FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Remote metadata extraction in the MANTA tool
Student:	Bc. Vladyslav Zavirskyy
Supervisor:	Ing. Michal Valenta, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

This work aims to design and implement a functional standalone application prototype that manages remote metadata extraction from various data management applications such as databases, ETL and reporting tools. This prototype should serve as a bridge calling various metadata extraction services and sending the extracted metadata to data flow analyzers for further processing.
The application will be part of the MANTA product. A crucial benefit of a separate application for metadata extraction is the possibility to connect to data management applications that are not available outside the customer internal network. Another possible advantage will be improved extraction performance thanks to lower network latency.
The biggest challenge would be integrating the new application into the existing MANTA infrastructure and making this solution secure and scalable.
The main goals of this works are:
1. Learn about the MANTA product.
2. Based on already existing initial analysis (containing requirements and analysis of existing solutions), design a standalone application for managing remote metadata extraction that will be integrated into MANTA product. This should cover design of the application itself, how this application is managed from the existing product (the application manager) and how the metadata extraction is orchestrated (the extraction orchestrator).
3. Implement the MVP of the application, application manager, and extraction orchestrator.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 6 June 2022 in Prague.

4. Prepare both user and programmer documentation for all modules.
5. Test all modules with unit tests and provide integration tests for the whole metadata extraction flow.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 6 June 2022 in Prague.

Master's thesis

REMOTE METADATA EXTRACTION IN THE MANTA TOOL

Bc. Vladyslav Zavirskyy

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Michal Valenta, Ph.D.
January 3, 2023

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Zavirskyy Vladyslav. *Remote metadata extraction in the MANTA tool.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

List of code listings

I would like to thank MANTA and its employees for all the help they provided me and for allowing me to write my thesis with them. Especially to Ing. Jakub Moravec for leading the project, without whom the thesis would not have the quality it has.
Next, I would like to thank the Czech Technical University in Prague, the Faculty of Information Technology, and its professors for all the knowledge they passed on to me. In particular, I want to thank my supervisor Ing. Michal Valenta, Ph.D, who guided me in this work.
Last but not least, I would like to thank my family and friends for their support during my studies.

Declaration

Hereby declare that I have authored this thesis independently, and that all sources used are declared in accordance with the "Metodický pokyn o etické přípravě vysokoškolských závěrečných prací".

I acknowledge that my thesis (work) is subject to the rights and obligations arising from Act No. 121/2000 Coll., on Copyright and Rights Related to Copyright and on Amendments to Certain Laws (the Copyright Act), as amended, (hereinafter as the "Copyright Act"), in particular § 35, and § 60 of the Copyright Act governing the school work.

With respect to the computer programs that are part of my thesis (work) and with respect to all documentation related to the computer programs ("software"), in accordance with Article 2373 of the Act No. 89/2012 Coll., the Civil Code, I hereby grant a nonexclusive and irrevocable authorization (license) to use this software, to any and all persons that wish to use the software. Such persons are entitled to use the software in any way without any limitations (including use for-profit purposes). This license is not limited in terms of time, location and quantity, is granted free of charge, and also covers the right to alter or modify the software, combine it with another work, and/or include the software in a collective work.

In Prague on January 3, 2023 .....

Abstract

This master thesis describes the problematics of the application implementation for remote metadata extraction. The application design and implementation ensure smooth integration of the application to the Manta Flow application. The results of this work are the analysis, design, and implementation of the Agent application.

Keywords MANTA, MANTA Flow, MANTA Flow CLI, cloud, metadata, extraction, agent

Abstrakt

Tato diplomová práce popisuje problematiku implementace aplikace pro vzdálené získávání metadat. Návrh a implementace aplikace zajišťují hladkou integraci aplikace do aplikace Manta Flow. Výsledkem této práce je analýza, návrh a implementace aplikace Agent.

Klíčová slova MANTA, MANTA Flow, MANTA Flow CLI, cloud, metadata, extrakce, agent

Introduction

If there is no struggle, there is no progress. - Frederick Douglass

Today, more and more companies are using cloud technologies to process and store data. In addition to the huge number of benefits, this approach also brings certain limitations and difficulties. One of the challenges is getting to those services running inside of secure private networks and not introducing security weaknesses.

Motivation and objectives

The Agent is a supporting application designed for cloud deployment of the MANTA Flow application. Cloud-based MANTA Flow CLI cannot connect to customers' source systems hidden in their local network and thus cannot extract them. The Agent should solve this problem. It is an on-premises application that extracts source systems that are unavailable from outside the network and sends the extracts to MANTA Flow for further processing.

The aim of the work

This work aims to design and implement the Agent application prototype that manages remote metadata extraction from various data management applications such as databases, ETL, and reporting tools. This prototype should serve as a bridge calling various metadata extraction services and sending the extracted metadata to data flow analyzers for further processing. The application will be part of the MANTA Flow product.

The most significant advantages of the Agent as a separate application are the possibility to connect to data management applications that are unavailable outside the customer's internal network without relaxing proxy inbound rules and possibly improved extraction performance thanks to lower network latency.

Chapter 1

Metadata for data lineage

In this chapter we will take a look at basic concepts that are needed to fully understand the topic in its whole complexity.

1.1 Manta

The target of this master thesis is to implement an application called MANTA Flow Agent that will be integrated into the MANTA project. The application should extract metadata from source systems and send them to the MANTA for further analysis and processing.

"MANTA thrives on helping mid-size and enterprise-level companies drive productivity, gain trust in their data, and accelerate digital transformation. We bring intelligence to metadata management by providing a world-class, data lineage platform that automatically scans your data environment to build a powerful map of all data flows and deliver it through a native UI and other channels to both technical and non-technical users. With MANTA, everyone gets full visibility and control of their data pipeline." [1]

MANTA Flow supports a large number of source systems to which it can connect and extract metadata. Those metadata are collected and analyzed to create a data lineage that visualizes data flow through different systems.

MANTA Flow supports an extensive range of different technologies. The most powerful feature is the possibility to analyze different types of languages (e.g., SQL, Java, Python, and others). MANTA analyzes the languages automatically and extracts the transformational logic described by them. Software is unique through the ability to recognize also hard to read programming code. In a reasonable time (usually a few hours), MANTA can read the whole database with a lot of data stored in different tables and create from its well-arranged data lineage map. It tremendously reduces the time needed to construct the map of data flows. In practice, MANTA is used mainly for data warehouse optimization, cost reduction of software development, impact analysis, and environmental documentation for regulatory agencies' needs.

The main MANTA product is MANTA Flow, which consists of multiple components that work together to provide data lineage for a user.

1.2 Metadata

If to put it simply, metadata are data about data. They may be presented in different formats. and describe different data. For instance, the metadata of a photo taken by a modern phone are the following:

- Date and time
- Location
- Size
- Resolution

When constructing the flow of data from one place to another based on metadata, the following metadata may be useful:

- Definitions of jobs in ETL tools
- Database SQL scripts
- Definitions of tables and columns from a database

MANTA constructs data lineage using all possible varieties of metadata from different systems.

1.3 Data lineage

Data lineage helps to understand how data flows through complex data systems.

"Data lineage is generally defined as a kind of data life cycle that includes the data's origins and where it moves over time. This term can also describe what happens to data as it goes through diverse processes. Data lineage can help with efforts to analyze how information is used and to track key bits of information that serve a particular purpose.

One common application of data lineage methodologies is in the field of business intelligence, which involves gathering data and building conclusions from that data. Data lineage helps to show, for example, how sales information has been collected and what role it could play in new or improved processes that put the data through additional flow charts within a business or organization. All of this is part of the more effective use of the information that businesses or other parties have obtained.

