCARD Switch consumes this ISO message.

It performs the corresponding business processing (authorization update, balance check, database modification, etc.).

The Switch then produces an ISO 8583 response, which is published to the Kafka topic pwc-net-out.

Step 5: ISO → CIS Remapping

The CIS Protocol consumes this ISO response, and performs an ISO → CIS mapping to reconstruct the response in Mastercard format (eg: ReversalRepToNw, AdvRepToNw).

This CIS response is published to Kafka (pwc-net-out).

Step 6: Send to Mastercard Network

The CIS Connector consumes the CIS response from Kafka, and physically sends it via TCP to the Mastercard network.

•

Case of acquiring authorization:

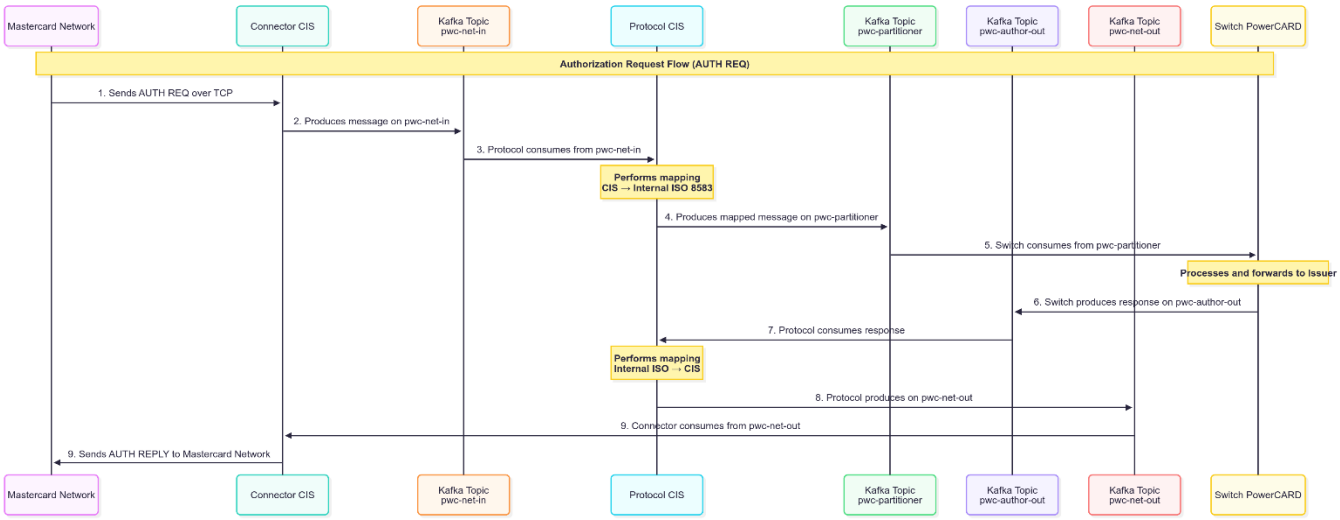


Figure 19 : Sequence diagram for autorisation in acquisition

This diagram shows the complete flow of an authorization request (AUTH REQ) initiated by Mastercard, typically a 0100 message.

In this scenario, the Mastercard network sends a message to obtain authorization for a real-time transaction (card payment). The PowerCARD system acts as a Switch + Issuer.

Detailed procedure:

Step 1: Receiving the Mastercard request

The Mastercard Network sends a CIS authorization message (AUTH REQ) to the CIS Connector over TCP.

Step 2: Kafka Publishing

The CIS Connector produces the message received on the Kafka topic pwc-net-in.

This topic is listened to by the CIS Protocol.

Step 3: Consume the CIS message

The CIS Protocol consumes the message from the pwc-net-in topic.

Step 4: CIS → ISO 8583 Mapping

The CIS Protocol converts the message into ISO 8583 internal format:

He then posts this mapped message to the pwc-partitioner topic.

This topic allows the message to be distributed to the correct partitioner according to the business logic (card, bank, etc.).

Step 5: Processing in the PowerCARD Switch

The PowerCARD Switch consumes the message on pwc-partitioner.

The Switch processes the request:

- Checking authorization rules
- Balance Check
- Anti-fraud control

Transmission to the banking core or to the issuer in the event of outsourcing.

Step 6: Production of the response

The Switch produces the ISO 8583 response (AUTH REPLY) on the pwc-author-out topic.

Step 7: Consume the response

The CIS Protocol consumes the response from pwc-author-out.

Step 8: ISO → CIS Mapping

The CIS Protocol performs the reverse mapping (ISO → CIS) to reconstruct the CIS response to return to the Mastercard network.

He produces this response on the pwc-net-out topic.

Step 9: Sending the response

The CIS Connector consumes the message on pwc-net-out and sends the AUTH REPLY response to the Mastercard network via TCP.

•

Case authorization issuing:

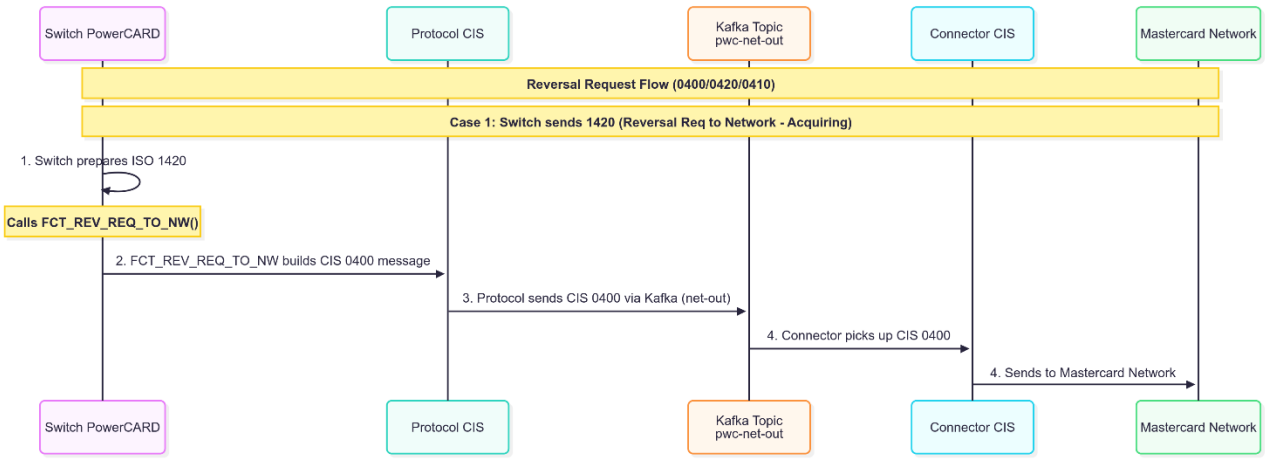


Figure 20 : sequence diagram for authorization issuing case

This diagram describes the reversal request flow initiated on the PowerCARD Switch side for an acquiring mode transaction.

Typically, this is an ISO 1420 → message transformed into CIS 0400 sent to Mastercard Network to signal a reversal of a previous transaction (example: cancellation of a payment).

Detailed procedure:

Step 1: Preparation of ISO 1420

The PowerCARD Switch prepares an ISO 1420 message (reversal request).

This message is generated in response to internal logic (timeout, cancellation, confirmation failure, etc.).

Step 2: CIS 0400 Construction

Constructs a message in CIS 0400 format (format expected by the Mastercard network).

This conversion follows business rules (ISO → CIS field mapping).

Step 3: Send Kafka

The CIS Protocol publishes the CIS 0400 message to the Kafka topic: pwc-net-out

This topic allows routing to the CIS Connector.

Step 4: Send to the Mastercard network

The CIS Connector consumes the CIS 0400 message from pwc-net-out → then sends it via TCP/IP to the Mastercard network.

•

Reversal issuing case:

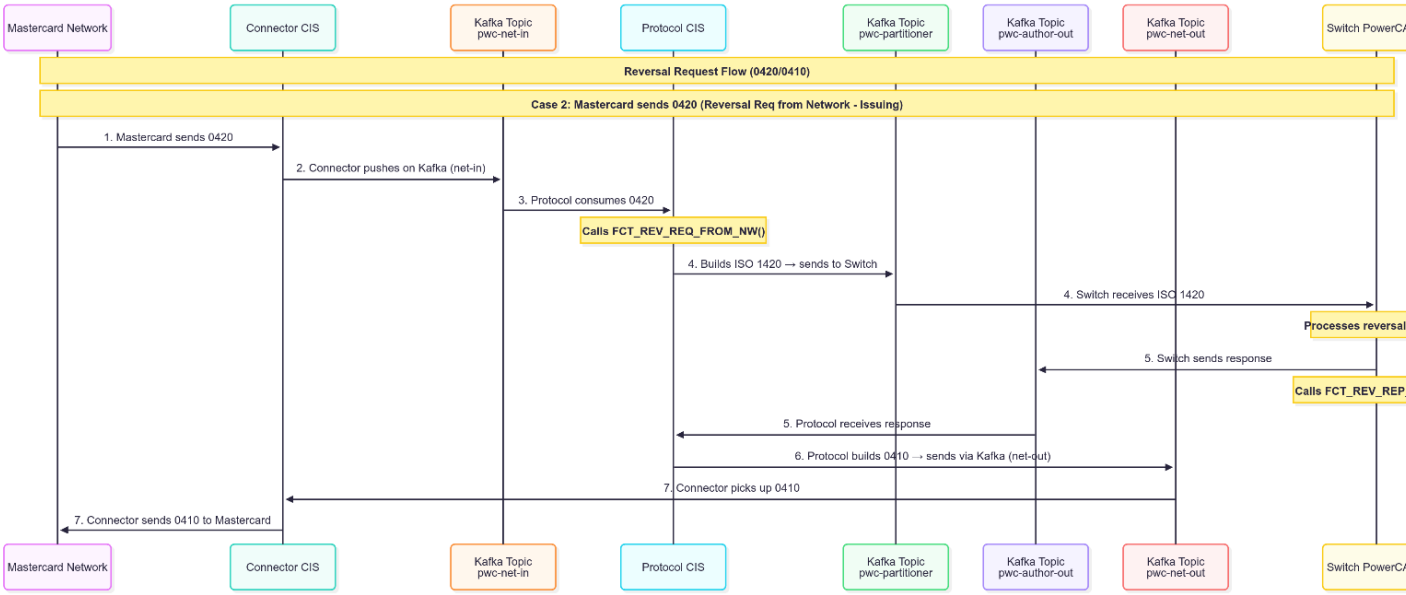


Figure 21 : Sequence Diagram of the reversal acquisition

This diagram illustrates the processing of a reversal request initiated by the Mastercard network to the PowerCARD Switch, in issuing mode.

Typically, the Mastercard network wants to cancel a transaction, so it sends a 0420 message to the system.

Step 1: Send by Mastercard

The Mastercard network sends a 0420 (Reversal Request) message to the system.

Step 2: Kafka Publishing

The CIS Connector receives the 0420 and publishes it on the Kafka topic:

pwc-net-in

Step 3: Consumption by protocol

The CIS Protocol consumes this 0420 message.

Step 4: ISO 1420 Construction

Constructed from an ISO 1420 message

Step 5: Reception and processing

The PowerCARD Switch receives the ISO 1420, processes the reversal request, and then prepares a response.

Step 6: Construction 0410

The CIS Protocol receives this ISO response, constructs a CIS 0410 (Reversal Response) message, and publishes it to the Kafka topic:

pwc-net-out

Step 7: Transmission to Mastercard

The CIS Connector consumes the 0410 message from Kafka, and sends it to the Mastercard network in response to the initial request.

•

Advice acquiring case:

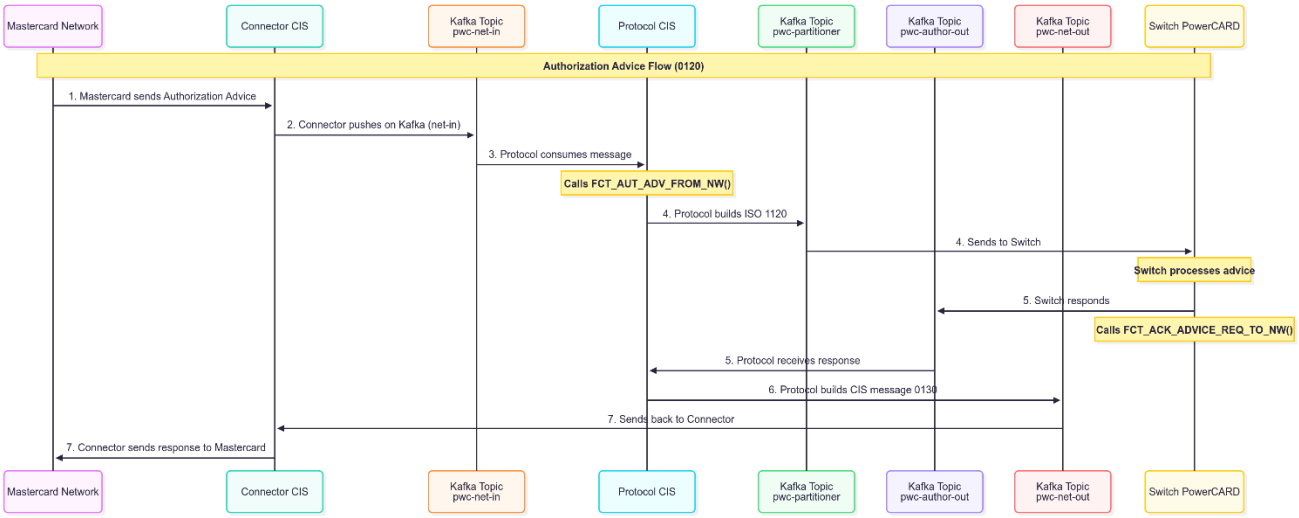


Figure 22 : sequence diagram of reversal acquisition

This diagram describes the Authorization Advice (0120) message flow initiated by the Mastercard network, in acquiring mode. This type of message is used to notify the Switch that a transaction has been authorized or regularized (e.g. offline transactions, correction, supplement).

Step 1: Receiving 0120

The Mastercard network sends an Authorization Advice 0120 message via TCP to the CIS Connector.

Step 2: Injection into Kafka

The CIS Connector publishes this message 0120 to the Kafka topic pwc-net-in for processing.

Step 3: Processing by the CIS Protocol

The CIS Protocol consumes the 0120 message and calls the function which is responsible for performing the reversal logic in acquisition.

Step 4: Convert to ISO 1120

The protocol converts the message into ISO 1120 format, the format expected by the PowerCARD Switch, and sends it to the Switch.

Step 5: Processing by the Switch

The PowerCARD Switch receives the ISO 1120 message and processes the authorization advice (update of databases, regularization, etc.).

Step 6: Construction of the 0130

The CIS Protocol constructs the CIS 0130 message (response to Advice) and publishes it to the Kafka topic pwc-net-out.

Step 7: Return to Mastercard

The CIS Connector consumes the 0130 message and transmits it to the Mastercard network, finalizing the advice processing.

•

File upload issuing case:

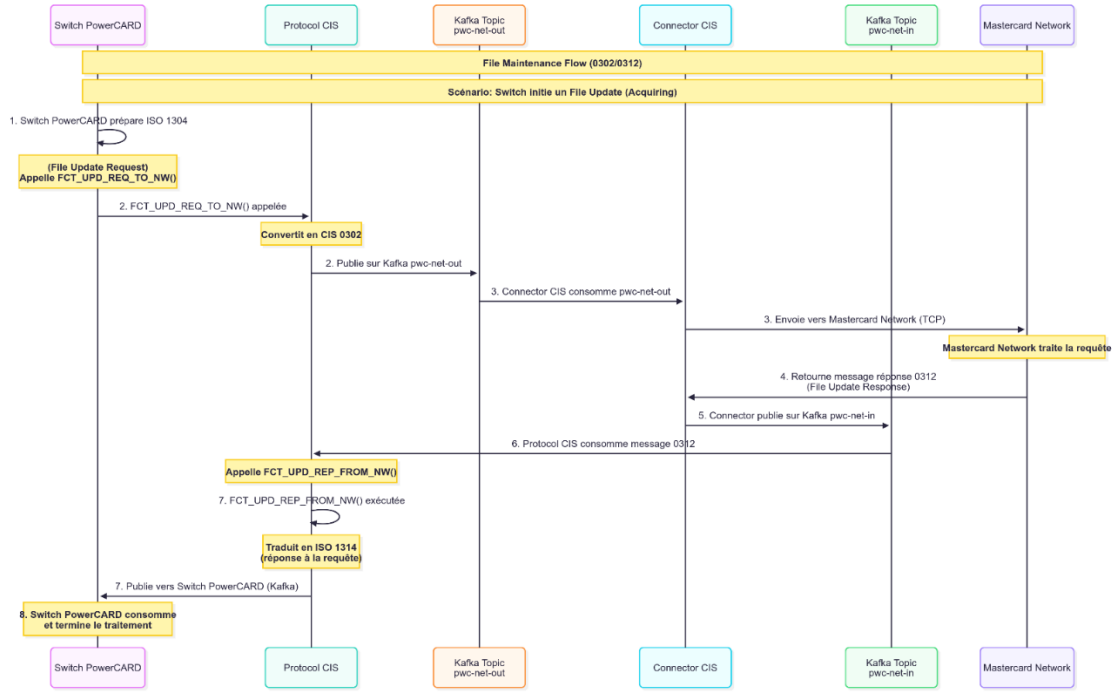


Figure 23 : sequence diagram of the emission authorization

This diagram describes the Authorization Advice message flow in issuing mode, initiated by the Mastercard network. This is a CIS 0120 message sent by Mastercard → which will be transformed into an ISO 1120 message to the PowerCARD Switch, to inform of an authorized operation (authorization advice, non-blocking).