Another use of data lineage, as pointed out by business experts, is in safeguarding data and reducing risk. By collecting large amounts of data, businesses and organizations are exposing themselves to certain legal or business liabilities. These relate to any possible security breach and exposure of sensitive data. Using data lineage techniques can help data managers handle data better and avoid some of the liability associated with not knowing where data is at a given stage in a process." [2]

In figure 1.1, we can see an example of how visualized data lineage may look like. Source of the figure is [3]

MANTA can analyze and visualize two types of data flow: direct and indirect. Direct flows are responsible for transferring a piece of information from one place to another. Indirect flows (or filter flows) affect direct flows. Indirect flows affect information transfer from one place to another. Indirect flows are represented by WHERE constructions and IF conditions. For instance, in IBM InfoSphere DataStage, we will specify columns that should be used as sorting keys if we would like to sort data as one of the stages in a job. A key is a column on which to sort the data. For example, if we have a salary column, we might specify this column as the sort key to get a sorted list of salaries. In this situation, the key column is the source of the filter flow, and the rest of the columns are the destination of the indirect flow.

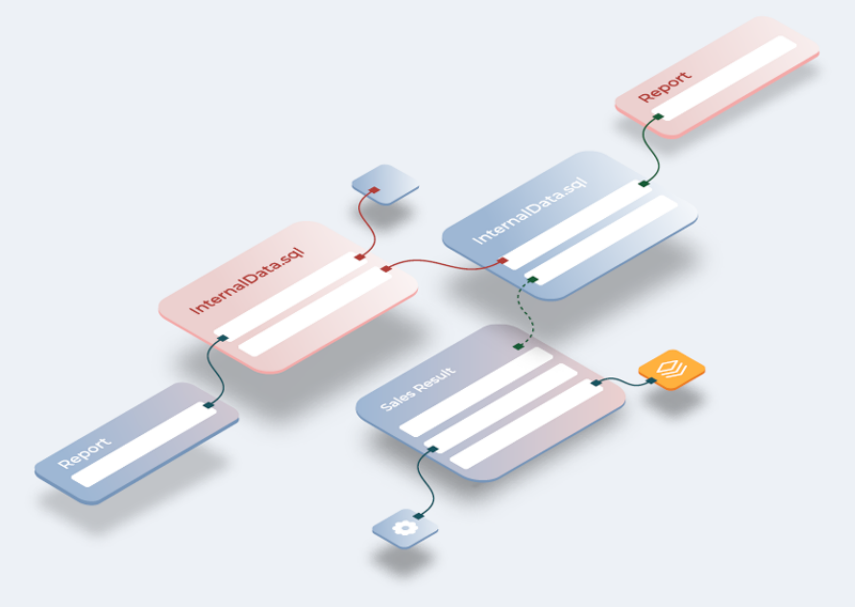


Figure 1.1 Data lineage visualization example (source [3])

1.3.1 Importance of data lineage

This subsection is heavily based on the "Why Are We Talking about Data Lineage in 2022?" section from the following article [3]

As time goes on, humanity generates and collects more and more data, often without thinking about what is being collected and why. For a long time now, companies have been using historical data for business decisions. Over time, many governments have begun to regulate what data is collected and how it is used. At the same time, the amount of data collected, as well as the data infrastructure, is growing in complexity. Simple systems are evolving into systems with many links and dependencies. So complex systems are too big to be analyzed by humans, and there are consequences.

Possible consequences because of poor data infrastructure understanding:

- Slower delivery of new analytical/predictive insights

  The reason for this is not a complete understanding of the environment. Frequent and difficult trackable changes in the environment also do not prosper the environment understanding. According to the statistics collected by MANTA, up to 40 percent of data engineering resources are spent on unproductive impact analysis, just assessing the impact of new development requirements, which is a huge number.

- Decreasing level of trust in reports, dashboards, and insights

  It happens because the origin of numbers in the report cannot be explained and presented to the business, which leads to a lack of trust in numbers and unnecessary prolonging of business decisions. Many efforts and days are wasted to answer even a simple question.

- Growing number of data incidents

  It is pretty difficult to assess the end-to-end impact of to-be-implemented changes in complex data systems. It is a better scenario when possible problems are seen immediately. However, it is much worse when problems emerge in a production environment where it is not a place for a mistake.

- Severe shortage of data engineering talent

  Every year number of data produced and collected increases, thus the number of data engineers required for their processing. Demand for data engineers goes up by 50 percent every year, and there are no signs of lower demand yet.

- Increasing risk of non-compliance and regulatory penalties

  Governments constantly implement new data regulation policies to control data collection and processing. Non-compliance with regulations may cause massive damage to a company through fees and reputational losses.

1.3.2 Data Lineage use cases

This subsection is heavily based on the ”Data lineage use cases” section from the following article [4]. This article is written by Collibra, one of the market leaders in data intelligence.

It is a number of different usage patterns of how data lineage can be used. We will take a closer look at them in this subsection.

- Regulatory compliance

  Pass regulation checks may be tricky and not that straightforward without data lineage. Data lineage helps businesses comply with regulations such as BCBS 239, GDPR, CCPA, and others. Businesses can retrieve information needed for reports much more quickly and reduce the possibility of mistakes using data lineages created by automatic tools. The manually created mapping may be inaccurate and can become obsolete quite fast.

- Self-service analytics

  By providing context around your data, data lineage facilitates better analytics and decision-making. Analyzers can discover relevant data context, such as source changes and usage, by exploring the data lineage upstream and downstream. Analyzing data assets in more detail allows business analysts to determine how they were created and where they originated. Making business decisions based on accurate, complete, and trustworthy data ensures business success.

- Impact analysis

  Data lineage makes it easier to conduct an impact analysis at a detailed level. Data lineage diagrams allow a data analyst to identify any change and upstream and downstream impacts quickly. The analyst can drill down and see the impacts on a table, column, or business report level.

- Data exploration and viability

  Data lineage allows to discover of all parts of data lineage and ensures precise analytics and decision-making.

- Asset management

  Data lineage helps a data analysis to identify the most usable and the least usable data assets across the companies data system.

- Cloud migration

  Data lineage makes it easier to plan and perform data modernization initiatives, such as moving data warehouses to the cloud by identifying and documenting the critical data elements for cloud migration.

1.3.3 Real life Data Lineage use cases

Bellow we will take a look at some real life examples of how data lineage may be used. Source of real life use cases is [5]

- Zero Trust implementation

  Zero Trust implementation means building a security model based on the idea that no user is trusted by default. Every user should be authenticated and authorized.

  The first step on the way to have it implemented is to define the space that we want to secure. Elements that should be determined are critical data, applications, and assets. In other words, what will be given the highest priority in the event of an attack.

  It is impossible to secure space without knowing what data are stored, where and how. Data lineage obtained from automatic solution is a good way how to retrieve required details.

- Self-writing documentation

  Ideally, the development processes and changes in the environment needed to be recorded, not only to improve future development but mainly to understand their data continuously. However, it is not that easy to achieve that. A straightforward approach is to write down, record, and share information about the change with the rest. It sounds easier to achieve than it is. Keeping documentation in its current state consumes an enormous amount of time and energy, which may be spent on developing new features. After all, developers only exist to make changes in the environment.

  Every team of developers has undocumented knowledge. They know how they have built the environment, and they are the ones continuously working on it. Some parts of the knowledge can be documented, but it is just impossible to have documented all the knowledge. Also, people come and go. Often, there was no prior documentation for a task that was about to be completed simply because no one had ever done that before in this company. Automatically created data lineage serves as a perfect documentation of data environment that should not be updated manually. With automated data lineage solutions, data can be read and documented automatically without any prior information.

- Fear of change

  Very complex data environments consisting of multiple databases constantly under development make it a difficult task to handle. The longer such environments exist, the more code becomes legacy code, which does not make the task easier.

  Sooner or later, in development teams, the question will start floating around in everyone's heads. The question is, what is going to break if we make changes? Data lineage can show the development team the impact a new release will have on existing code, making it easier to catch most of the larger issues during the testing phase. Introducing the data lineage will reduce the time needed for testing and release. Also, it will reduce the number of significant issues left uncached before deploying the code to the production environment.

1.4 Data lineage construction

This section is heavily based on the "Let's Get Technical: How to Create Data Lineage and Keep It Up to Date" section from the following article [3]

Metadata are quite often associated with simple things like tables, columns, and reports. In lineage metadata, however, the focus is more on logic - instructions or code in any form. It can be an SQL script, a procedure stored in a database, a job in a transformation tool, a Java API call, or a complex macro in an Excel spreadsheet. Anything that moves, transforms, or modifies data is data lineage. Before Data lineage construction, two moments should be considered: which data source to use to get metadata for data lineage and how to build data lineage.

1.4.1 Metadata sources for Data lineage

Metadata for data lineage construction may be obtained from three primary sources: data, logs, and code.

- Data as a source/pattern-based lineage

  This technique reads metadata about tables and columns and uses information about data profiles to create relationships representing possible data streams based on common patterns or similarities. Examples of such similarities are tables or columns with similar names and columns with very similar data values. Furthermore, if many similarities are found between two columns, they can be linked together in a data flow diagram.

  There is one significant advantage to this approach. If only data is tracked and not algorithms, there is no need to worry about technology. Any technology is used, and each technology's details are unimportant. If it is possible to retrieve data from it, then it is enough as it can be analyzed. However, this approach is usually not accurate. The impact on performance would be significant as we are working directly with the data, which amount may be enormous. Working directly with data also opens security risks, as data may be leaked.

  Even though this approach is not perfect, it may be sufficient in some cases. It may be helpful in cases when it is impossible to retrieve logic from the technology in some format that may be later analyzable. The approach is also good when it is needed to cover data flow outside any system.

- Logs as a source/run-time lineage

  This approach uses all possible forms of runtime information to construct data lineage. The following runtime information may be used for data lineage construction: log files, runtime workflows from ETL/ELT tools, and any other source with sufficient runtime details. This approach is sensitive to different technologies and data stacks, as each technology may have a different format. Even input and output data formats may differ within one technology, so this moment should be considered. Different techniques, such as regular expressions, AI/ML, and others, may be used to identify important parts of log files. Also, data tagging may be used to tag data, so data are traceable from start to finish of data lineage.

  The runtime nature of the data lineage is quite helpful for incident resolution. It gives exact information about the flow of data that was recognized as incorrect. This strength is also its biggest weakness. The data lineage is generated only based on executed data streams that may cause inaccurate data lineage. The main word here is executed. If the data stream is conditional and it goes from one place to another, only if some condition is fulfilled, then the data lineage will not show this flow until that condition is fulfilled. Because of this feature, some parts of data lineage may stay undetected.

  Another limitation of the data lineage constructed based on runtime information is the absence of details about data transformation. Not all runtime details are logged, especially in the case of complex algorithms. Enabling debug logging level is essential to retrieve as many details as possible. Mainly the data lineage can capture only high-level details of data flows

  Blindly using such metadata poses a high risk to the organization. If such data lineage is used:

  - to perform an impact analysis, it increases the risk of problems during development and implementation of new features.
  - to prepare a regulatory report, it increases the risk of inaccurate reports, which may cause incidents and fines.
  - to analyze and prepare data to train a new model, it increases the risk of an incorrectly trained model.

- Code as a source/design lineage

  This approach assumes that we have direct access to the source code, so we can look directly into the code that processes and transforms data records. Code here means any definitions of data transformation and data moving. Such definitions may be in the form of scripts, database-stored procedures, ETL/ELT workflows, Java code, and many other possible formats. A large number of different technologies require specialized scanners for each technology as it is impossible to use one general scanner for all technologies, as it is with logs.

  Even though this approach is the most difficult to implement, it possesses significant advantages that are not achievable by other approaches. Because of it, this approach is the primary strategy when a precise data lineage is needed.

  - This approach is the most accurate. Incorrectly detected data flow is almost impossible. Trustworthy data lineage simplifies problem investigation, making it faster and more effective.
  - Aside from other approaches, this approach detects indirect (filter) data flows. Indirect data flow impacts direct data flow, which is difficult to detect by any other approach. This feature is essential when planning changes in the data system or migration projects.
  - This approach detects all pieces of data lineage without a minimal probability of missing anything. Complete data lineage is crucial for migration projects and doing any changes in the data system.
  - This approach captures details of the transformation logic used to transform data. This feature is vital for writing reports for regulatory compliance.

1.4.2 How data lineage is constructed

There are different ways how to construct data lineage. Below, we will take a look at the currently most used approaches.

- Manual Data Lineage analysis

  This approach is the most simple at first glance, but with time, it is the most time-consuming and inaccurate. This approach is basically about manually creating data lineage and updating it with repeatable revisions. It is a top-down approach, as it starts with the mapping of knowledge possessed by people in the organization. When enough information is gathered, data lineage is created in some vizualization tool.

  Manual data lineage construction is mainly done based on source code. The code is analyzed manually by one person or a team. In the better case, the analyst will be the same person who wrote the code. Manual code examination requires a lot of time and knowledge about source code. The most challenging moment of this approach is to keep up to date with the current state of the data system. When any changes occur in the data system, they should be manually propagated to the data lineage.

  This approach may be good in cases when there is no code, or it is inaccessible. In such cases, the only way to construct data lineage is to get information about data flows from people working in the organization.

- Self-contained Data Lineage analysis

  This approach is fully automated and used by ETL/ELT vendors. Data lineage and workflow definition are basically the same things in ELT/ELT tools. The concept is that all operations with data are performed within one tool that will also provide data lineage.

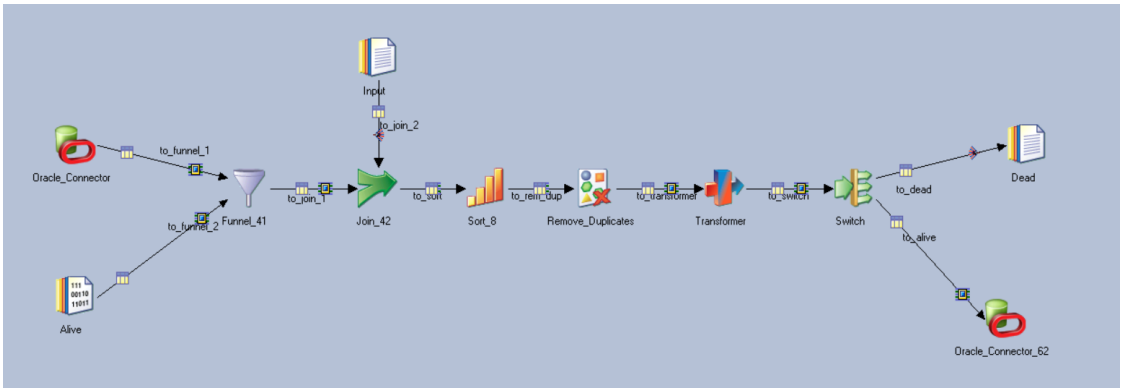


Figure 1.2 Example of parallel job definition in IBM InfoSphere DataStage

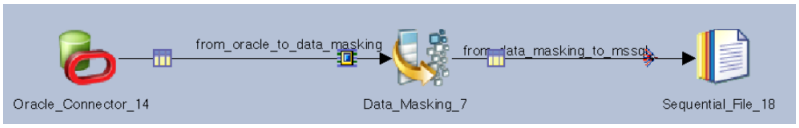


Figure 1.3 Parallel job definition in IBM InfoSphere DataStage

The main disadvantage here is that data lineage is limited only to the platform. All data transformations and movements that are happening outside of the platform are not captured by data lineage. Also, earlier or later, a data engineer will face the limitation of the platform.

Example of such workflow definition in IBM InfoSphere DataStage may be seen in figure 1.2. This workflow definition may be used also as a data lineage.

- External automated data lineage analysis

  This approach is also fully automated as the previous one. The biggest advantage of this approach, compared to the previous one, is that this approach is not limited to one technology. The external data lineage tool may access different systems and construct data lineage based on information from different sources.