Step 1: Receiving message 0120

The Mastercard network sends an Authorization Advice 0120 message via TCP to the CIS Connector.

Step 2: Injection into Kafka

The CIS Connector publishes this message to the Kafka topic pwc-net-in, for processing by the protocol.

Step 3: Processing by the CIS Protocol

The CIS Protocol consumes message 0120. It calls the function which:

- Converts the message to ISO 1120
- Inserts the original into the temporary table so that the ACK can be processed later.

The ISO 1120 message is sent to the PowerCARD Switch.

Step 4: Processing in the Switch

The PowerCARD Switch receives the ISO 1120 message and processes the advice (internal update, log, statistics, etc.). The Switch then returns an ISO 1130 or equivalent response message (indicating success or other status).

Step 5: Preparing the ACK

The CIS Protocol receives this response, calls a function which:

- Retrieves the original query.

- Constructs the CIS 0130 response.

Step 6: Return to Mastercard
The CIS Connector then forwards this CIS 0130 message to the Mastercard network, finalizing the advice flow.

-

File upload in acquisition:

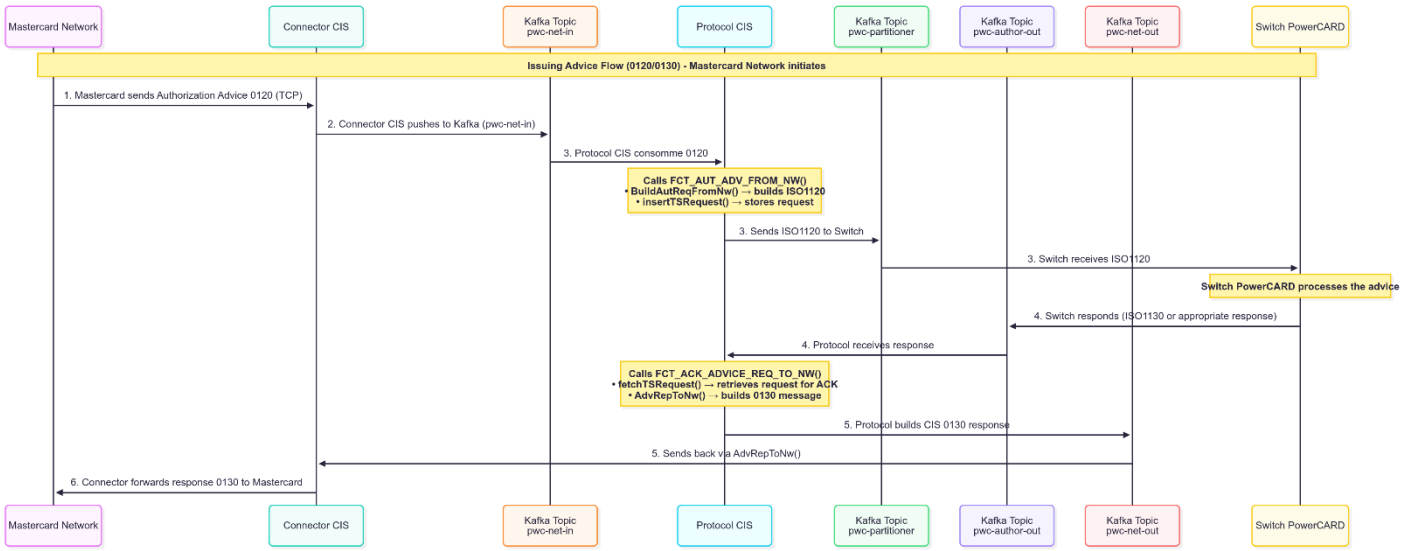


Figure 24 : File upload sequence diagram

This diagram describes the file maintenance request flow initiated on the PowerCARD Switch side in acquiring mode. This is typically an ISO 1304 message (file update request) → transformed into a CIS 0302 message sent to the Mastercard network to perform an update on a remote file (addition, modification, deletion or information request).

Detailed procedure:

Step 1: Preparation of ISO 1304
The PowerCARD Switch prepares an ISO 1304 message representing a file update request. This message is constructed based on the operations requested by the back office or management system.

Step 2: CIS 0302 Construction
Constructs a message in CIS 0302 format (format expected by the Mastercard network). This conversion respects the mapping rules between ISO and CIS fields.

Step 3: Kafka Publishing
The CIS Protocol publishes the CIS 0302 message on the Kafka topic: pwc-net-out. This topic is used to transmit the message to the CIS Connector.

Step 4: Send to Mastercard
The CIS Connector consumes the 0302 message from pwc-net-out and sends it via TCP/IP to the Mastercard network.

Step 5: Return response 0312
The Mastercard network processes the file update request and returns a CIS 0312 response (file update response). The CIS Connector receives this response and publishes it to Kafka: pwc-net-in.

Step 6: Processing response 0312
CIS Protocol consumes response 0312 from pwc-net-in

Step 7: Convert to ISO 1314
Convection of the CIS response into an ISO 1314 message (response to the file update request), and publishes it to Kafka for the Switch.

Step 8: Finalization
The PowerCARD Switch consumes the ISO 1314 message and completes processing the update request.

1.2.7 Activity diagram

•

Request for authorization for acquisition (Acquiring):

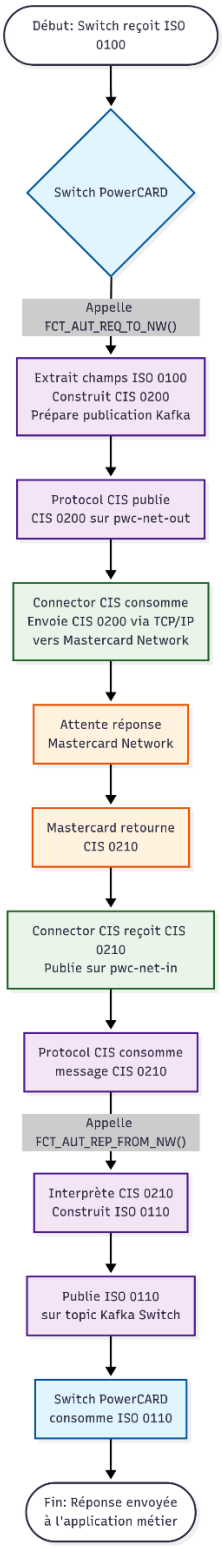


Figure 25 : Activity Diagram Acquisition Authorization Request

This diagram illustrates the complete processing of an Authorization Request message in Acquiring mode, from its reception by the Switch to the return of the network response.

The flow begins with the PowerCARD Switch receiving an ISO 0100 message. Upon receipt, the Switch triggers a function that extracts the relevant fields from the ISO message,

constructs a CIS 0200 message, and publishes it to the pwc-net-out Kafka topic.

The CIS Connector consumes this message and then sends it to the Mastercard network via TCP/IP. Once the request is sent, the system enters a phase where it waits for the network response. When Mastercard returns a CIS 0210 message, it is received by the CIS Connector and published to the pwc-net-in Kafka topic.

The CIS Protocol then consumes the 0210 message and calls a function that interprets the CIS 0210 response and constructs the response message in ISO 0110 format. The ISO 0110 message is published to the Kafka topic intended for the Switch.

Finally, the Switch consumes this message and transmits the response to the business application. The processing cycle is thus complete, ensuring accurate and structured tracking of the Authorization Request Acquiring flow.

•

Request for Issuing Authorization:

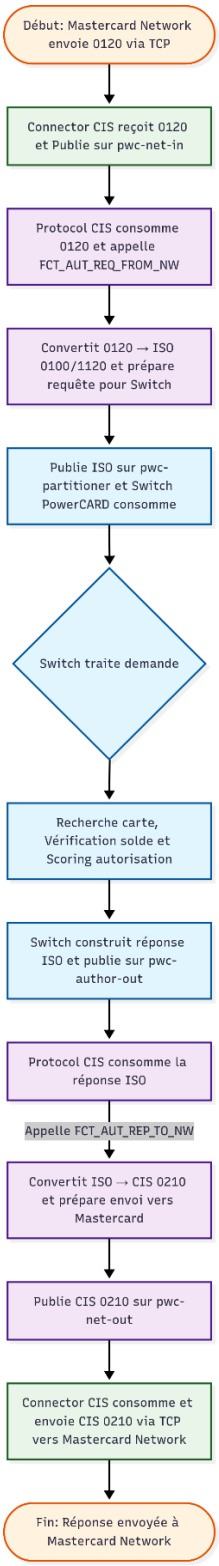


Figure 26 : Activity Diagram Schema Authorization Request Issuing

This diagram describes the processing of an Authorization Advice message in Issuing mode, where the Mastercard network initiates the communication.

The process begins when Mastercard Network sends an Authorization Advice 0120 mes-

sage via TCP. The CIS Connector receives this message and publishes it to the pwc-net-in Kafka topic. Then, the CIS Protocol consumes this message and calls a function that performs the CIS → ISO conversion (usually to ISO 0100/1120), preparing a compliant message for the PowerCARD Switch.

The Switch consumes this ISO message and performs its business processing: balance checks, scoring, authorization or refusal of the request. Once the decision is made, the Switch prepares a response (ISO 0110/1130) and publishes it to Kafka (pwc-author-out).

The ISO response is then consumed by the CIS Protocol, which calls a function to construct the CIS 0130 response. This response is published to Kafka (pwc-net-out), consumed by the CIS Connector, and then sent back to the Mastercard network.

This flow ensures complete monitoring of the Authorization Advice initiated by Mastercard, while respecting the internal conversions and business logic of the Switch.

•

Acquisition reversal:

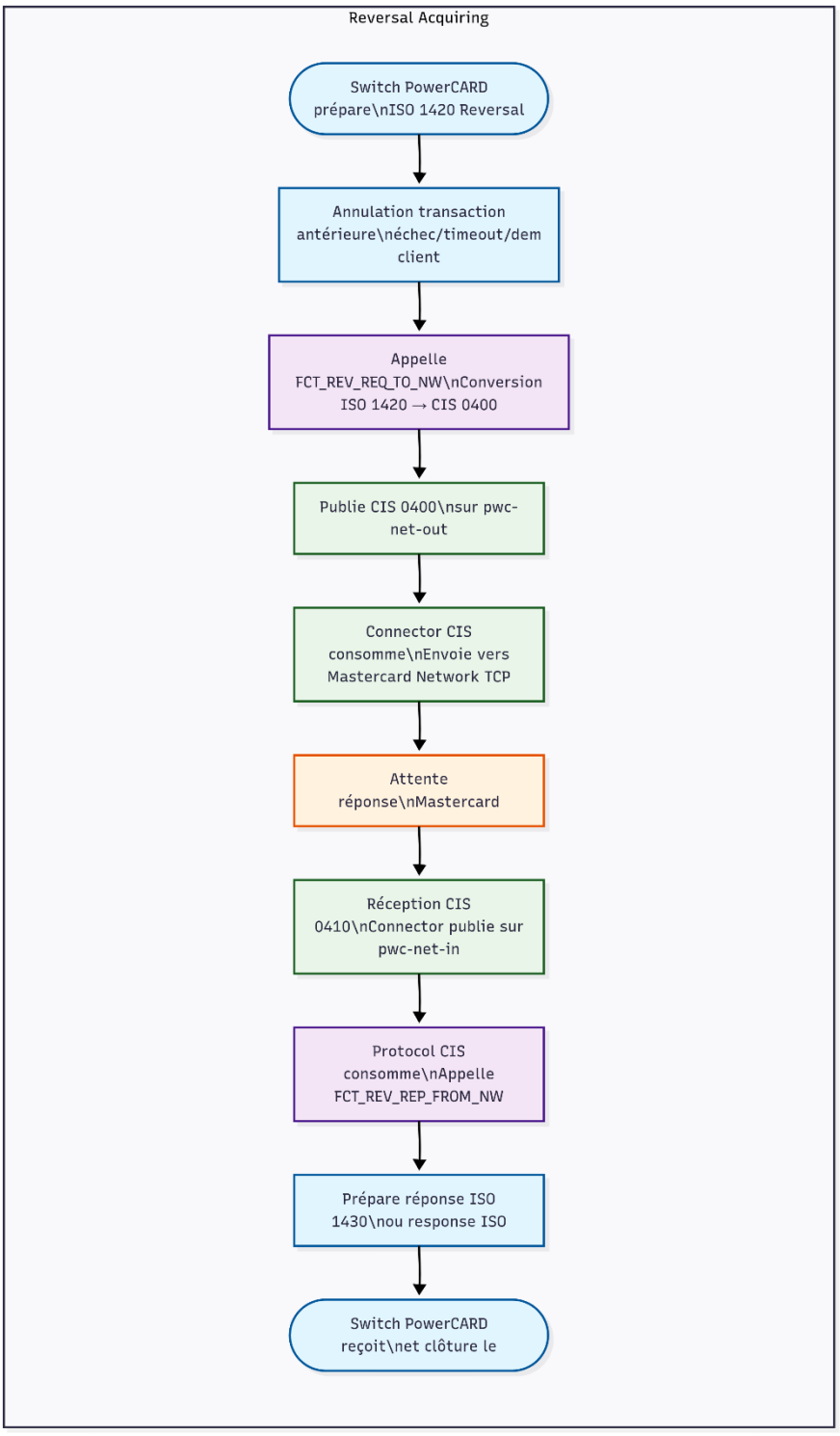


Figure 27 : Activity Diagram Reversal Acquisition

This diagram describes the processing flow of a Reversal Acquiring message, in the event that the PowerCARD Switch initiates a request to cancel a previous transaction to the Mastercard network.

The process starts when the PowerCARD Switch prepares an ISO 1420 Reversal Request message, often triggered by a failure, timeout, or client cancellation request.

The Switch then calls the function that converts this ISO 1420 message into a CIS 0400 message that conforms to the Mastercard protocol.

This CIS 0400 message is published to the pwc-net-out Kafka topic and consumed by the CIS Connector, which then routes it via TCP to the Mastercard network.

The system then enters the response waiting phase. Upon receipt of the CIS 0410 response by Mastercard, the CIS Connector publishes this message to the pwc-net-in Kafka topic.

The CIS Protocol consumes this response, calls the function that constructs the ISO response (ISO 1430 or other as appropriate).

Finally, the PowerCARD Switch consumes this ISO response via Kafka and closes the Reversal processing, thus updating its internal databases.

•

Reversal of issue (Issuing):

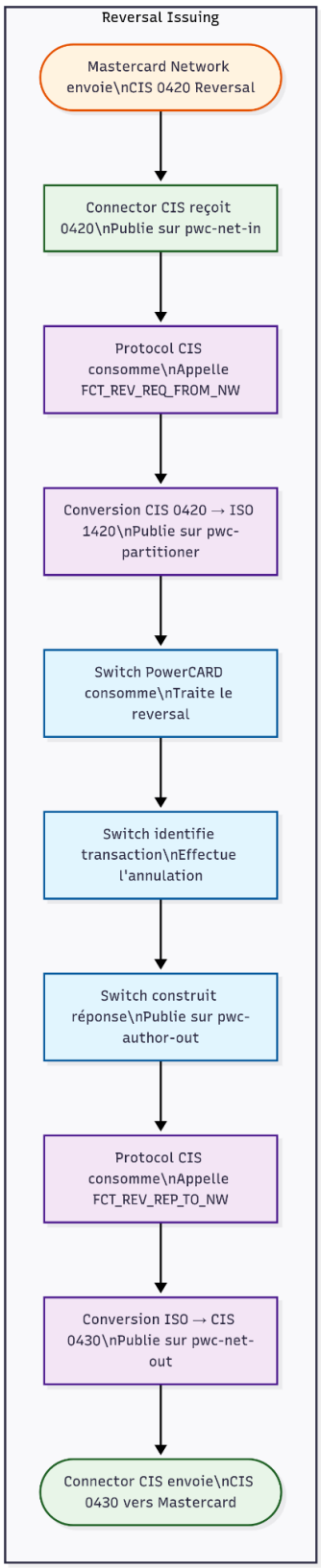


Figure 28 : Activity Diagram Query Reversal Issuing

This diagram illustrates the processing flow of a Reversal Issuing, i.e. a case where it is the Mastercard network that initiates a request to cancel a transaction by sending a CIS 0420 message.

The process begins with the reception, in TCP, of the CIS 0420 message by the CIS Connector, which then publishes it on the Kafka topic pwc-net-in.

The CIS Protocol consumes this message, calls the function that handles the issuance of a request and converts the message from CIS 0420 to ISO 1420. The ISO message is then published on the pwc-partitioner topic for processing by the PowerCARD Switch.

The Switch consumes this ISO message, identifies the transaction concerned and executes the corresponding cancellation operation (update of statuses, accounting cancellation, etc.).

It then prepares an ISO response and publishes it to the Kafka topic pwc-author-out.

This response is consumed by the CIS Protocol, which calls a function that converts it to a CIS 0430 message and then publishes it to pwc-net-out.

Finally, the CIS Connector sends this CIS 0430 message to the Mastercard network, signaling that the cancellation has been successfully processed on the Switch side.

•

Authorization Advice:

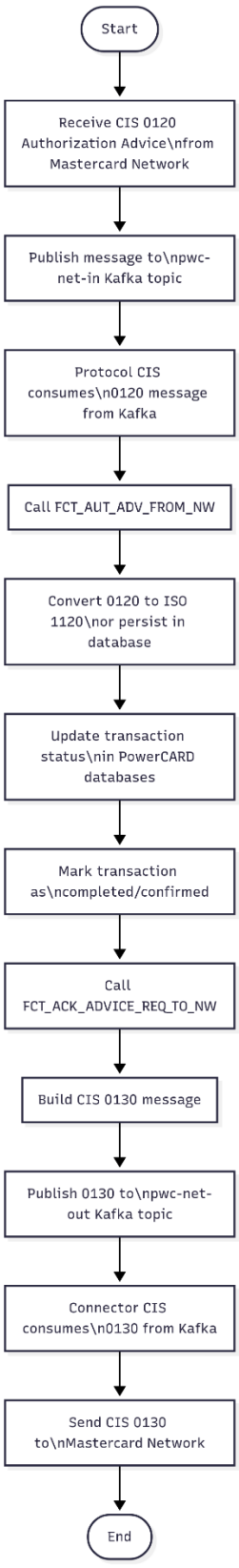


Figure 29 : Activity Diagram Query Reversal Issuing

This diagram describes the processing flow of an Authorization Advice (CIS 0120) message, typical of issuing mode, when the Mastercard network initiates notification of a transaction status.

The process begins with the CIS Connector receiving a CIS 0120 message via TCP. The CIS Connector publishes the message to the pwc-net-in Kafka topic.

The CIS Protocol consumes the message, triggers the function that processes the emission mode for the advice message type, and performs the conversion from CIS 0120 to ISO 1120. This conversion is persisted in the PowerCARD database.

The system then updates the status of the transaction in the database (confirmed or finalized status). The transaction is marked as completed to prevent any subsequent reprocessing attempts.

Then, a function is called to prepare the response. The protocol constructs a CIS 0130 message (acknowledgement of the Advice) and publishes it to the Kafka topic pwc-net-out.

Finally, the CIS Connector consumes this CIS 0130 message and returns it to the Mastercard network, thus closing the Advice flow.

•

File upload :

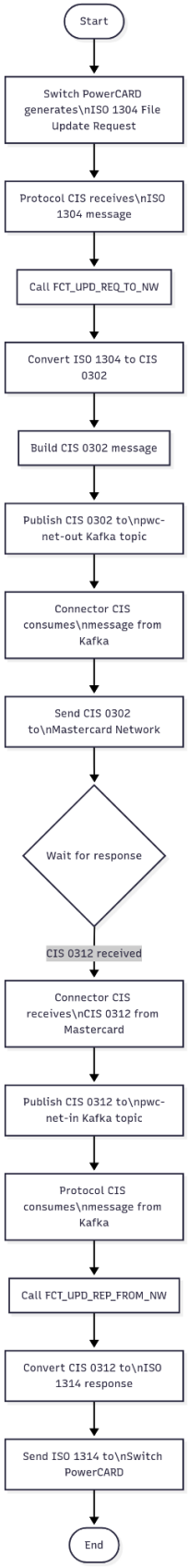


Figure 30 : Activity Diagram Query Reversal Issuing

This diagram represents the processing of a File Update request initiated by the PowerCARD Switch in Acquiring mode.

The flow begins when the Switch generates an ISO 1304 File Update Request message, based on an internal need (file update, configuration, etc.).

The CIS Protocol receives this ISO 1304 message, calls the function that processes the file upload in transmit mode to convert it into a CIS 0302 message, and constructs the appropriate message.

The CIS 0302 is then published on the Kafka topic pwc-net-out to the CIS Connector.

The Connector consumes this message and sends it to the Mastercard network via TCP/IP.

Once the request is processed, Mastercard returns a CIS 0312 response message.

The CIS Connector receives this response and publishes it to the Kafka topic pwc-net-in.

The CIS Protocol consumes the message, calls the function that processes the message file upload response in transmit mode, and performs the conversion of the CIS 0312 response to an ISO 1314 message.

Finally, this ISO 1314 message is sent to the PowerCARD Switch to finalize the update cycle.

1.2.8 Architecture Channel

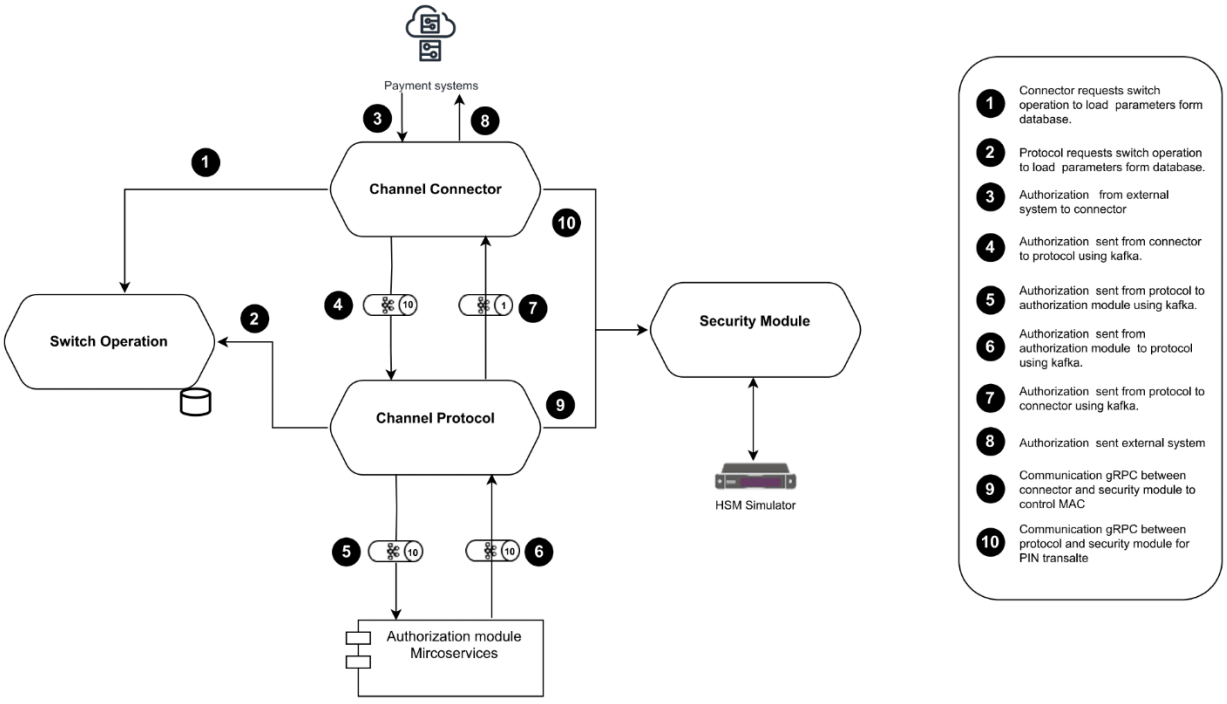


Figure 31 : Microservices communication schema

This diagram presents the general architecture of the authorization flow within the Switch PowerCARD system. When an authorization request is initiated (for example, from the Mastercard network), it is first received by the Channel Connector, which interacts with the Switch Operation module to load the necessary configuration parameters. The authorization message is then published to a Kafka topic and consumed by the Channel

Protocol, which in turn forwards the request to the authorization microservices. These microservices apply business logic (fund validation, scoring, security rules, etc.) and return their response to the Channel Protocol. The latter then constructs the response message and publishes it again to Kafka so that the Channel Connector can retrieve it and send it back to the external network. At the same time, the Channel Protocol communicates with the security module (HSM Simulator) via gRPC calls for sensitive processing: MAC generation and validation, PIN data management. This decoupling by Kafka ensures high availability and scalability of processing, while the use of gRPC for the security layer guarantees efficient and secure exchanges.

1.2.9 Channel connect diagram

Abstract diagram:

The diagram above represents the abstract architecture of the Channel Connector, a key component responsible for providing the interface between the internal protocol and the Mastercard network. This component also relies on a common SDK, providing the building blocks necessary to implement a standardized and reusable network connector.

The processing is divided into Kafka pipelines and abstract transformation functions:

- PwcChannelInboundConsumerSource receives inbound messages from the network (via a TCP/SSL interface) and injects them into the pipeline.
- The Abstract inbound processor function applies protocol-specific processing (decoding, format verification, etc.).
- Messages are then published to the protocol via PwcChannelProtocolSink on the pwc-net-in topic.

In reverse:

- Protocol output messages (pwc-net-out) are consumed by PwcChannelProtocolSource.
- The Abstract protocol processor function prepares messages to be sent over the network.
- PwcChannelOutboundSink provides transmission over the network interface (TCP/SSL).

The connector also integrates:

- PwcChannelHeartbeatSink for real-time monitoring.
- PwcChannelMetricSink for collecting technical metrics (published on the pwc-metric topic).
- PwcChannelEventLogSink for event traceability, powering an OpenSearch (Elastic) engine.

This decoupled, Kafka-based architecture enables asynchronous, scalable, and robust processing of Mastercard flows, while ensuring isolation between protocol logic and network transport.

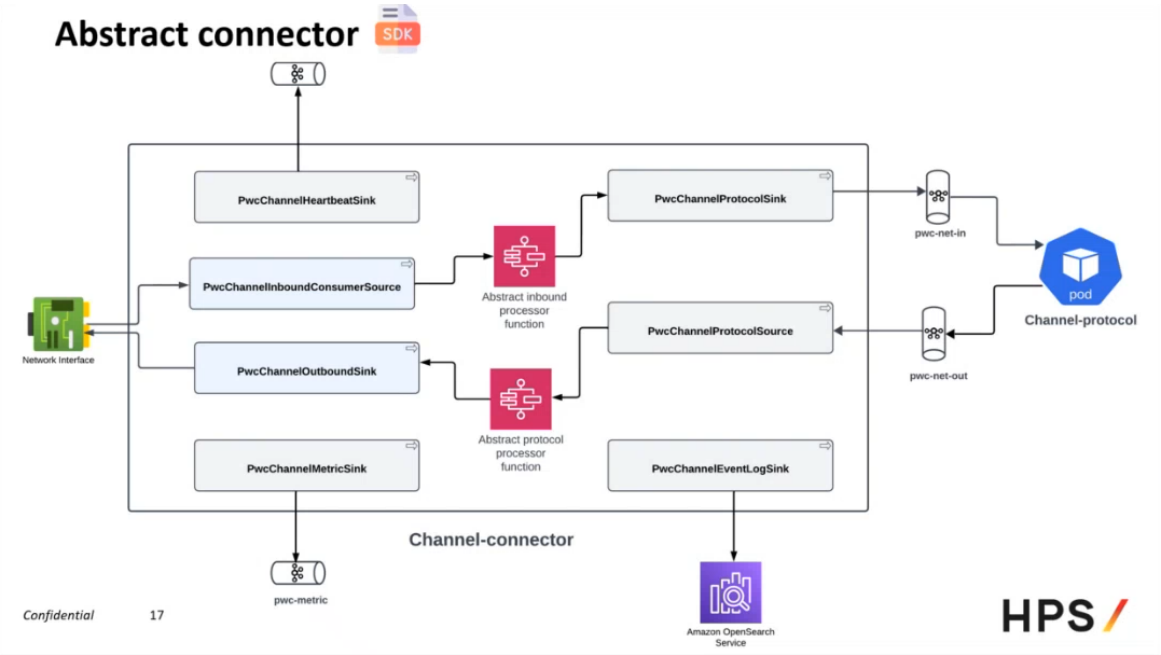


Figure 32 : Abstract connector schema

Incoming request:

This flow describes the processing of an incoming request (e.g., an authorization request) within the Channel Connector microservice.

The authorization message is sent by the Mastercard network and captured by the Channel Connector via a TCP connection. The message is then temporarily stored in an internal memory queue.

A free thread, from the thread pool dedicated to processing authorizations, takes charge of the message and performs format and integrity checks.

If a MAC (Message Authentication Code) check is enabled, the message is passed to the Security Module via gRPC call for verification.

After verification, the message is published to the Kafka topic pwc-net-in, using a partition key derived from authorization elements, to ensure correct ordering. This topic has multiple partitions, and partition selection depends on the message key.

The message is then consumed by the Channel Protocol microservice for further processing.

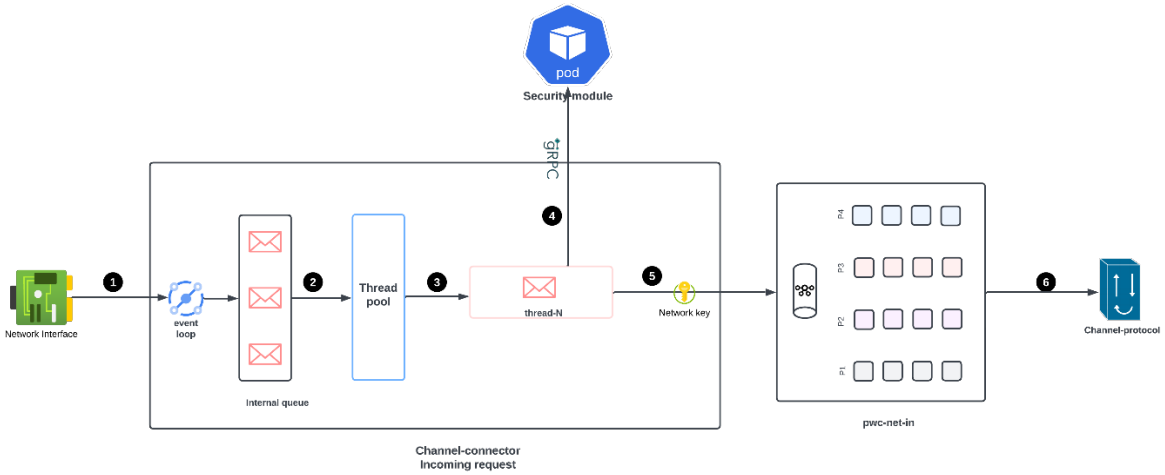


Figure 33 : Incoming request explicative schema

Outgoing response:

This flow describes the processing of an outgoing response within the Channel Connector (response to a previously issued request).

The authorization response message is sent by the Channel Protocol microservice on the pwc-net-out (multi-partition) Kafka topic.

The Channel Connector consumes this message on its dedicated partition. After consumption, the message is sent to the Mastercard network via the TCP connection.

Finally, the connector publishes technical metrics on the observed response time, by sending a message on the pwc-metric topic.

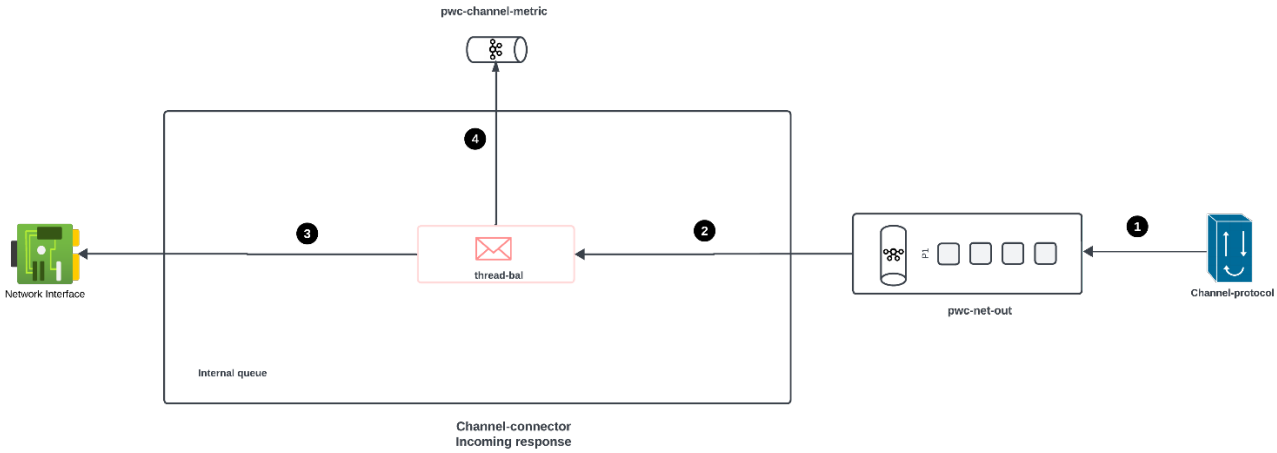


Figure 34 : Outgoing response explicative schema

Outgoing request:

This flow describes the processing of an outgoing request (request generated on the Switch side to Mastercard).