1.5 Example of data lineage construction

In this section, we will take a look at the example of data lineage generated by Manta. This lineage involves IBM InfoSphere DataStage parallel job, PostgreSQL database and Oracle database. For the sake of brevity, we will limit ourselves to several stages.

IBM InfoSphere DataStage is an ETL (extract, transform and load) tool from IBM. For more information about how DataStage is analyzed in MANTA, see the bachelor thesis "Design and implementation data flow analysis of jobs in IBM DataStage for Manta project" [6]

In figure 1.3 we can see a simple Parallel job definition in IBM InfoSphere DataStage. As it was written before, this job definition may also be directly used as a data lineage. The most significant disadvantage of such data lineage is that we cannot see what is happening outside the job definition. In this case, we cannot see what else is happening inside the Oracle database or with the destination file.

In figure 1.4 we can see data lineage of the job generated by MANTA. The data lineage also shows movements and origins of data within Oracle database.

In figure 1.5 displayed one interesting feature of MANTA, flow highlighting. This feature allows easily trace movement of some particular data within data flow.

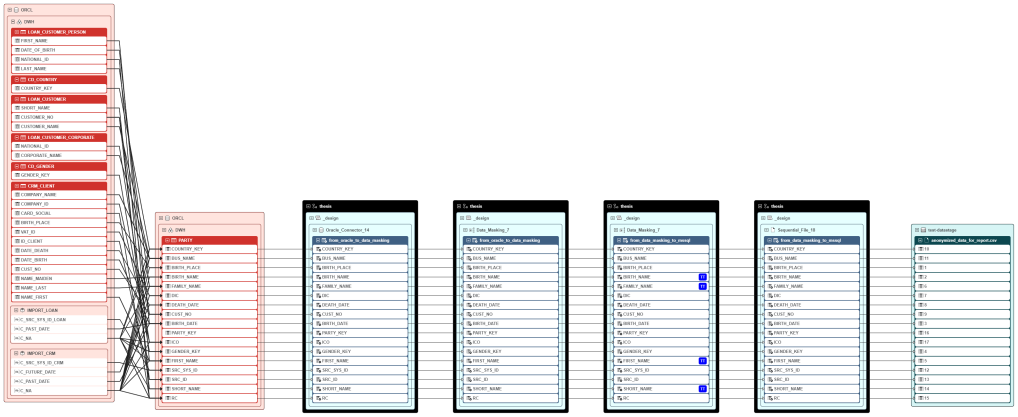


Figure 1.4 Data lineage of IBM InfoSphere DataStage parallel job generated by MANTA

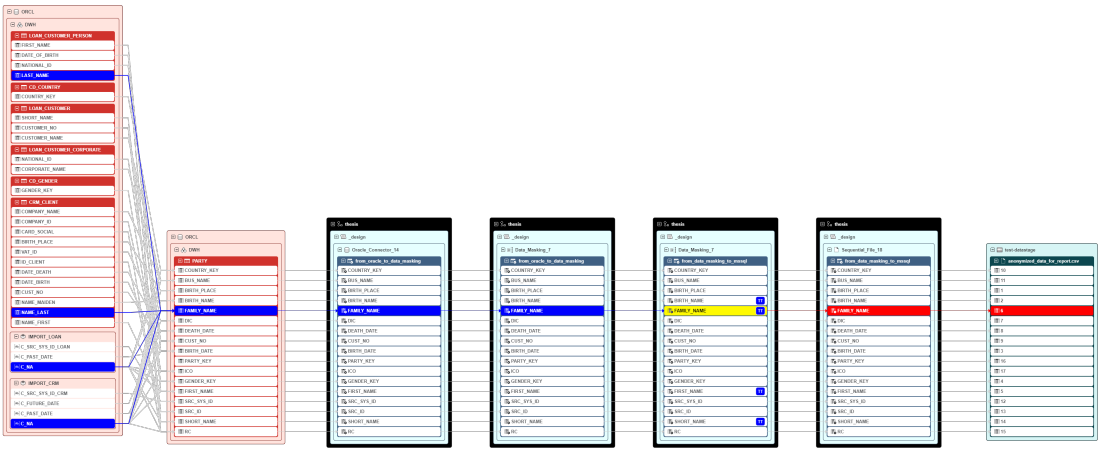


Figure 1.5 Data lineage with highlighted family name flow of IBM InfoSphere DataStage parallel job generated by MANTA

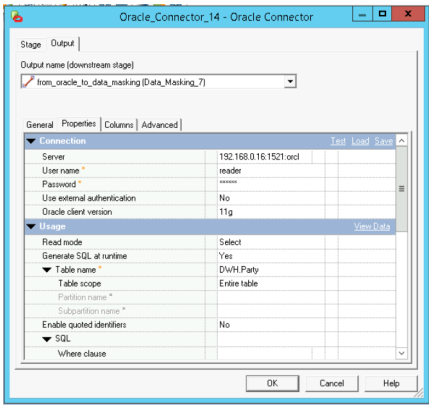


Figure 1.6 Oracle connector stage configuration in parallel job definition in IBM InfoSphere DataStage

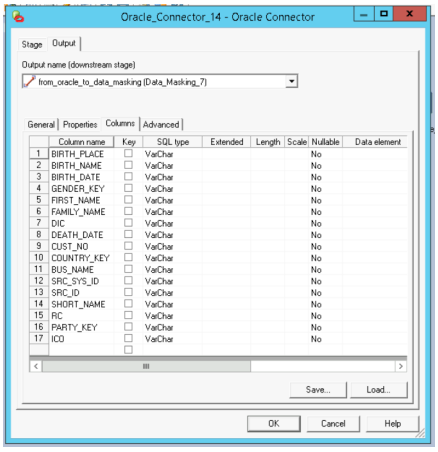


Figure 1.7 Oracle connector stage output columns configuration in parallel job definition in IBM InfoSphere DataStage

This parallel job defines the extraction and preparation of the data for later use. The job consists of three stages, where each stage represents extraction, transformation, or a data load. The stage is a basic element of jobs in IBM InfoSphere DataStage. The parallel job contains the following stages ordered from the start of the flow to its end:

- Oracle database connector stage

  This stage connects to Oracle database and extracts or loads data to Oracle database. In figure 1.6 we can see a configuration of Oracle database connector stage.

  The connector is configured to extract data from the Party table stored in the DWH schema. SQL used to extract data will be generated during runtime, so we do not to have define it manually, but it is also a possibility to do so.

  In figure 1.7 we can see definition of output columns of data that should be extracted from the database.

  Oracle database connector is represented in figure 1.5 by node which is connected by its input with the Party table in the DWH schema, what corresponds to the connector configuration. The node output is connected with data masking stage.

- Data masking stage

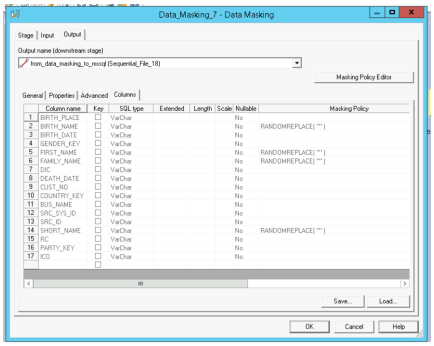


Figure 1.8 Data masking stage configuration in parallel job definition in IBM InfoSphere DataStage

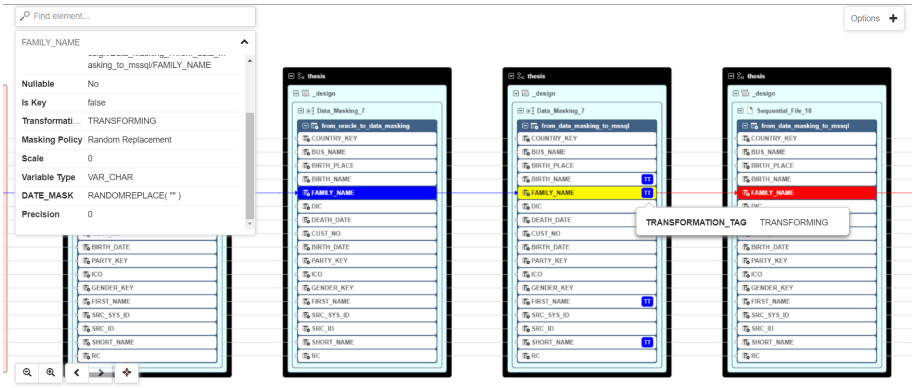


Figure 1.9 Data masking stage visualization in MANTA

This stage masks sensitive data that should be used for analysis, reports and so on. In figure 1.8 we can see a configuration of Data masking stage. The stage is configured to replace data in BIRTH_NAME, FIRST_DATE, FAMILY_NAME and SHORT_NAME columns with random strings, so data are anonymized.