The authorization message is published by the Channel Protocol on the pwc-net-out (multi-partition) Kafka topic. The Channel Connector consumes this message on a specific partition.

The authorization context is saved in a local RocksDB database (to allow subsequent matches).

The message is then sent to the Mastercard network via the TCP connection.

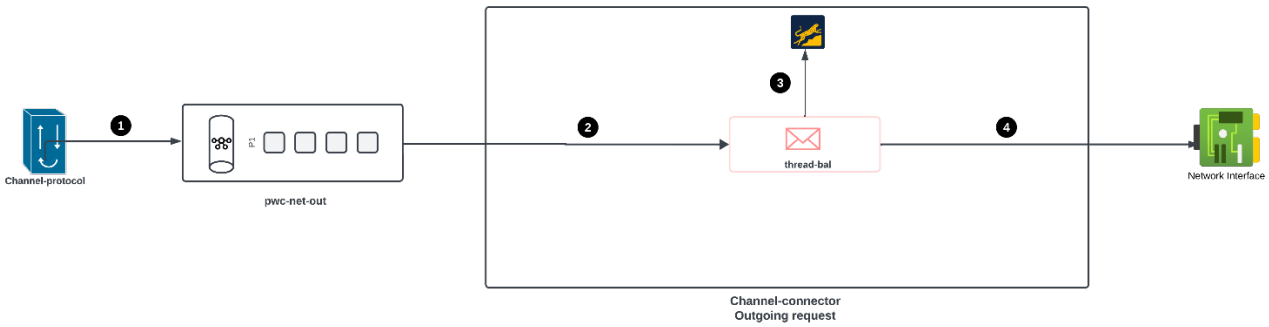


Figure 35 : Outgoing request explicative schema

Incoming response:

This flow describes the processing of an incoming response (from Mastercard) as part of an outgoing transaction (initiated by the Switch).

The message is captured by the Channel Connector via TCP and then temporarily stored in a memory queue.

A free thread from the processing pool takes over this message and performs the format checks.

The Channel Connector then retrieves the corresponding authorization context from RocksDB.

If MAC checking is enabled, the message is sent to the security module via gRPC for verification.

Finally, the message is published to Kafka (pwc-net-in), with a partition key consistent with the original context. The Channel Protocol will consume this message to reconstruct the full authorization.

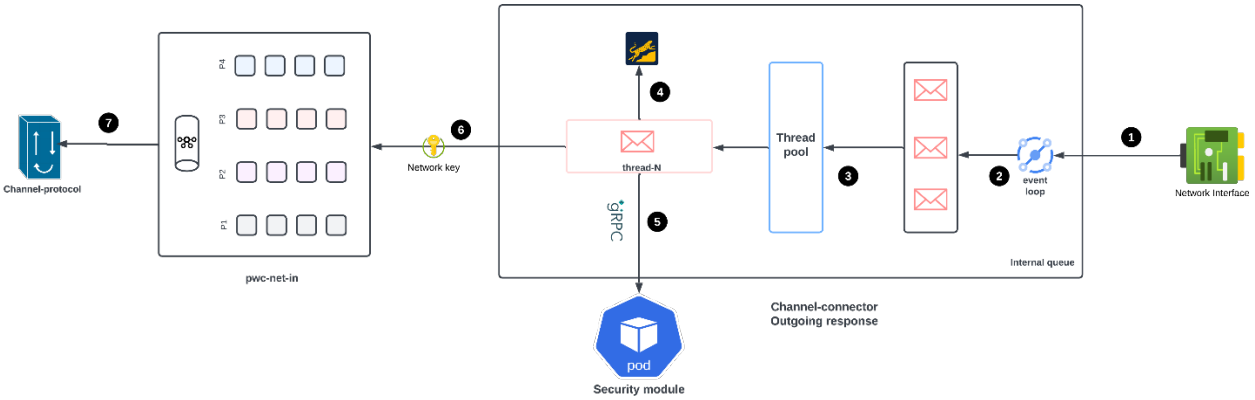


Figure : Incoming response explicative schema

1.2.10 Channel protocol diagram

Abstract diagram:

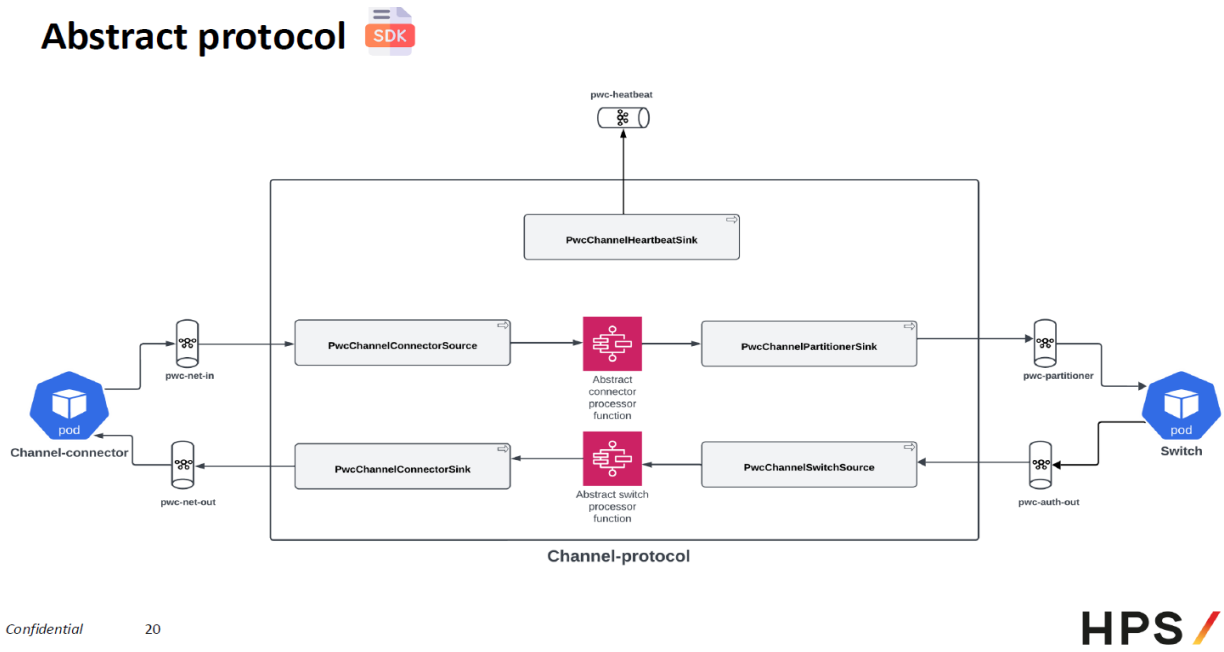


Figure 37 : protocol abstract schema

The abstract diagram above represents the logical architecture of the Channel Protocol microservice, which is based on an SDK common to all implemented protocols (CIS, ISO, SID, etc.). Its role is to ensure generic interfacing between the Channel Connector (to the Mastercard network) and the PowerCARD Switch (core business), using Kafka to guarantee scalability and resilience.

The different components are specialized processors, each consuming and publishing to dedicated Kafka topics:

- PwcChannelConnectorSource consumes incoming network messages (pwc-net-in topic), performs protocol processing (e.g. CIS → ISO conversion), then publishes to PwcChannelPartitionerSink or PwcChannelSwitchSource depending on the business logic.
- PwcChannelPartitionerSink publishes to the partitioned topic pwc-partitioner, which is consumed by the Switch for synchronous processing (e.g. authorization requests).
- PwcChannelSwitchSource consumes business responses from the Switch (pwc-auth-out topic) and forwards them to the Connector via PwcChannelConnectorSink.
- PwcChannelHeartbeatSink publishes heartbeat messages to monitor protocol health.

The SDK also provides abstract processing functions (Abstract connector processor / Abstract switch processor), on which each protocol implements its own business rules (e.g. CIS □ ISO field mapping).

This modular approach makes it easy to implement new protocols, while respecting a robust and validated common base.

Incoming request

The diagram below describes the processing flow of an incoming authorization message within the Channel Protocol microservice.

The authorization message is sent to the Protocol microservice via the pwc-net-in Kafka topic.

The Channel Protocol uses a Kafka consumer pool to read messages from this topic.

A consumer thread retrieves the message and performs the protocol conversion (CIS → ISO).

The original message is stored in a RocksDB database (allowing correlation with the future response).

The message is then published to the Switch microservices via the partitioned topic pwc-online-message-partitioner, the partition key being based on business elements of the transaction (card number, STAN, etc.).

Switch microservices then consume the message for business processing.

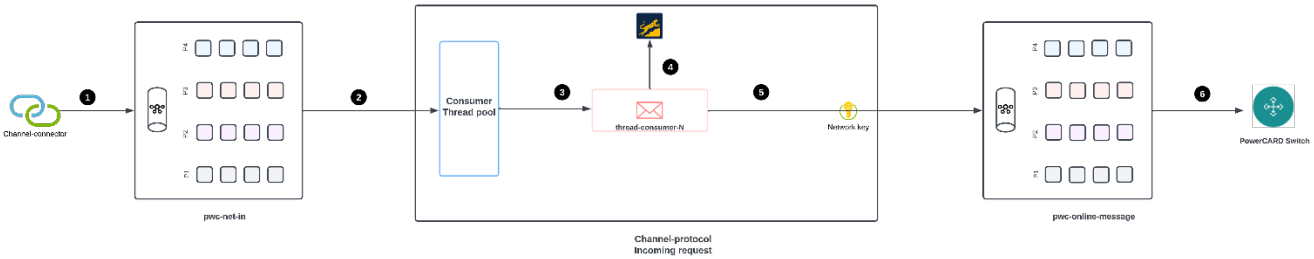


Figure 38 : incoming request schema

Incoming response

The following diagram describes the processing flow of an incoming authorization response message within the Channel Protocol.

The response message is emitted by the Switch microservices on the Kafka topic pwc-auth-out.

The Channel Protocol uses a pool of consumers to read messages on this partitioned topic.

A consumer thread retrieves the message and performs the protocol conversion.

The protocol retrieves the initial authorization request stored in RocksDB to ensure consistency of the response message.

The message is then sent to the Connector via the pwc-net-in topic, to be transmitted to the external network.

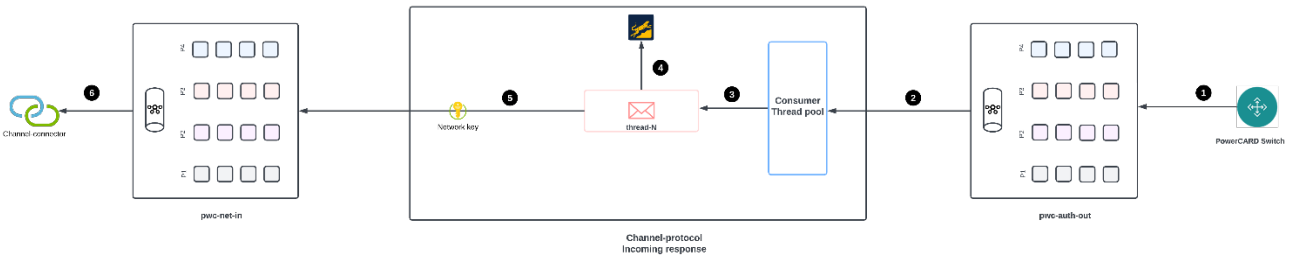


Figure 39 : incoming response schema

Outgoing request

This diagram illustrates the processing flow of an outgoing request to the external network.

The message is produced by the Switch microservices on the Kafka topic pwc-auth-out.

The Channel Protocol consumes this message and performs the protocol conversion.

If the request contains a PIN, the Channel Protocol calls the HSM module (via gRPC) to perform the PIN translation.

The original message is stored in RocksDB.

The message is then published to the pwc-net-in topic to be transmitted to the Connector, which relays it to the external network.

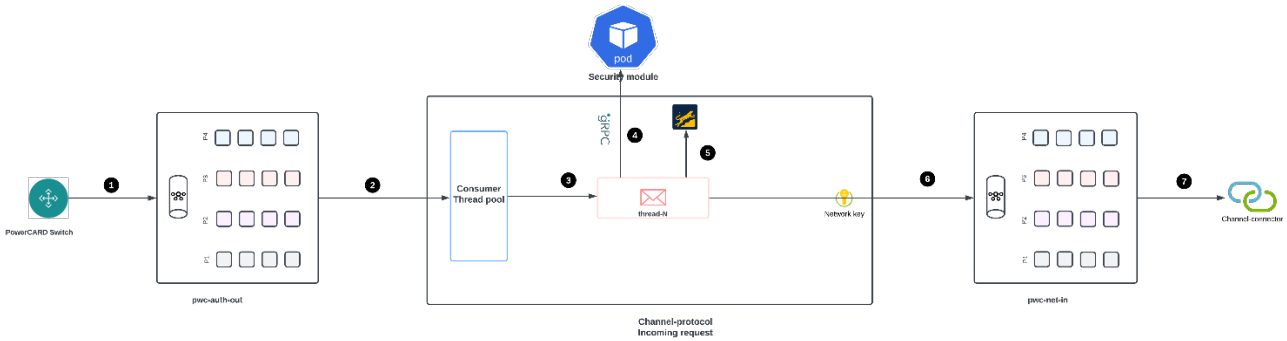


Figure 40 : outgoing request schema

Outgoing response

Finally, the diagram below describes the processing of an outgoing response.

The message is sent to the Channel Protocol via the Kafka topic pwc-net-in.

The Channel Protocol consumes this message and performs the protocol conversion.

It retrieves the original request from the RocksDB database, prepares the response and publishes it to the Switch microservices via the partitioned topic pwc-online-message-partitioner.

Switch microservices consume the message and continue business processing.

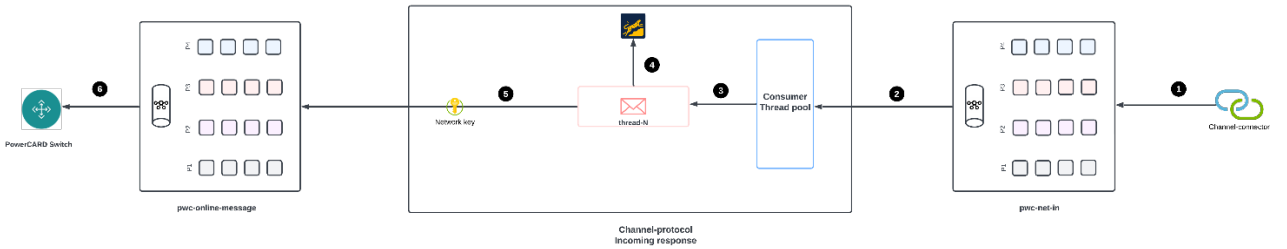


Figure 41 : outgoing response schema

1.2.11 Conclusion :

This chapter established the functional and conceptual foundations of the Mastercard CIS protocol migration solution to the PowerCARD Switch V4 architecture.

Through the analysis of requirements, the definition of use cases, and the modeling of flows (sequence and activity diagrams), we clarified the role of the key components Channel Protocol and Channel Connector as well as their interactions with the various stakeholders in the payments ecosystem: Mastercard network, the Switch, issuing and acquiring banks.
This work highlighted the inherent complexity of migrating a critical payment protocol within a high-performance, real-time, and highly secure transactional environment. The need to process heterogeneous flows (acquiring/issuing, requests/responses, Authorization , Advice, Reversals, File Updates) calls for a modular, scalable, and robust architecture precisely what a microservices approach enables.
The different models produced (UML diagrams, message flows, context management) also helped anticipate certain technical challenges related to the migration, particularly the fine-grained handling of authorization contexts, end-to-end message traceability, and smooth integration with the legacy components of the Switch.
This analysis and design phase thus served as a key step to ensure a structured, coherent, and standards-compliant implementation one that meets the high demands of the banking and payments industry.

Chapter 4: Implementing the Solution

1.3.1 Introduction :

This chapter presents the concrete technical implementation of the microservices developed during this project: Channel Protocol and Channel Connector.
The work also includes the development of the CIS SDK used by the microservices, as well as the setting up of tests and validation environments.
The project was carried out following the standards and practices of the Online Product team, which provides a microservice skeleton generator called Channel Initializer, included in the pwc-switch-v4-test-tools project.