Data masking stage is represented by two nodes in 1.5. First node represents input links of the data masking stage, when the second node represents output links. Data masking policies are configured in output links, what is also visualized in data lineage. In the figure 1.9 is shown, that masked columns are tagged with transformation tag. Used masking policies and additional informations are shown in the columns properties.

- Sequential file stage

  This stage reads or writes data to text files. In figure 1.10 we can see a configuration of sequential file stage.

  The stage is represented in data lineage in figure 1.5 by one node. The node is connected by its outputs to anonymized_data_for_report.csv file stored on filesystem.

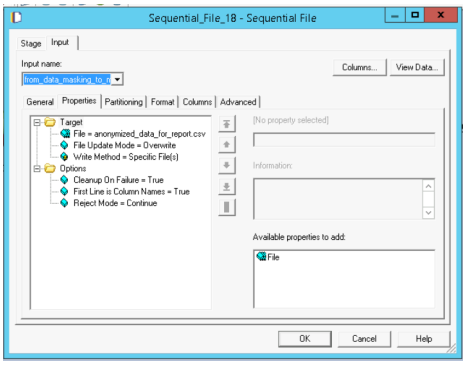


Figure 1.10 Sequential file stage configuration in parallel job definition in IBM InfoSphere DataStage

Chapter 2
Analysis

In this chapter, we will talk about the analysis, which took place on the start. The chapter is divided into five parts. In the first part will be discussed requirements analysis with functional and non-functional requirements. In the second part, we will choose technologies for further implementation, and after in the next part, we will talk about actual MANTA Flow structure. In the fourth part, we will say some words about Agent intergration into MANTA Flow.

Analysis is heavily based on internal MANTA materials.

2.1 Requirements analysis

This section captures all features and constraints that must be addressed in the proposed solution.

2.1.1 Functional requirements

"In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

Functional requirements in software engineering help you to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform." [7]

- F1: Extraction from customer's private network

  Allow MANTA Flow Cloud to run extraction (and eventually also Export into third-party tools) from the customer's infrastructure in a private network.

  Two-way communication is preferred as we can avoid polling or similar patterns, but MANTA Flow Cloud can never open an outbound connection to MANTA Flow Agent as this would not be allowed by firewalls in the internal network.

- F2: Terminate running operation

  The user must be able to terminate any operation running on MANTA Flow Agent from MANTA Flow Cloud.

- F3: Registering and Disconnecting agents

  The user should be able to register an agent to MANTA Flow Cloud. That will be performed by starting Agent at the on-premise platform, which will auto-register itself to the MANTA Cloud if it is configured correctly.

If there are multiple agents registered, one Agent may have access only to a limited group of third-party technologies. We need to define which Agent has access to which technology to choose an agent which will handle a particular extraction request. We may want a user to tell which Agent has which technology, and agents may perform a scan of the environment.

The user will be able to disconnect the connected Agent to the MANTA Cloud without getting a notification about lost connectivity to one of the Agents. If users terminate Agents on the on-premise side, the MANTA Cloud should show some notification about lost Agents.

- F4: Perform Extraction

  Agent must be able to perform extraction on-demand triggered from MANTA Flow Cloud.

  To execute extraction scenarios agent must implement the following list of technologies, for example, JDBC, REST, SDK and shell scripts.

  In case of extraction with a SDK, the agent will not be able to have all libraries necessary to perform the extraction. These libraries must be added by users manually - the lib-ext folder must be available for that.

- F5: Stream-upload Extracted Metadata to MANTA Flow Cloud

  The default transport mechanism will be the upload of the extracted metadata directly to the cloud without the necessity to store the metadata on a local disk.

- F6: Local Disk Extraction

  The customer has to be able to configure the Agent to store the extracted metadata on the local disk instead of sending them to MANTA Flow Cloud. This will be used as a fallback option in case MANTA Flow Agent won't be able to send the metadata directly to MANTA Flow Cloud (for whatever reason).

- F7: GUI

  As Agent should be configured during installation, the GUI should be provided to have a more user-friendly installer.

  It is not decided yet, how it will be possible to see the status of MANTA Flow Agent directly on the machine where it is installed. Primary source of information is MANTA Flow Cloud, but at least something should be on the side of MANTA Flow Agent as well.

2.1.2 Non-functional requirements

"A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?

A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are ¿ 10000. Description of non-functional requirements is just as critical as a functional requirement." [7]

- NF1: Performance

  The extraction performance of the MANTA Flow Agent should be ideally identical or slower in tenths of a percent (1.1x, 1.2x) to the performance of MANTA Flow CLI, with the possibly significant difference that MANTA Flow Agent will often be geographically closer to the source system that is being extracted.

- NF2: Documentation

  For the Agent should be created inner documentation on MANTA's Confluence. Code will be documented using Javadoc, and in addition, user documentation with an installation manual will be created on the MANTA's Knowledge Base.

- NF3: Compatibility

  Agent must be deployable on both Windows and Linux. Agent must support the same list of Linux distributions as MANTA does.

  The Agent will have a list of MANTA Cloud versions with which it is compatible. In addition, MANTA Cloud will validate the Agent's version when communication between them is initiated and forbids communication if their versions are incompatible. MANTA Cloud will offer an update for Agent if required.

  There must be backward compatibility for communication between Agent and MANTA Cloud, which they will use to do Agent's update.

  As for MANTA Flow Cloud, we should not do anything in this architecture that would restrict MANTA Flow Cloud from being independent of a proper public cloud environment (meaning – not "require" that Amazon AWS components, for example, are part of the solution). This would then allow flexible configurations, even if a site's "cloud" is really just a multi-machine infrastructure where everything is on-premises but distributed.

- NF4: Configurability

  The agent needs to know the MANTA Cloud IP address and port. The agent must be able to prove its identity to MANTA Flow Cloud - certificate or username with password. The agent must be able to retrieve passwords from a security vault (CyberArc or equivalent). The vault should be accessed from MANTA Flow Cloud for cloud-based technologies and from MANTA Flow Agent for extraction of on-premise technologies. MANTA Flow Cloud will store the configurations for connection to the vault(s) and will provide them to MANTA Flow Agent when necessary. In addition to that, it should be possible to store the connection information to vault(s) on the MANTA Flow Agent side - for security reasons. The connection's configuration will be obtained from MANTA Flow Cloud on demand. Deployment (e.g. connection to MANTA Flow Cloud) configuration of MANTA Flow Agent must persist the update process without the necessity of manual merging.

- NF5: Information security

  The agent needs to know the MANTA Cloud IP address and port. The agent must be able to prove its identity to MANTA Flow Cloud - certificate or username with password. The agent must be able to retrieve passwords from a security vault (CyberArc or equivalent). The vault should be accessed from MANTA Flow Cloud for cloud-based technologies and from MANTA Flow Agent for extraction of on-premise technologies. MANTA Flow Cloud will store the configurations for connection to the vault(s) and will provide them to MANTA Flow Agent when necessary. In addition to that, it should be possible to store the connection information to vault(s) on the MANTA Flow Agent side - for security reasons. The connection's configuration will be obtained from MANTA Flow Cloud on demand. Deployment (e.g. connection to MANTA Flow Cloud) configuration of MANTA Flow Agent must persist the update process without the necessity of manual merging.

- NF6: Extraction logging

  MANTA Agent will send extraction logs to MANTA Flow Cloud to Log Viewer.

- NF7: Audit logs

  MANTA Agent will generate a log of networking requests and logs of performed operations. If multiple instances are running simultaneously, each process will use its own log file with a

suffix of instance ID. Logs will be stored in the 'logs' directory in the MANTA Flow Agent installation folder. In addition, logs will also be sent to the MANTA Flow Cloud to Log Viewer.

- NF8: Memory

  Max allocated memory must be lower than the memory required by MANTA. We should aim to fit into the default max heap size 256 MB.

- NF9: Technologies, frameworks, and libraries

  The application will be written in Java using the Spring framework.

- NF10: Reliability

  If MANTA Flow Cloud loses connection to one of the connected agents, it will display a notice. MANTA Flow Agent will try to reopen communication every configurable-value seconds if it loses it for an infinite time. If there is running extraction, the agent will try to reopen the connection every configurable-value seconds for an infinite time. This period should be shorter than in standby mode. When reconnection isn't possible in a reasonable time, we can either shut down the extraction process or store the metadata on the local disk. When extraction streaming runs into issues with connectivity Agent will pause the extraction. No caching should be used as it requires more memory. In case of the connection lost, the agent should have a trace log about the problem when the connection is established again.

- NF11: Architecture

  The agent must include the extraction logic from Manta Flow CLI. The agent does not have to be a web application. The agent will be the initiator of the communication.

- NF12: Integrations to other systems and interface specification

  Integration with CyberArk (and possibly other vaults).

  Integration with MANTA Cloud.

- NF13: Monitoring

  Customer's MANTA Agents will be monitored from MANTA Cloud. Monitoring works locally on the same machine where the Agent is running, and monitoring metadata are sent to the cloud for further processing and visualization. It was resolved within the following master thesis. [8]

- NF14: System installation and update

  The application must be installed into the customer's environment. VMware Installbuilder should be used as it is already used by MANTA for MANTA Flow installation.

  It will be possible to update the Agent manually with an updater and remotely from MANTA.

  The Agent should be able to perform self-update, and it must preserve configuration without merging.

  For remote updates, Agent will be able to download an update package to perform an update automatically. Updates will be handled from the cloud by sending the update package to the Agent.

  There must be a way to update Agent manually. If a manual update will require an Agent shut-down, the Agent has to be able to start itself again automatically.

  The MANTA Flow Agent will provide MANTA Flow Cloud with its version. MANTA Flow Cloud will check the version compatibility.

  The update process must not introduce any vulnerabilities.

- NF15: Testing

  If the custom solution is chosen, the application will have a JUnit test, and code coverage will meet MANTA standards. Automatized end-to-end tests have to be created.

- NF16: Administration

  The minimal and initial configuration will be done in the installer, which means credentials and MANTA Flow Cloud URL. This process has to be repeatable - rerunning the installer should allow modifications.

  Further configuration should be done from the cloud Agent should be able to receive this configuration. Ideally, that would mean if users change something for the agent and during the next polling interval, it will be downloaded (or immediately pushed if two-way communication is available).

- NF17: The same extraction result as through the CLI

  The extraction result after extraction with the Agent should be the same as extraction with CLI. It is essential to deliver the same result data lineage after analyzing the extraction outputs with the Agent and CLI.

2.2 Selecting approach

In this section we will take a look at existing solutions, will discuss possible approaches how to implement the Agent and in the end, will select the best approach.

2.2.1 Existing solutions

This subsection briefly looks into existing solutions and what approaches these companies use.

2.2.1.1 Remote Engine from Talend

"A remote engine is an on-premises execution environment that can be installed behind your firewall or security processes within your corporate infrastructure, giving the Talend Studio access to local resources, such as files, databases, or other applications." [9]

"Remote Engines allow you to run tasks that use on-premises applications and databases.

Remote Engines allow you to run Job, Route, and data service tasks that use on-premises applications and databases. By default, maximum 3 tasks can run in parallel on the same engine. If you are using a Remote Engine to run identical tasks simultaneously, your Remote Engine must be v2.12.0 onwards.

Outbound communication between Talend Cloud Management Console and Remote Engines is fully secured as data is not staged.

Upload a local file (for example, an excel or CSV file listing new customers) to a cloud application (for example, Marketo). In this case, Remote Engines are within your data center with access to the needed resources (local file system, databases and so on) and the data ends up directly in the destination cloud application (like Marketo or Salesforce) without going through Talend Cloud Management Console, as the communication is only outbound." [10]

In the figure 2.1, we can see diagram of local to cloud / cloud to local data transfer using Talend Remote Engine.

From the description and documentation, Remote Engine looks like a Thick Agent. The Thick Agent is a definition which is an agent that extracts data from servers, process it, and passes further.

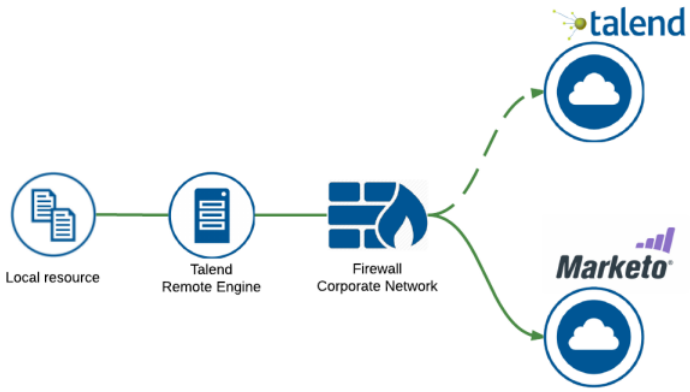


Figure 2.1 Local to cloud / cloud to local data transfer (source [10])

2.2.1.2 AWS Direct Connect from Amazon

"The AWS Direct Connect cloud service is the shortest path to your AWS resources. While in transit, your network traffic remains on the AWS global network and never touches the public internet. This reduces the chance of hitting bottlenecks or unexpected increases in latency. When creating a new connection, you can choose a hosted connection provided by an AWS Direct Connect Delivery Partner, or choose a dedicated connection from AWS—and deploy at over 100 AWS Direct Connect locations around the globe. With AWS Direct Connect SiteLink, you can send data between AWS Direct Connect locations to create private network connections between the offices and data centers in your global network." [11]

"AWS Direct Connect can reduce network costs, increase bandwidth throughput, and provide a more consistent network experience than internet-based connections. It uses industry-standard 802.1q VLANs to connect to Amazon VPC using private IP addresses. You can choose from an ecosystem of WAN service providers for integrating your AWS Direct Connect endpoint in an AWS Direct Connect location with your remote networks. AWS Direct Connect lets you establish 1 Gbps or 10 Gbps dedicated network connections (or multiple connections) between AWS networks and one of the AWS Direct Connect locations. You can also work with your provider to create sub-1G connection or use link aggregation group (LAG) to aggregate multiple 1 gigabit or 10 gigabit connections at a single AWS Direct Connect endpoint, allowing you to treat them as a single, managed connection.

A Direct Connect gateway is a globally available resource to enable connections to multiple Amazon VPCs across different regions or AWS accounts. This feature also allows you to connect to any participating VPCs from one private VIF, reducing AWS Direct Connect management, as shown in the following figure." [12]

In figure 2.2 we can see an example of AWS Direct Connect usage.