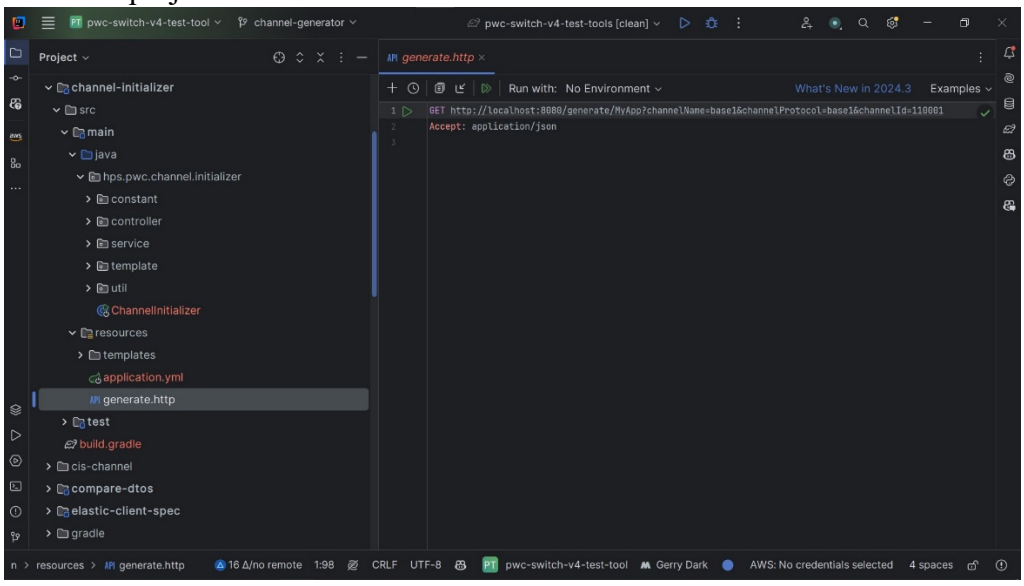


Figure 42 : screenshot of the tool channel initializer
This generator provides the basic architecture common to all channels developed around the Switch V4, and guarantees component consistency.

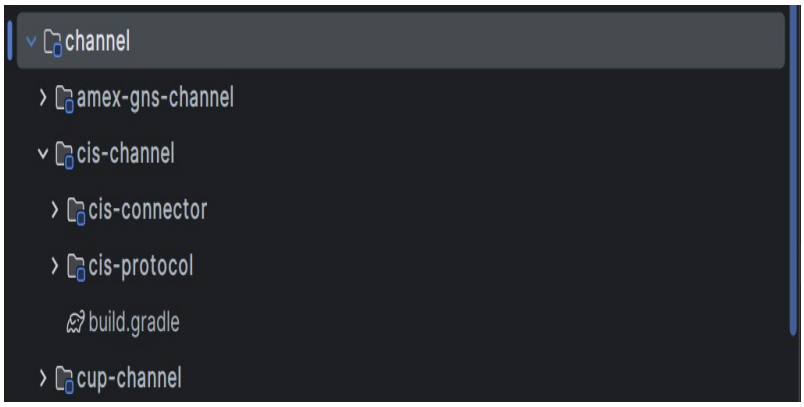


Figure 43 : : screenshot of the abstract skeleton of a channel

1.3.2 Connector microservice implementation:

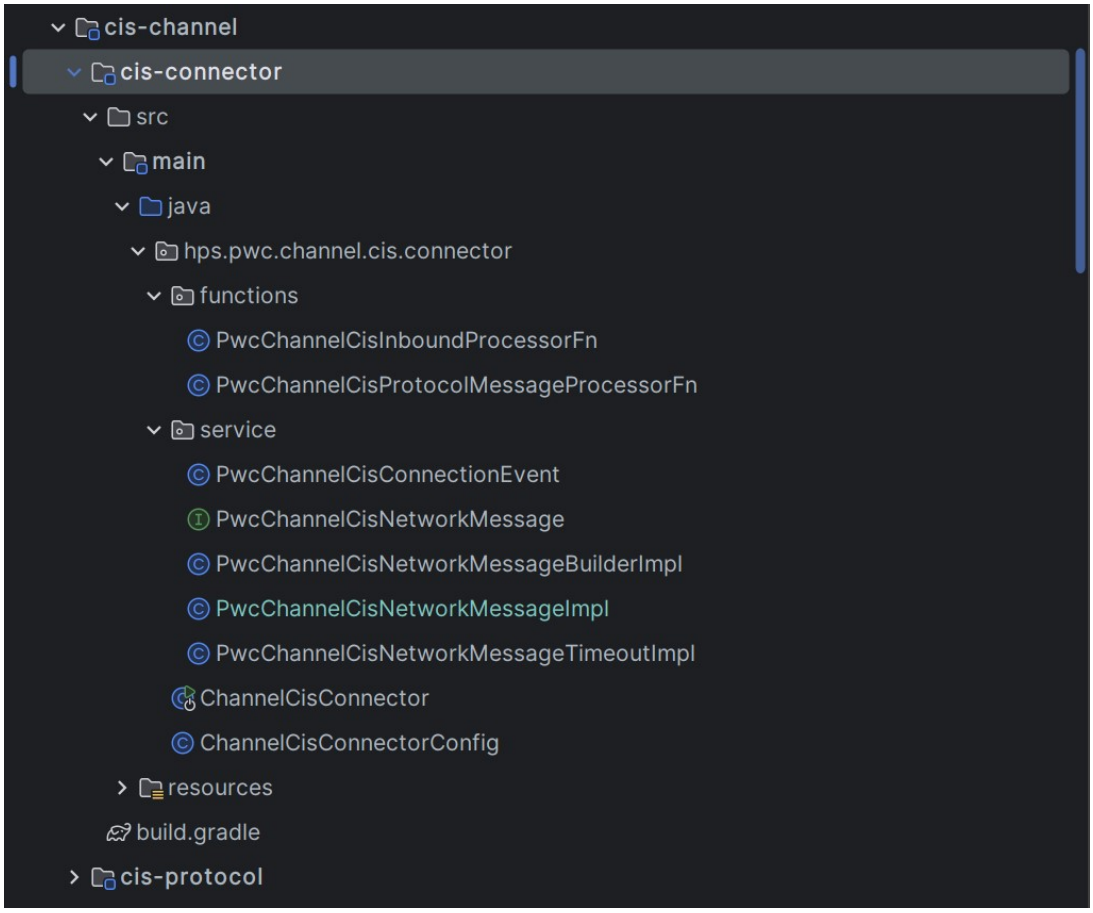


Figure 44 : screenshot of the connector microservice structure

The Channel Connector microservice relies on several internal dependencies: a channel core, an abstract SDK for the protocol, a library for managing TCP/IP communications, advanced logging tools, and an SDK for managing Kafka streams.

The development work consisted of migrating and reimplementing in Java all the network logic from the legacy C code.

This includes in particular:

- Management of Network Management type messages (0800 and 0820 session management messages), allowing connection/disconnection operations and network status verification to be processed.

- Processing of cryptographic key exchanges with the network, necessary for the proper establishment of secure sessions.
- The implementation of a robust initial connection mechanism, incorporating several automatic attempts in the event of failure, in order to ensure maximum availability of services.
- Implementation of network message sending functions (connect, disconnect, echo test), adapting specific cases from the C standard to consistent processing in Java.

Thanks to this work, all processing related to network management was able to be migrated and stabilized in the Connector microservice, respecting Java best practices and the overall architecture of the project.

1.3.3 Implementation of the Protocol microservice

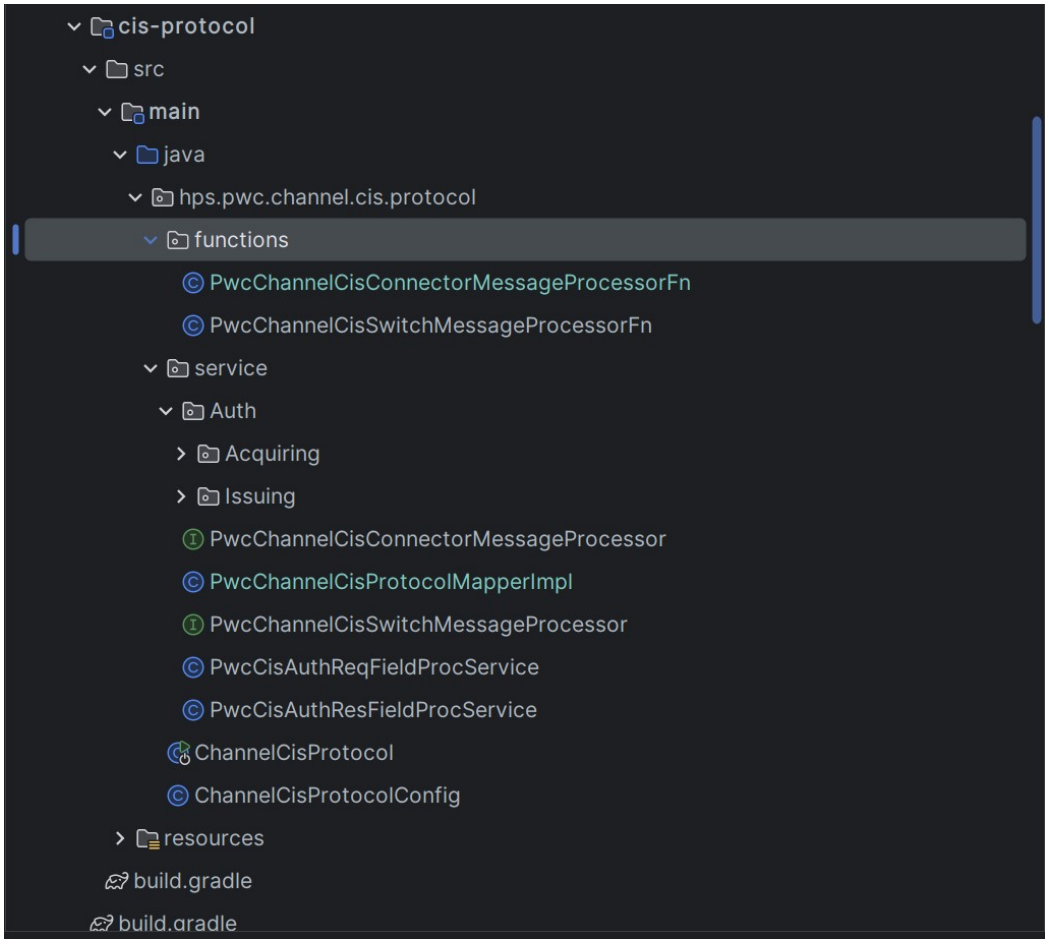


Figure 45 : screenshot of the protocol microservice structure

The Channel Protocol microservice is built on an abstract common SDK that provides the technical foundations needed to implement the protocol.

The adopted software architecture is based on a modular model allowing the processing flows between acquiring and issuing transactions to be clearly separated.

For each type of message whether it's an authorization request, a reversal, an advice, or a file update dedicated components are responsible for orchestrating the processing.

These components rely on mappers and utilities to perform message conversion between ISO 8583 and CIS formats, in both directions.

The core of the business logic comes from the migration of several source files from legacy C code (management of advices, file updates, network communication, reversals and authorizations). This logic was translated into Java while respecting the framework constraints and integrating modern best development practices.
The CIS SDK, developed in parallel, provides all the constants, enumerations, and utilities necessary for the consistency and reliability of the processing carried out in the Protocol.

1.3.4 CIS SDK Development

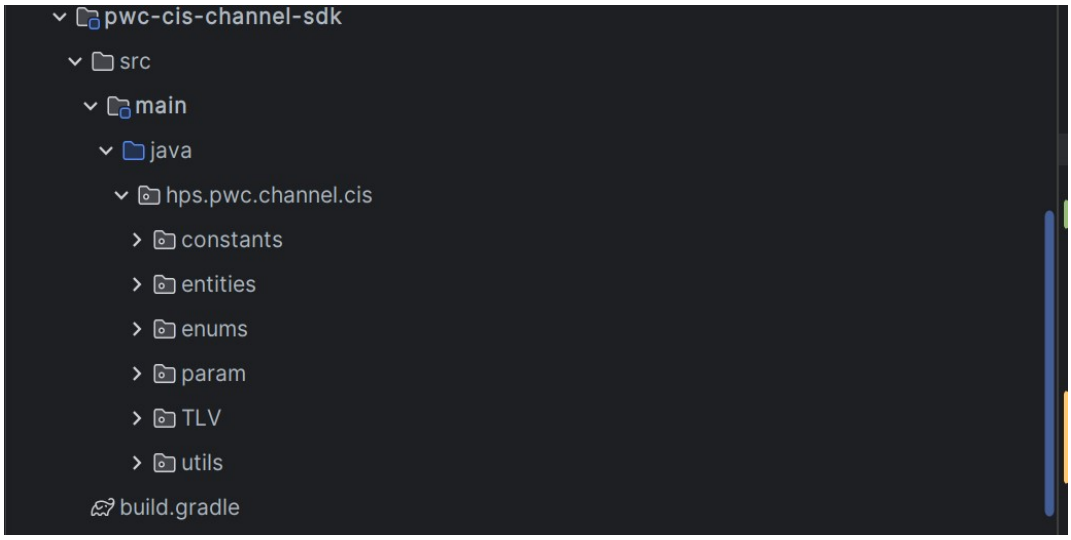


Figure 46 : screenshot of the CIS SDK structure
The SDK allows you to factor out all the constants, enumerations, utilities, and parameters needed by the Protocol and Connector microservices.
The SDK contains:

- All CIS enumerations
- Functions for tlv processing not handled by the abstract SDK or other SDK
- CisFieldUtils: file that manages all repetitive processing in the project fields
- PwcCisMessageFormatBuilder: this file creates a map for all the fields to build the incoming and outgoing Cis message, this file processes it by obligatoi
- Format/Size Constants
- Parameterized management entities (loading tables)
- Functional utilities:
- buildCisMessageHeader
- getMcEpiArea
- getCountryAlphaCode
- isoToCisAdditionalAmounts
- MciToIsoMdesMemberDefData

This SDK ensures consistency between treatments and simplifies future evolution.

1.3.5 Integration tests of migrated microservices:

To prepare the integration test environment, we used Docker Desktop to orchestrate and monitor the various necessary containers, such as Kafka, Zookeeper, and infrastructure services. the Docker Desktop interface allows you to view the status of the containers, check their resource consumption, and easily access exposed ports, facilitating rapid deployment of the environment.

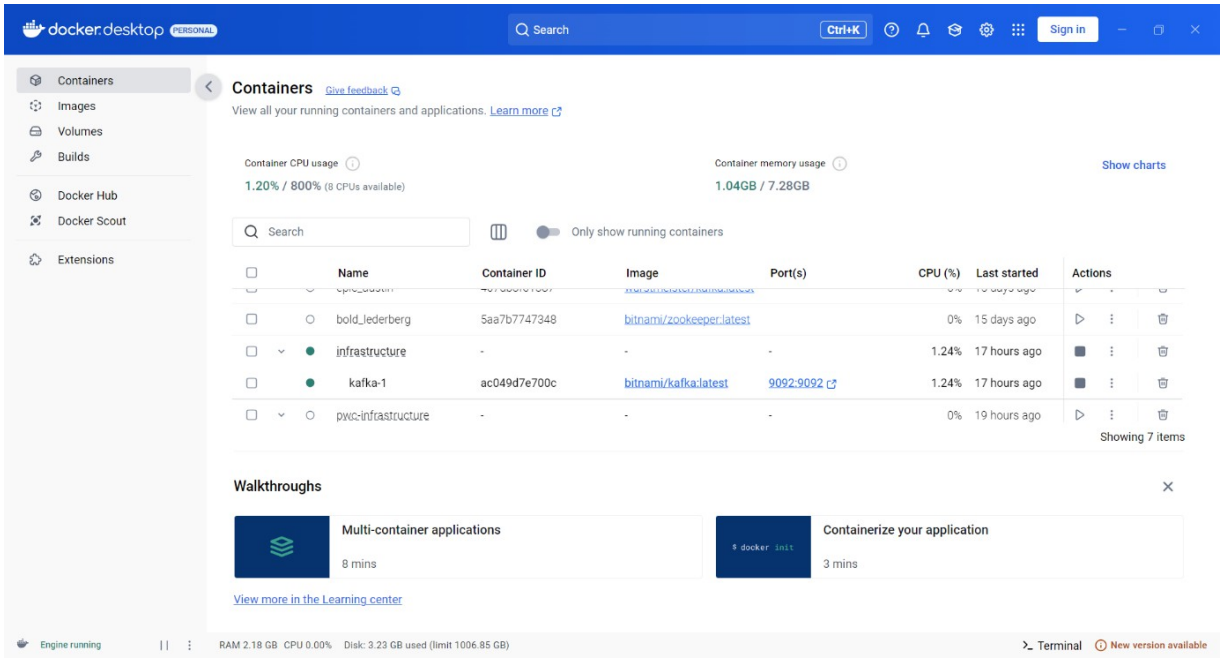


Figure 47 : screenshot of the Docker containers

For the OpenLens setup, we added the clusters as shown in the following screenshot . Once connected to the cluster via double-click, we selected the Services tab under Config, then searched for and clicked on the postgresql service The exposed port (5432) was then forwarded to the application.yaml file to ensure connectivity with the database.