AWS Direct Connect works like VPN, so generally it is just about creating a secure tunnel for data transfer from one server to another through this application. For more information about VPN see [13]

2.2.1.3 On-Premise Data Gateway from Microsoft

"The on-premises data gateway acts as a bridge to provide quick and secure data transfer between on-premises data (data that isn't in the cloud) and several Microsoft cloud services. These cloud services include Power BI, PowerApps, Power Automate, Azure Analysis Services, and Azure Logic Apps. By using a gateway, organizations can keep databases and other data sources on their on-premises networks, yet securely use that on-premises data in cloud services" [14]

"When your Power App, Power Automate, or any of the other services needs data, it sends an

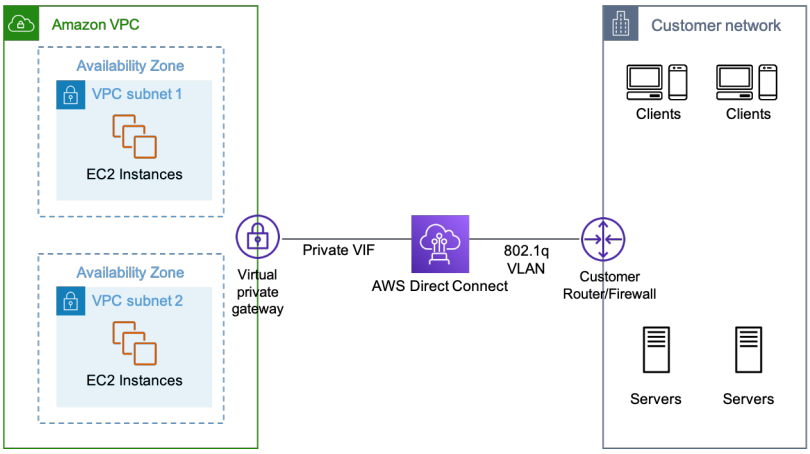


Figure 2.2 AWS Direct Connect usage example (source [12])

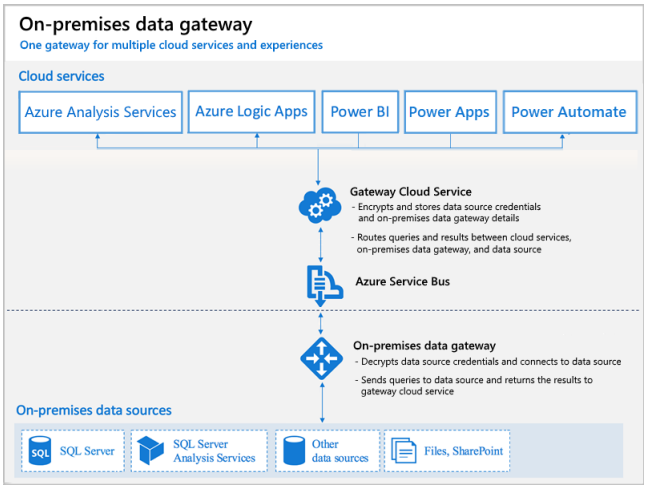


Figure 2.3 On-premises data gateway how it works (source [14])

encrypted request with credentials to the cloud gateway service. The gateway service then sends that encrypted request to your on-premises gateway. And your on-premises gateway decrypts the request, extracts the credentials, and connects to your data source with those credentials. It sends the query to the data source and, when it gets a response, sends the encrypted results back through the gateway service, which sends it to cloud service that requested the data in the first place." [15]

In figure 2.3 we can see how On-premises data gateway works.

On-Premise Data Gateway looks like to be Thin Agent. The Thin Agent is an Agent that extracts data from servers and sends it directly to cloud without any processing on the Agent side.

2.2.1.4 Azure Logic Apps Integration Service Environment from Microsoft

"Integration Service Environment (ISE) is a private and isolated Logic Apps instance within your Azure virtual network. The private instance uses dedicated resources such as storage and runs separately from the public global Logic Apps service." [16]

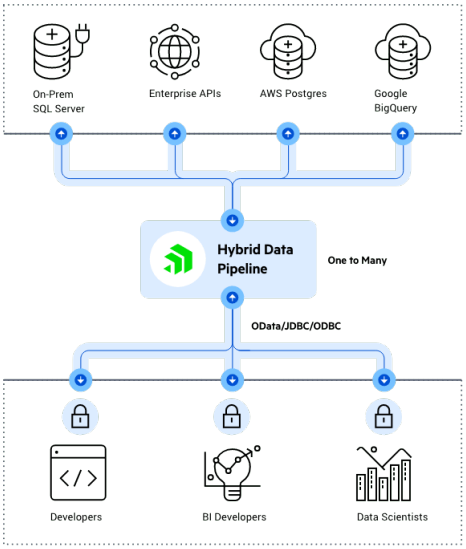


Figure 2.4 Diagram of hybrid data pipeline usage (source [17])

"One of the benefits of using an Integration Service Environment is the ability to select a Virtual Network (VNET) for your Azure Logic Apps deployment. This allows you to run Azure Logic Apps within an isolated network that is not exposed to the public internet." [16]

"What makes this capability interesting is that for organizations who have created a site to site VPN or Express Route connections between their data center and the cloud, these connections occur over a VNET. So the VNET capability that is now introduced, in Integration Service Environment (ISE), allows Azure Logic Apps to communicate with on-premises assets without the use of an on-premises data gateway, by using these VNETs." [16]

Azure Logic Apps Integration Service Environment seems to work as a VPN.

2.2.1.5 Hybrid Data Pipeline from Progress (product DataDirect)

Progress DataDirect is cloud and on-premises data connectivity solution across Relational, NoSQL, Big Data and SaaS data sources. It allows create a custom ODBC or JDBC driver, access Cloud and On-Premises data, embed connectivity in an application or platform. [17]

"With a full library of data connectors, Hybrid Data Pipeline brings manageability to your data with single sign-on identity based access controls for teams or individuals, universal client connectivity (ODBC, JDBC, OData, REST) and much more." [17]

In figure 2.4 we can see a usage of hybrid data pipeline.

Hybrid Data Pipeline seems to work as Thick Agent.

2.2.2 Possible solutions

This section describes all possible solutions. The solutions are listed below:

- VPN

  Redirect request from cloud to the target system. For more information about VPN see [13]

- Thin Agent

  As it was told before, the Thin Agent is an Agent that extracts data from servers and sends it directly to cloud without any processing on the Agent side.

- Thick Agent

  The Thick Agent is an Agent that containing extraction and parsing logic. The extraction will be performed on-premise and results will be sent back to the cloud.

- Hybrid Data Pipeline from Progress Software Corporation

  This solution is closest to the solution currently used by MANTA in CLI, but it seems it does not support all the technologies that are used in the CLI. The technologies like File Transfer, SDK and shell script execution are not supported by Data Pipeline solution.

  A partnership with Progress would mean making MANTA Cloud depending on third-party software, and also, the installation process should be discussed with the Progress company.

2.2.3 Conclusion

The agreed solution is a Thick Agent developed in-house by the MANTA team.

The Thick Agent allows us to prepare extracted metadata before sending them to the MANTA Flow and perform metadata parsing in simple cases.

The main reason why it was decided to develop the Thick Agent in-house is no need to depend on third-party organizations with such an essential component of the MANTA Flow tool. Such an approach will allow to development of the Agent tailored exactly to MANTA's needs. In case of problems with the Agent, the MANTA team will be responsible for fixing them so a fix can be delivered faster.

This Thick Agent will be responsible for extracting and transferring extracted data to MANTA Flow Cloud. For most extraction scenarios, the whole extraction will be done on the Agent side, and only extracted files are sent to data analysis in the cloud.

Database extraction is more complicated as the extracted data must be parsed first. It will be done in the cloud as it is memory consuming operation. The DDL parsing must be performed in the correct order. Extraction scenarios extract DDLs in this correct order, so MANTA Flow Agent must preserve this order during the transition to the cloud.

2.3 MANTA Flow

In this section we will take a closer look at MANTA Flow product, its structure and how to integrate the Agent with it.

2.3.1 MANTA Flow Scanning Process

This subsection is heavily based on the internal MANTA documentation.

MANTA defines a scan of a particular technical resource (such as a single database) as the overall process involving the capture, analysis, loading, and merging of lineage metadata into the MANTA Repository. Scans are typically run against a set of connections for a given technology (such as all Oracle databases), but there is a lot of flexibility regarding their execution, which can be invoked by shell script on the command line, REST API, or by using the graphical user interface. Sites generally determine (based on the needs of their users and the change dynamics of their code and metadata) how frequently to run their scans.