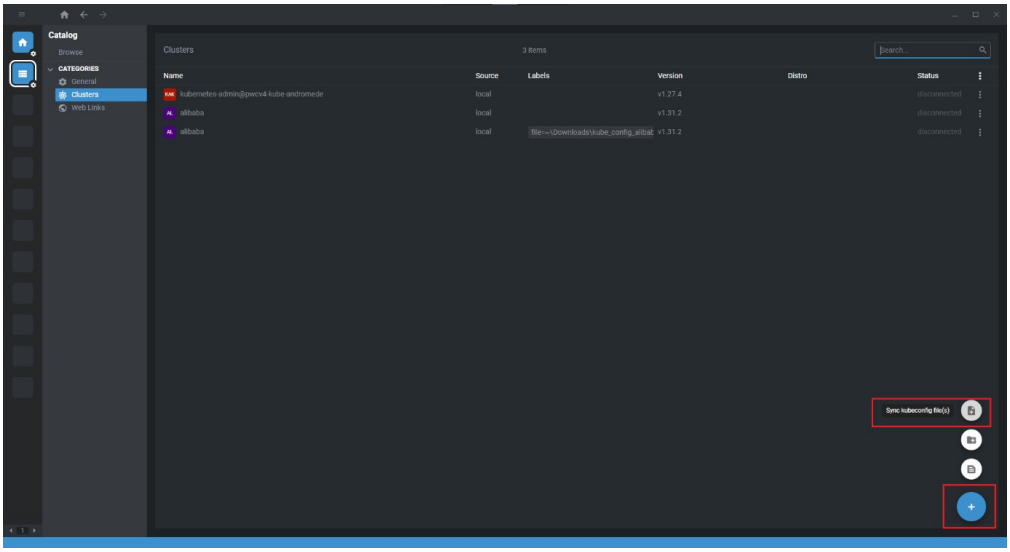


Figure : screenshot of Openlens : clusters

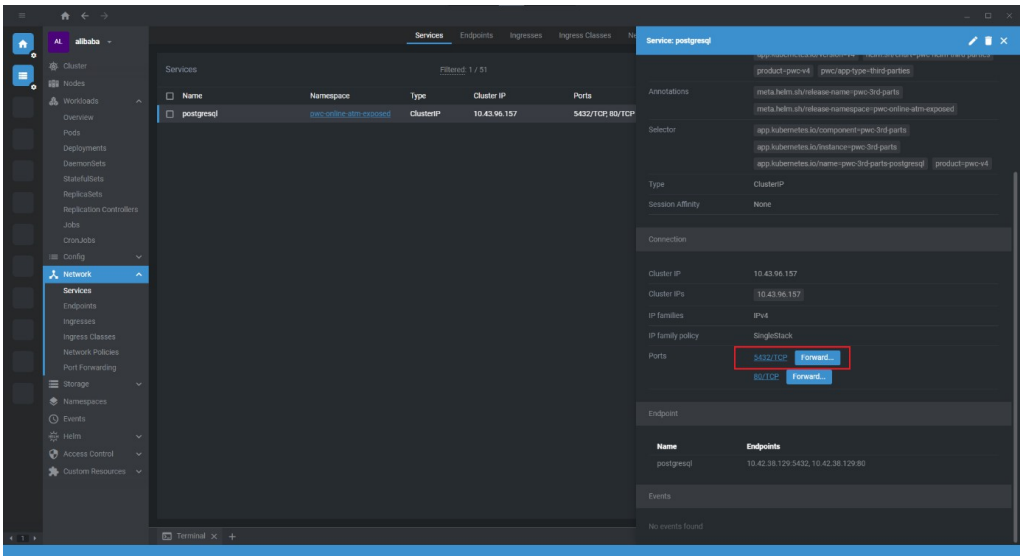


Figure 49 : screenshot of Openlens : forwarding the DB postgresql

We then launched the KafkaComposeUp task under Docker tasks :

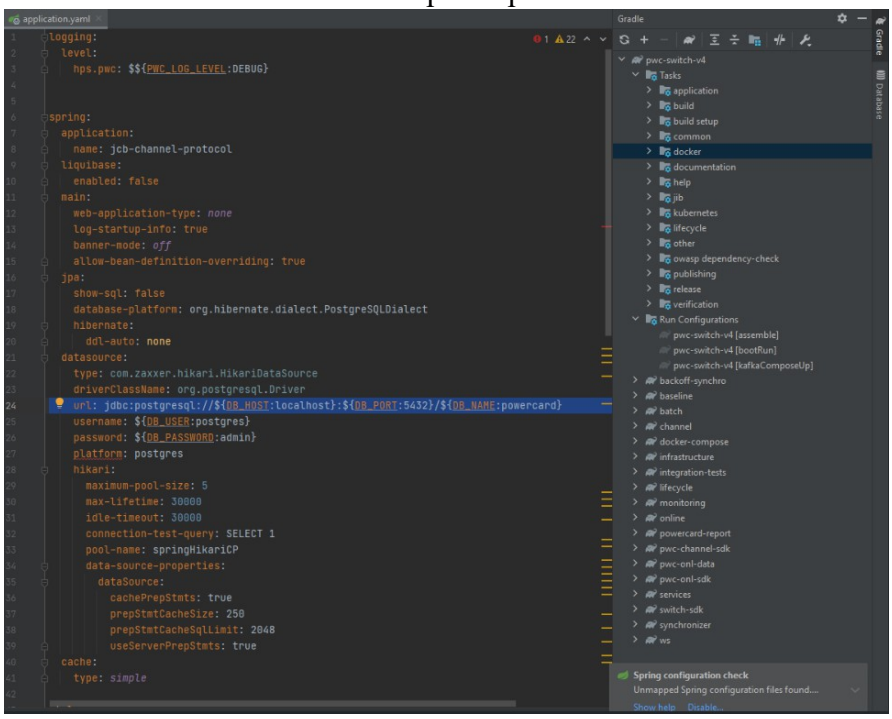


Figure 50 : screenshot of xml config to read the forwarded config

Once the service is started, the Kafka environment is

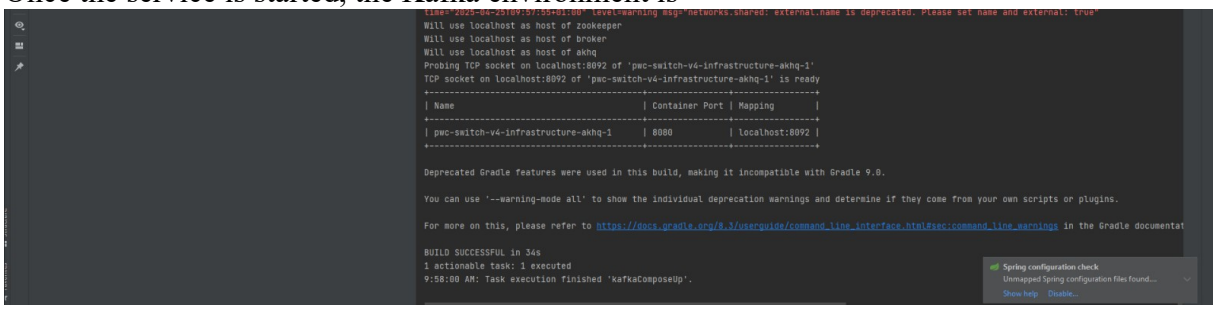


Figure 51 : screenshot of out of kafka compose up

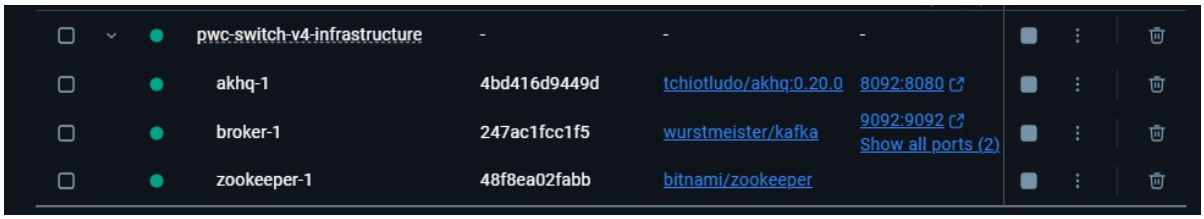


Figure 52 : screenshot of images up in docker

After setting up Kafka, we started the CIS Channel connector :

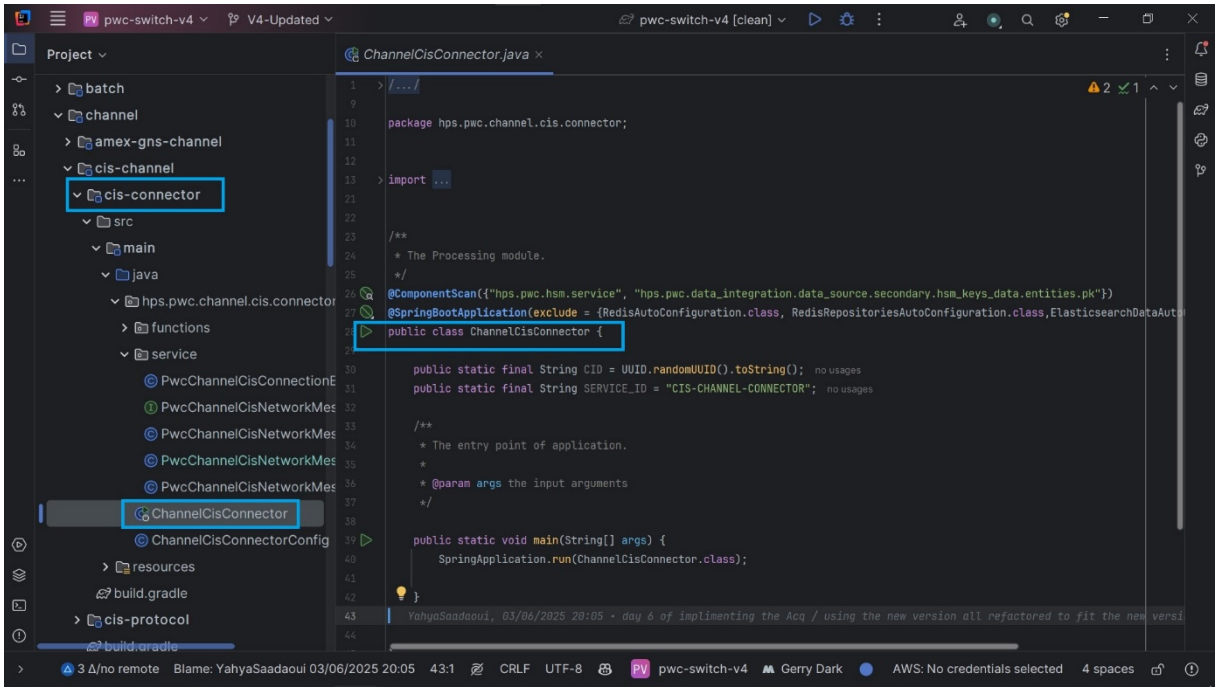


Figure 53 : screenshot of how to lunch the connector

Once fired, the onOpenConnection event initiates sending a signon message to the PSTT simulator, which in turn responds.

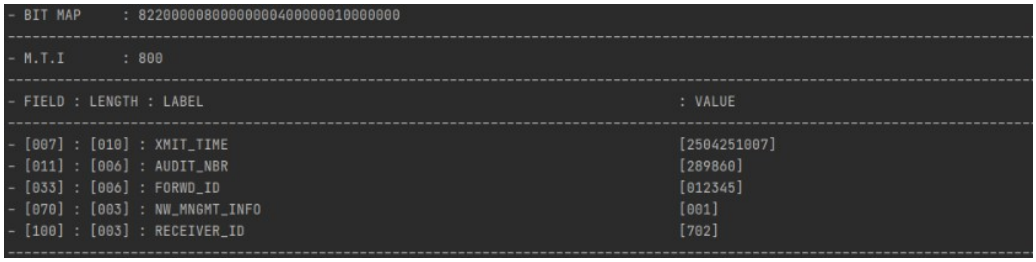


Figure 54 : screenshot of the results Sending sign on message

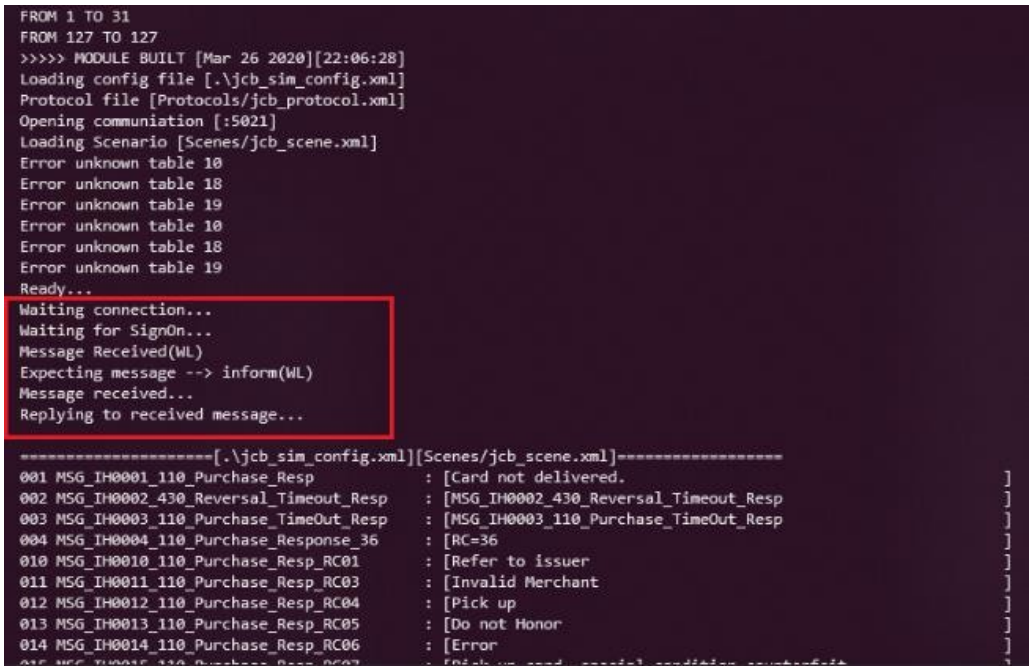


Figure 55 : screenshot of the results PSTT response

We then started the protocol microservice and the switch simulator:

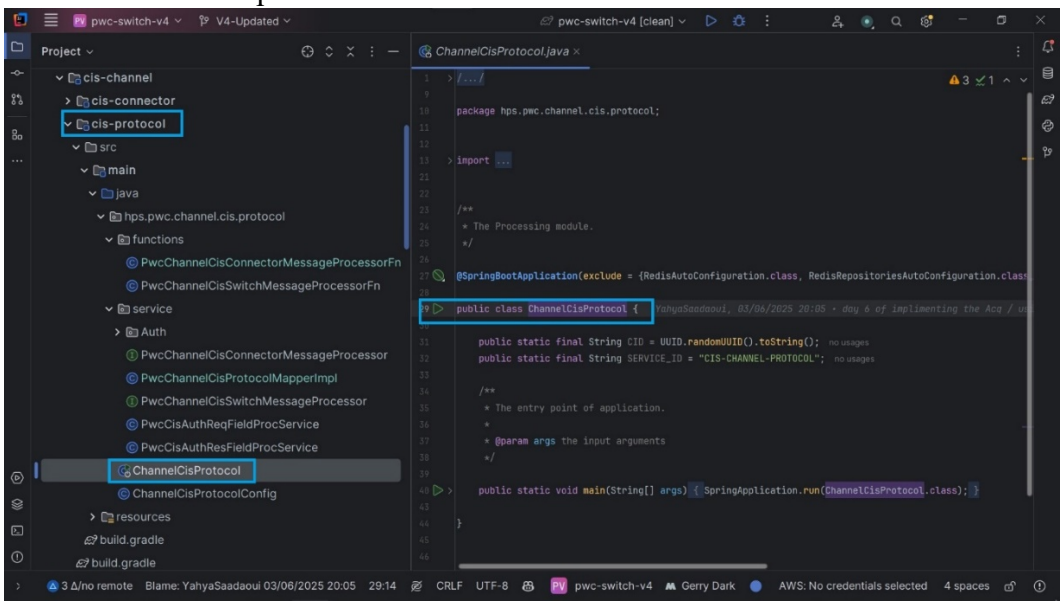


Figure 56 : screenshot of how to lunch protocol

At this point, it is possible to send an authorization message from the PSTT simulator and observe the traces generated at each step:

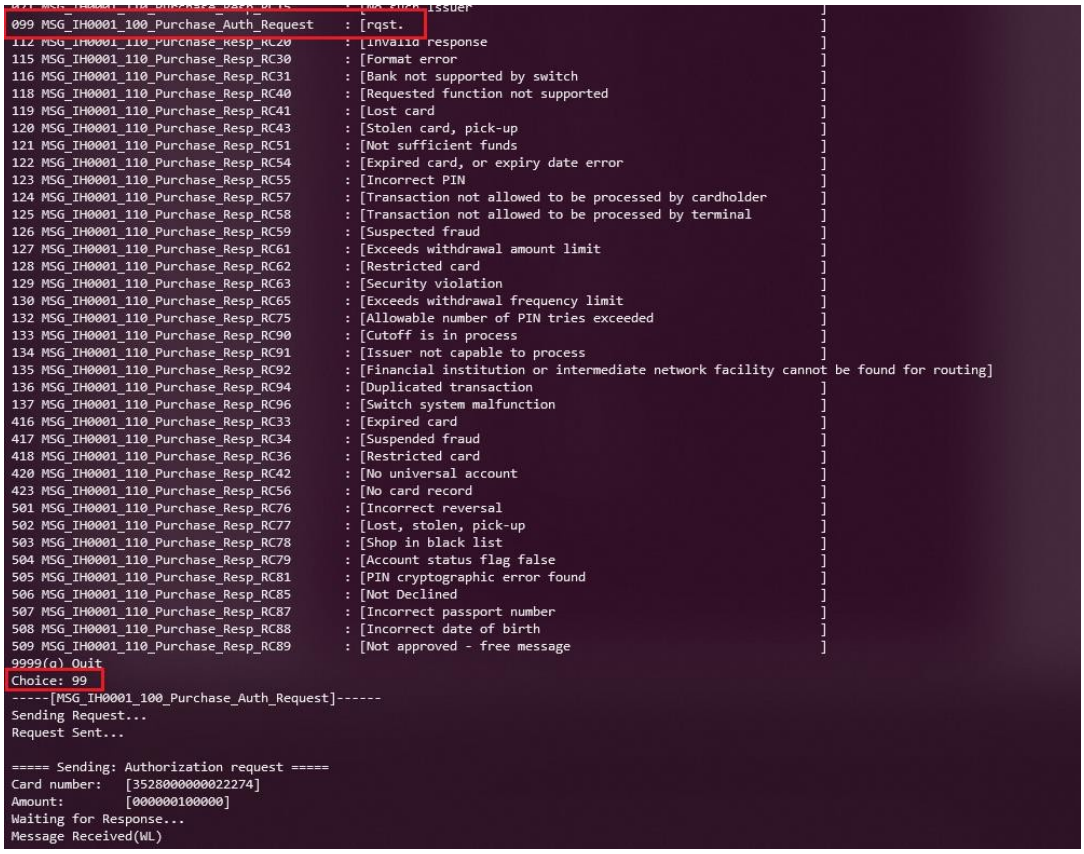


Figure 57 : screenshot of results when Sending the message from the PSTT

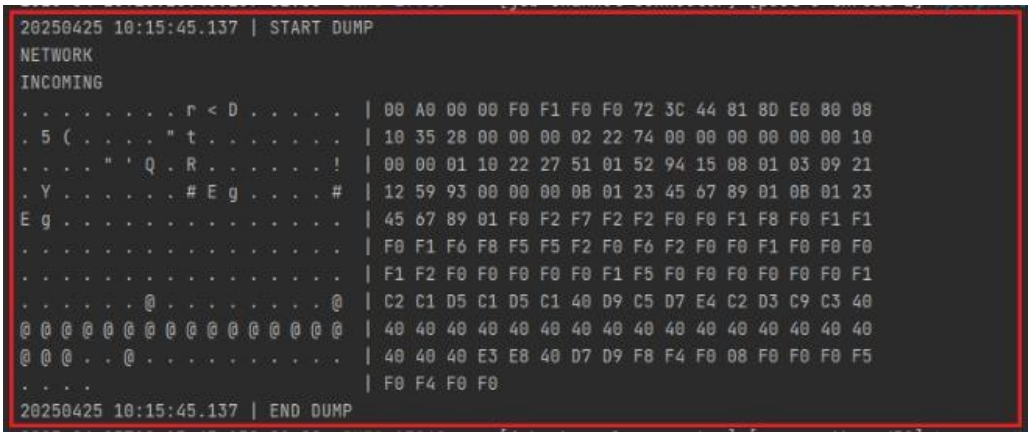


Figure 58 : screenshot of results Receiving the incoming message in the connector

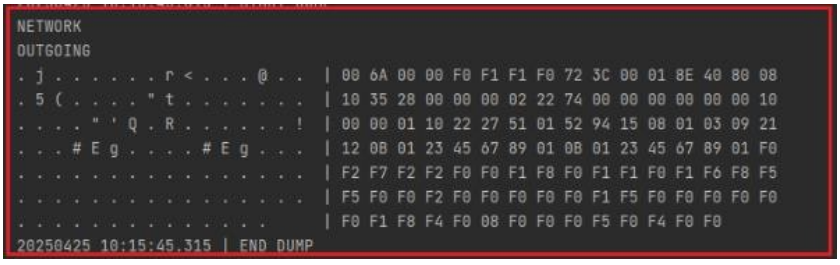
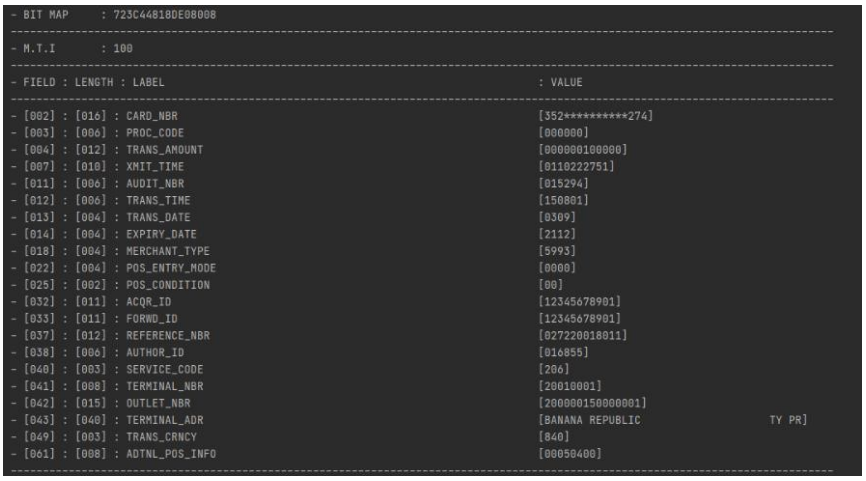
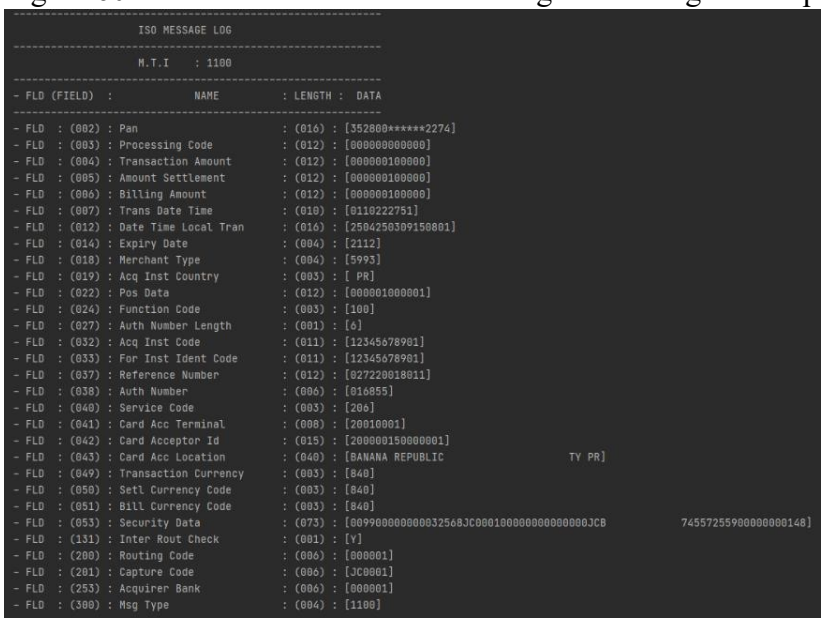


Figure 59 : screenshot of results Sending the connector response to the PSTT



```
- BIT MAP     : 723C44818DE08008
-------------------------------------------------------------------------------------------------------------------------------
- M.T.I       : 100
-------------------------------------------------------------------------------------------------------------------------------
- FIELD : LENGTH : LABEL                                                    : VALUE
-------------------------------------------------------------------------------------------------------------------------------
- [002] : [016] : CARD_NBR                                                 [352**********274]
- [003] : [006] : PROC_CODE                                                [000000]
- [004] : [012] : TRANS_AMOUNT                                             [000000100000]
- [007] : [010] : XMIT_TIME                                                [0110222751]
- [011] : [006] : AUDIT_NBR                                                [015294]
- [012] : [006] : TRANS_TIME                                               [150801]
- [013] : [004] : TRANS_DATE                                               [0309]
- [014] : [004] : EXPIRY_DATE                                              [2112]
- [018] : [004] : MERCHANT_TYPE                                            [5993]
- [022] : [004] : POS_ENTRY_MODE                                           [0000]
- [025] : [002] : POS_CONDITION                                            [00]
- [032] : [011] : ACQR_ID                                                  [12345678901]
- [033] : [011] : FORWD_ID                                                 [12345678901]
- [037] : [012] : REFERENCE_NBR                                            [027220018011]
- [038] : [006] : AUTHOR_ID                                                [016855]
- [040] : [003] : SERVICE_CODE                                             [206]
- [041] : [008] : TERMINAL_NBR                                             [20010001]
- [042] : [015] : OUTLET_NBR                                               [200000150000001]
- [043] : [040] : TERMINAL_ADR                                             [BANANA REPUBLIC                    TY PR]
- [049] : [003] : TRANS_CRNCY                                              [840]
- [061] : [008] : ADTNL_POS_INFO                                           [00050400]
-------------------------------------------------------------------------------------------------------------------------------
```

Figure 60 : screenshot of results Receiving the message in the protocol



```
-------------------------------------------------------------
                  ISO MESSAGE LOG
-------------------------------------------------------------
                  M.T.I    : 1100
-------------------------------------------------------------
- FLD (FIELD)  :            NAME          : LENGTH :  DATA
-------------------------------------------------------------
- FLD  : (002) : Pan                      : (016) : [352800******2274]
- FLD  : (003) : Processing Code          : (012) : [000000000000]
- FLD  : (004) : Transaction Amount       : (012) : [000000100000]
- FLD  : (005) : Amount Settlement        : (012) : [000000100000]
- FLD  : (006) : Billing Amount           : (012) : [000000100000]
- FLD  : (007) : Trans Date Time          : (010) : [0110222751]
- FLD  : (012) : Date Time Local Tran     : (016) : [2504250309150801]
- FLD  : (014) : Expiry Date              : (004) : [2112]
- FLD  : (018) : Merchant Type            : (004) : [5993]
- FLD  : (019) : Acq Inst Country         : (003) : [ PR]
- FLD  : (022) : Pos Data                 : (012) : [000001000001]
- FLD  : (024) : Function Code            : (003) : [100]
- FLD  : (027) : Auth Number Length       : (001) : [6]
- FLD  : (032) : Acq Inst Code            : (011) : [12345678901]
- FLD  : (033) : For Inst Ident Code      : (011) : [12345678901]
- FLD  : (037) : Reference Number         : (012) : [027220018011]
- FLD  : (038) : Auth Number              : (006) : [016855]
- FLD  : (040) : Service Code             : (003) : [206]
- FLD  : (041) : Card Acc Terminal        : (008) : [20010001]
- FLD  : (042) : Card Acceptor Id         : (015) : [200000150000001]
- FLD  : (043) : Card Acc Location        : (040) : [BANANA REPUBLIC                  TY PR]
- FLD  : (049) : Transaction Currency     : (003) : [840]
- FLD  : (050) : Setl Currency Code       : (003) : [840]
- FLD  : (051) : Bill Currency Code       : (003) : [840]
- FLD  : (053) : Security Data            : (073) : [0099000000000032568JC0001000000000000000JCB          7455725590000000148]
- FLD  : (131) : Inter Rout Check         : (001) : [Y]
- FLD  : (200) : Routing Code             : (006) : [000001]
- FLD  : (201) : Capture Code             : (006) : [JC0001]
- FLD  : (253) : Acquirer Bank            : (006) : [000001]
- FLD  : (300) : Msg Type                 : (004) : [1100]
```

Figure 61 : screenshot of results Transformation of the message to the ISO Switch protocol

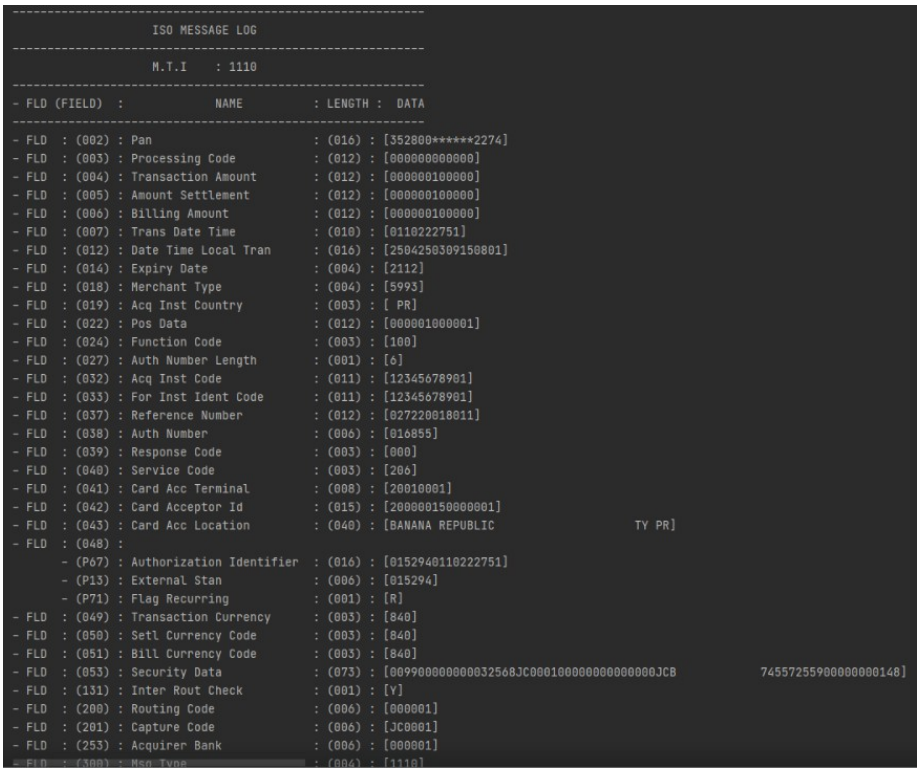


Figure 62 : screenshot of results Receiving the response from the switch simulator

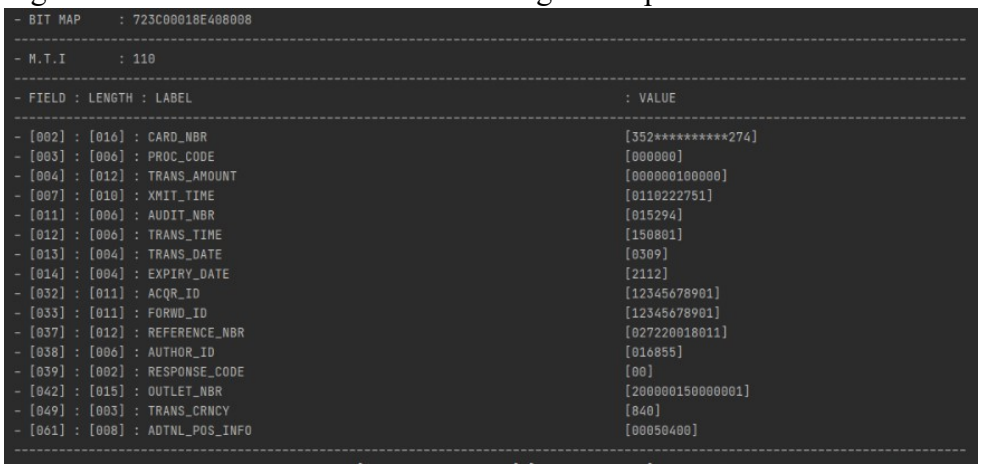


Figure 63 : screenshot of results Transformation of the response to the external protocol

This sequence illustrates the complete journey of an authorization message, from initiation to response, passing through all key components of the migrated architecture

1.3.6 Conclusion :

The implementation work carried out during this project made it possible to thoroughly modernize the management of the CIS protocol by migrating most of the logic inherited from the C code to a microservices-oriented Java architecture. Thanks to this approach, the processing is now more modular, maintainable, and aligned with the technical standards of the Switch V4 ecosystem.

The microservices developed (Channel Protocol and Channel Connector), as well as the associated CIS SDK, now offer a robust and consistent basis for processing acquiring

and issuing flows, ensuring compatibility with Mastercard specifications and production requirements.

The project allowed for the full use of modern technologies such as Kafka for asynchronous exchanges, gRPC for secure communication with the HSM module, and RocksDB for managing transactional contexts.

Among the most complex technical aspects, we can cite the fine migration of the Reversals logic, the management of Advice (specific business cases) and the porting of network management to TCP/IP.

This work also enabled significant skill development on the following themes:

- Distributed microservices architecture
- Asynchronous processing and Kafka stream management
- Payment security (MAC, PIN translation via HSM)
- Software industrialization and compliance with good development practices

This technical foundation paves the way for the evolution and future extension of the capabilities of the CIS protocol within the Switch V4.

Chapter 5: Architecture and technologies

1.4.1 Introduction :

The objective of this chapter is to present the general software architecture of the developed solution, as well as the main technologies and tools used within the framework of the project.

The chosen architecture is based on a microservices approach, allowing for better separation of responsibilities, fine scalability of components, and easier maintenance. As the CIS protocol plays a critical role in real-time transaction processing, it was essential to guarantee optimal performance and strong system resilience.

The technological choices were guided by the standards in force in the Online Product team, while taking into account the specificities of the CIS protocol and the requirements of Switch V4.

In this chapter, we will detail:

- The overall architecture of the system
- The main technical building blocks used
- Development and industrialization tools
- The deployment and testing environment

1.4.2 Functional architecture

The Channel V4 architecture is based on several specialized layers:

- **Connector**: manages the network communication layer in TCP/IP, ensuring the reading/writing of messages to/from external networks (Mastercard, Visa, etc.).
- **Protocol**: manages the conversion of messages between the external format (CIS, ISO 8583, etc.) and the internal format of the Switch V4. It also performs message conformity checks.
- **Gateway**: allows handling of HTTP communications for other channel types (not used here for CIS).
- **Channel**: global orchestration layer that assembles the Protocol and Connector components.

This division ensures a clear separation of responsibilities and facilitates the maintenance and evolution of components.

1.4.3 Application architecture: from C monolith to Java microservices :

The CIS (Channel Interface System) protocol implementation has transitioned from a tightly coupled C-based monolith to a modular microservice architecture, meticulously designed to adhere to core software engineering principles.

Monolithic Legacy:

The original CIS implementation suffered from the classic limitations of a monolithic design:

- **Tight Coupling:** The C codebase exhibited high coupling between core functionalities, hindering maintainability and evolution. The specific areas of coupling were:
  - OSI communication with external components
  - **Network Communication Layer:** Handling the CIS protocol's network interactions was intertwined with other logic.
  - **Business Logic (ISO ↔ CIS Mapping):** The translation between ISO 8583 (or another financial standard) and the specific CIS protocol was not cleanly separated.
  - **Transaction Management:** Functions like acquiring, issuing, and reversal processing were tightly integrated.
- **Deployment Challenges:** Any update, regardless of scope, required a full recompilation and redeployment of the entire system.
- **Limited Scalability:** It was impossible to scale specific components independently to address performance bottlenecks.
- **Reduced Resilience:** A single failure point could bring down the entire CIS processing chain.

Microservice Transformation (Design-Centric):

To overcome these limitations, we migrated to a microservices architecture, emphasizing a clean separation of concerns and adherence to SOLID principles. This design emphasizes:

- **Decoupling and Modularity:** The CIS channel is now decomposed into smaller, independent services such as:
  - **Connector Service:** Handles network connectivity, protocol-specific interactions, and initial message processing. Think of this as the ”edge” of the CIS channel, responsible for interfacing with external CIS systems.
  - **Protocol Service:** Responsible for CIS-specific data validation, transformation, and mapping to internal domain objects. Implements the specific ISO ↔ CIS mapping.
  - **Services per Transaction (Acquiring, Issuing):** Further decomposition could lead to separate microservices dedicated to specific transaction types, allowing for granular scaling and specialization.
- **The design respects the SOLID** This pattern ensures that each service is maintainable and testable on its own
  - **S (Single Responsibility):** Each service follows Single Responsibility so it doesn't have different reason to change.
  - **O (Open Close):** New functionality can be added (extended) without altering the code already implemented in those modules. For example the addition of a new Acquirer will not alter the service.
  - **L (Liskov Substitution):** It is respected so an object can be passed as super class and still behave as expected.
  - **I (Segregation):** No one class implements all methods they should implement some of them (and only the needed)
  - **D (Inversion of Dependency):** Modules don't depend on concrete classes, but abstraction.
- **Design Pattern Utilization:**
  - **Repository Pattern:** Abstraction of data access, promoting testability and decoupling from specific data storage technologies.
  - **Factory Pattern:** Dynamic construction of routing paths
  - **Adapter Pattern:** Abstracts complexities from external third-party libraries
  - **Strategy Pattern:** Utilized to add new services and processes without altering the old ones.
- **SDK-Driven Development:** A dedicated SDK encapsulates CIS-specific data structures, enumerations, and utility functions to provide a consistent and reusable interface for developers working with the CIS channel. This reduces code duplication and promotes adherence to the CIS protocol standards.
- **Benefits Realized:**

- **Improved Modularity:** The clear separation between the Connector and Protocol services significantly improves code organization and maintainability.
- **Independent Scalability:** We can now scale the Connector service to handle increased network traffic or scale the Protocol service if complex mapping logic becomes a bottleneck.
- **Enhanced Resilience:** Failures in the Connector service (e.g., network issues) do not necessarily impact the Protocol service, and vice versa. This is a consequence of decoupling.
- **Faster Deployment Cycles:** Teams can iterate and deploy updates to individual microservices without affecting other parts of the system.
- **Increased Interoperability:** Enables seamless integration with other services within the ecosystem, facilitating new functionalities like real-time fraud monitoring.

1.4.4 Main technologies used :

The development is based on a modern technological foundation consistent with industry standards:

Figure 64 : spring boot logo

In the backend part of the application, the company required us to use the Spring Boot Java Framework. From my research I have done I found that Spring Boot greatly facilitates application development by providing default configurations and automating many common tasks.

This allowed us to focus on business logic rather than technical configuration.

- Spring Boot is well-suited to microservice architecture. My teammates and I were each tasked with developing a specific microservice.
- Spring Core: The foundation of the Spring ecosystem, it manages dependency injection, includes Spring MVC for the web and modules for database access.
- Spring Data: Allows you to easily communicate with different types of databases using interfaces and naming conventions.
- Spring Security: Essential module for authentication, authorization, and API security, although complex to master.
- Spring Cloud: Key tool for administration and tolerance
- to failures in microservices architectures.
- Spring Boot: Simplifies autoconfiguration, dependency management, monitoring, and deployment of Spring applications.

The team systematically used Spring Initializr (start.spring.io) to quickly generate suitable Spring Boot projects with the necessary dependencies, which standardized the configuration and reduced the risk of errors.

Figure 65 : spring integration logo

Spring Integration makes it easy to implement complex message flows between components:

- It provides communication channels (MessageChannel),
- adapters (KafkaAdapter),
- routers and filters.

This allows for the construction of robust, easily testable and reusable flows.

Channels used:

- QueueChannel
- PriorityChannel
- DirectChannel
- ExecutorChannel

Figure 66 : kafka logo

In the backend, the company imposed the use of Apache Kafka, an essential distributed messaging platform for managing real-time data flows thanks to its distributed architecture, high availability, and low latency. Kafka decouples components via a publish-subscribe model: producers publish events to topics, while consumers subscribe to process messages at their own pace, which promotes horizontal scalability and resilience, as each broker can replicate partitions across multiple servers.

The integration of Kafka made it possible to centralize and make reliable exchanges between microservices, legacy applications, databases and external systems, to ensure the durability of messages via a distributed commit log, to facilitate scaling by increasing partitions or consumers, and to manage message consumption flexibly while guaranteeing order within partitions.

The main components used are the Producer, the Broker, the Consumer, the Topic, and the Partition. Kafka has thus established itself as the foundation for asynchronous integration, ensuring robustness, scalability, and interoperability with various technologies, while guaranteeing the consistency and resilience of distributed processing.

Kafka has therefore established itself as the foundation for integration and asynchronous communication of our system, allowing us to build robust, scalable data pipelines that can be easily interfaced with various technologies (databases, APIs, legacy systems, cloud, etc.), while ensuring the consistency and resilience of distributed processing.

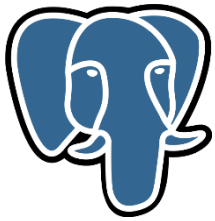


Figure 67 : postgres logo

Relational database used primarily by Switch components. In this project, it is mainly used for:

- store configuration settings,
- manage session metadata.

Figure 68 : RocksDB logo

RocksDB is an ultra-fast embedded key-value database.
It is used locally in the Protocol microservice to:

- temporarily store authorization contexts,
- allow the correlation between a request and its response,
- ensure reliable management even if the microservice is restarted

Figure 69 : Open Telemetry logo

In distributed and microservice architectures, observability has become essential to ensure reliability, performance, and security. It allows you to understand the internal state of a system through its outputs (logs, metrics, traces) and offers a global, real-time view, beyond simple monitoring.

Observability is based on three pillars:

- Logs (event records).
- Metrics (aggregated measurements like latency or error rate).
- Traces (tracking requests between services).

The company adopted OpenTelemetry, a standardized open-source framework that facilitates application instrumentation and the unified collection of logs, metrics, and traces, which can be exported to Elastic Stack or other APM tools. OpenTelemetry automates instrumentation, unifies incident correlation, promotes proactive anomaly detection, and optimizes system monitoring, establishing itself as the leading solution for cloud-native observability.

Figure 70 : Open Lens logo

Open Lens is a lightweight graphical interface for monitoring Kubernetes resources (pods, services, namespaces, etc.). In this project, OpenLens was used to:

- monitor the status of deployed pods (Protocol, Connector, Switch, etc.),
- check logs in real time,
- restart pods when the application is updated,
- control persistent volumes (RocksDB...).

Figure 71 : Docker logo

Docker allows you to create local test environments for microservices:

- protocol and connector can be run in isolated containers,
- the developer can test Kafka streams by simulating a complete environment,
- also allows you to launch a local Kafka for processing validation.

This makes testing easier without relying on a remote Kubernetes cluster.

Figure 72 : Gradle logo

To automate the compilation, dependency management, and deployment of Java microservices, the team adopted Gradle as its primary build tool. Gradle stands out for its flexibility, speed, and ability to adapt to complex architectures, particularly thanks to its declarative scripts in Groovy or Kotlin. Unlike more traditional tools like Maven, Gradle makes it easy to customize build tasks, integrate various plugins, and optimize performance through incremental compilation and parallel task execution. Its dependency management system is particularly effective for multi-module projects, typical of microservice architectures, and simplifies the integration of third-party libraries via centralized repositories. Gradle also supports wrappers, which ensures reproducible builds across all development environments. Thanks to its widespread adoption in the Java ecosystem and its compatibility with major IDEs, Gradle is establishing itself as a leading choice for accelerating the development cycle, ensuring reliable deliveries, and facilitating the maintenance of large-scale applications.

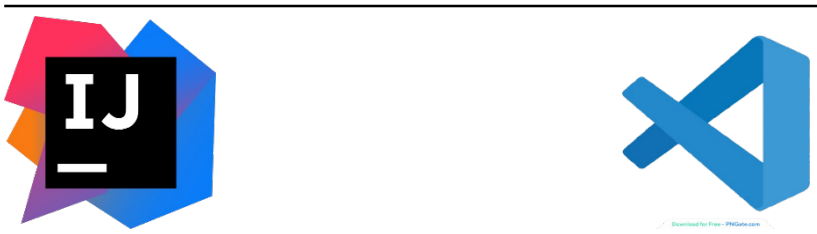


Figure : : Intellij and Vs Code logo

For development:

- **IntelliJ IDEA** was the main environment for the Java part (Protocol, Connector, SDK). Thanks to its advanced features:
  - smart completion,
  - powerful refactoring,
  - static analysis tools,
  - quick navigation in the code,
  - ultra-productive shortcuts (top-tier for Java), it has allowed gains in efficiency and code quality.
- **Visual Studio Code** was preferred for consulting and analyzing legacy C code. Its lighter interface, adapted to C projects, facilitated:
  - reading source files,
  - analysis of existing algorithms,
  - the migration of processing to Java.

Figure 74 : Git and BitBucket logo

The project's source code is versioned on Bitbucket, with full management in Git. This has allowed:

- a branch work (feature / fix / release),
- precise tracking of commits,
- code reviews (pull requests),
- a complete history of technical developments.

Figure 75 : ElasticSearch logo

Elasticsearch is a distributed search and analytics engine, scalable data store, and vector database built on Apache Lucene. It's optimized for speed and relevance on production-scale workloads. Use Elasticsearch to search, index, store, and analyze data of all shapes and sizes in near real time. Kibana is the graphical user interface for Elasticsearch. It's a powerful tool for visualizing and analyzing your data, and for managing and monitoring the Elastic Stack.

Elasticsearch is the heart of the Elastic Stack. Combined with Kibana, it powers these Elastic solutions and use cases:

- Elasticsearch: Build powerful search and RAG applications using Elasticsearch's vector database, AI toolkit, and advanced retrieval capabilities.
- Observability: Resolve problems with open, flexible, and unified observability powered by advanced machine learning and analytics.
- Security: Detect, investigate, and respond to threats with AI-driven security analytics to protect your organization at scale.

1.4.5 Java vs C Migration

The project required migrating from a legacy (monolithic) C codebase to a modern Java Spring Boot architecture, compliant with the microservices standards used in the PowerCARD V4 ecosystem.

This technological change has brought many benefits:

Improved security and memory management

- Automatic garbage collection management
- Better control of memory issues (no manual allocation)

Increased productivity for developers

- Higher-level, simpler, and more readable syntax
- Advanced support by modern IDEs (IntelliJ IDEA, Eclipse)

Easier maintenance and better code readability

- Native exception handling
- Object-oriented programming (OOP): modular architecture, better structured code

Rich ecosystem and extensive support

- Access to many standard libraries (Spring, Hibernate, OpenTelemetry, etc.)
- Large community and abundant documentation

Platform independence

- “Write Once, Run Anywhere” thanks to the JVM
- Cross-platform portability without recompilation

Better compatibility with modern technologies

- In C/C++, it is often complex to find libraries compatible with the latest microservice technologies (Kafka, gRPC, etc.)
- In Java, the ecosystem is mature for modern distributed architectures

Optimizing DevOps cycles

- In C/C++, maintaining Docker images is expensive: frequent updates are needed to maintain security and version compatibility.
- In Java, integration with CI/CD tools is easier, with more stable containerized images.

Limits observed:

- Higher memory overhead (JVM related).
- CPU performance sometimes lower on certain critical cases.
- More verbose syntax than C.

In conclusion, the gains in readability, maintainability, scalability and compatibility with the microservices ecosystem fully justify this technological migration. This transformation has made it possible to thoroughly modernize the software architecture of the CIS solution.

1.4.6 Conclusion :

The transition from a monolithic C architecture to a modern architecture based on Java Spring Boot microservices has enabled a profound transformation of the CIS solution.

The adoption of a robust technological base (Spring Boot, Spring Integration, Apache Kafka, RocksDB, etc.) and a microservices-oriented approach has considerably improved the modularity, maintainability and scalability of the system.

The choices made today offer:

- Better resilience of components,
- Fine scalability per service,
- Easy integration into CI/CD pipelines,
- Compatibility with current standards of the digital banking ecosystem.

The technologies used also guarantee optimal interoperability with modern tools and frameworks used by the Online Product team.

This technological migration not only allowed the software architecture to be modernized but also simplified future maintenance work, functional development and integration with other systems.

The project thus constitutes a solid foundation for future developments of the CIS protocol in the PowerCARD V4 ecosystem.

General conclusion:

This final year project was for me a real immersion into the professional world of modern payment architectures, and more specifically within the PowerCARD Switch developed by HPS.

Participating in the migration of the Mastercard CIS Channel, from a monolithic C code language to a Java Spring Boot microservices architecture, allowed me to consolidate essential skills in backend development, real-time payment flow processing, as well as advanced software industrialization practices.

Throughout this internship, I was able to deepen my understanding of how a payment switch works, the specificities of the Mastercard CIS protocol, and the complex mechanisms related to the processing of Issuing and Acquiring flows. I also gained a clear vision of the performance, security (MAC management, HSM, PIN translation), and reliability constraints required in such a critical environment.

On the technical side, this experience allowed me to master modern technologies such as Spring Boot, Spring Integration, Apache Kafka, RocksDB, and associated DevOps practices. I also progressed in the art of designing robust, scalable, and easy-to-maintain microservices.

From a human perspective, this internship gave me the opportunity to work within an experienced and demanding team, in a stimulating professional environment. I was able to learn how to better organize my work, manage priorities in a complex project, and interact effectively with technical experts.

Finally, this project allowed me to better understand the PowerCARD ecosystem and the modernization challenges faced by a company like HPS, a world leader in payment solutions.

I emerged from this experience enriched both technically and personally, with a better understanding of modern software architectures and a strong motivation to pursue my career in the development of critical and innovative solutions.

Bibliography:

- https://docs.spring.io/spring-boot/docs/current/reference/html/: Spring Boot Documentation
- https://www.baeldung.com/spring-boot: Tutorials in Spring
- https://www.iso.org/standard/75839.html: ISO 8583 overview
- https://developer.mastercard.com/
- https://corporate.visa.com/content/dam/VCOM/download/corpor ate/media/visanet-technology/visa-net-booklet.pdf
- https://www.scribd.com/document/425952095/dlscrib-com-vip- system-base-i-technical-specifications-volume-ii-pdf
- https://www.hps-worldwide.com/product/powercard-switch
- Internal HPS documentation on PowerCard V4
- Mastercard CIS Specifications
- https://chatgpt.com/
- https://claude.ai/
- https://mermaidchart.com
- https://www.deepseek.com/
- https://gemini.google.com