2.3.1.1 Phases of a Scan

There are following phases of the scan:

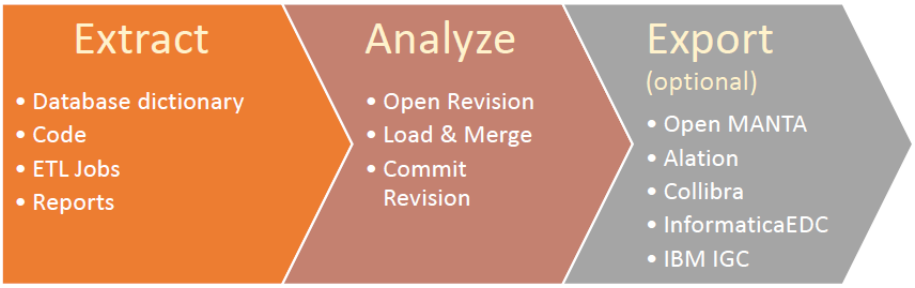


Figure 2.5 Phases of a Scan (source is internal MANTA documentation)

- Extract

  Metadata is extracted from the host system. Where possible, this is done via direct connection. This ensures that the metadata being retrieved is the most current and most relevant and reflects the truth of how the system is running today. This is usually done by APIs, but it depends on the technology. For most databases, this is done by directly accessing the database metadata tables. The extract phase first retrieves database dictionary information (primarily tables and columns) and then picks up assets that define lineage. These are typically views and stored procedures for a database but might be external data access steps and transformation modules when applied to an ETL (extraction/transformation/loading) tool. Business intelligence (reporting) solutions deliver lineage details about their queries along with columns in the report and their potential transformations. The extracted metadata is then ready for analysis.

- Analyze

  Each MANTA scanner is developed after extensive research into the selected technology. This research yields a deep understanding of how a particular syntax is derived, what constitutes a source or a target, and how data transformations are defined and stored. Specific dialects for certain languages are explored (such as the subtle differences between SQL-based solutions), and important structural issues are reviewed (a tool or technology's folder, project, and process structure). This allows the scanner, when implemented, to outline the exact flow of data through the selected tool or technology. MANTA strives with every scanner to provide detailed insight for data flows at the individual column or element level, while also capturing exact transformation syntax. During the analyze phase, the dictionary of column information (created earlier during extract) supports validation and proper column usage.

- Load-Merge

  This phase is where the analyzed metadata is put into the MANTA Repository. New metadata is loaded directly; refreshed metadata is compared to existing assets to detect changes in lineage. These changes can be reviewed later by users who require detailed lineage history. Further, lineage metadata is merged with other lineage metadata that is already in the repository. Names of columns used across multiple technologies are reconciled, allowing connections to be made across different technology types. This ensures the ability to trace the entire data pipeline and achieve end-to-end lineage.

We can see the phases of a scan in figure 2.5

Each phase of the scanning process described above is called a scenario. Workflows run sequentially and some scenarios can run in parallel. Each scenario can be launched for each configuration of the source system in parallel. Therefore, for example, a scenario that extracts DDL scripts from a database runs in parallel for each configured database. The details below describe the activities in each phase as they relate to specific technologies. This is all managed by

the MANTA Flow Process Manager, but it is described here to provide a deeper understanding of what is occurring as MANTA retrieves and analyzes your lineage metadata.

At the beginning of the analytic phase, a new revision scenario is called, which creates a new revision in the internal metadata repository so it is ready to accept new metadata.

For the export phase, there are several scenarios. Each is used independently, depending on the requirements.

2.3.1.2 Revision

Scans are typically sequenced as part of a full MANTA run. A full MANTA run is a coordinated execution of all important scans. The metadata for these scans and their resulting lineage are grouped together in a single revision. Each revision represents a snapshot in time of all the coordinated metadata in the corporate systems and technologies used in the enterprise. Users will see the most recent or current revision by default, but can easily request lineage or review of an asset at an earlier point in time, supporting any kind of historical lineage research or comparisons.

2.3.1.3 Export

MANTA export feature is the ability to run a specific (licensed by MANTA) technology export or a more generic export. These export capabilities support third-party solutions or other home-grown solutions where MANTA lineage metadata is loaded into other complementary systems. Quite often, these are data catalog solutions, but there are many other examples where MANTA lineage metadata is used to augment other applications. The export phase, once configured, obtains selected MANTA metadata and packages it for shipment to a set of CSV files (in the generic case). Some types of exports also may automatically put them into proprietary API calls (usually a REST API payload in XML or JSON) for a third party and makes the calls on behalf of the MANTA user or site. This loads the lineage metadata into the other application, where it can be used for additional purposes.

2.3.2 MANTA Flow structure

In this section we will take a closer look at the MANTA Flow structure. MANTA Flow consists of multiple applications working together to deliver its functionality. In figure 3.1 we can see the MANTA Flow architecture.

"MANTA Flow is the application that actually gathers all the metadata to reconstruct complete lineage. MANTA Flow is made up of three major components:

- *MANTA Flow CLI is a Java command-line application that extracts all scripts from source databases and repositories, analyzes them, sends all gathered metadata to MANTA Flow Server, and optionally, processes and uploads the generated export to a target metadata database.*
- *MANTA Flow Server is a Java server application that stores all gathered metadata in its metadata repository, transforms it into a format suitable for export to a target metadata database, and provides it to its visualization or third-party applications via API.*
- *MANTA Admin UI is a Java server application providing a graphical and programming interface for the installation, configuration, update, and overall maintenance of MANTA Flow."*

[18]

Currently, the "extracts all scripts from source databases and repositories" part is performed by Manta Flow CLI component. This functionality should be moved to the newly created

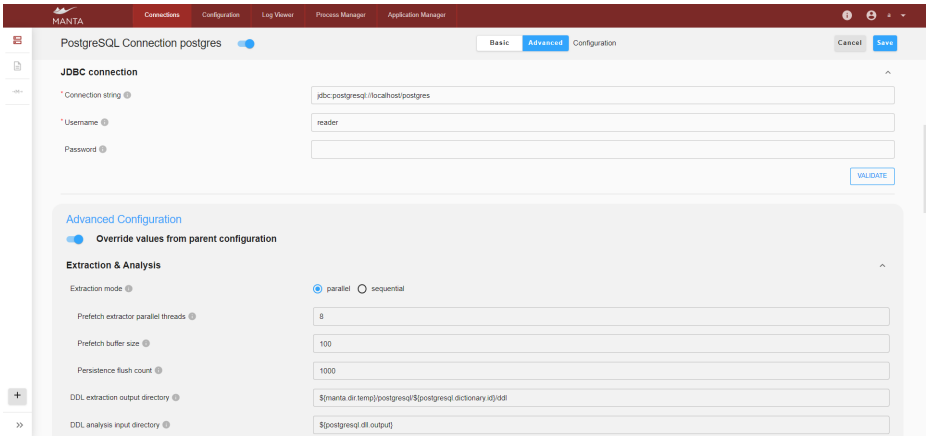


Figure 2.6 PostgreSQL connection configuration in Admin UI

MANTA Flow Agent component (henceforth we will call it just an Agent). The Agent will be the main result of this master thesis.

2.3.3 Extraction phase flow

The trickiest part of the Thick Agent implementation is its integration into the current structure of MANTA. Extraction that MANTA Flow CLI currently performs, should be moved entirely to the Agent.

The extraction phase may be divided into the following steps: extraction configuration, launch extraction, extraction, and collect extracted metadata. The extraction step is divided into metadata extraction and parsing/processing.

2.3.3.1 Configure extraction

The configuration of each connection to source systems that should be analyzed is called connection. Each connection configuration consists of two parts: connection-specific configuration and the common configuration that is general for all connections. The connection and the common configurations may be configured by the configurator application, which is a part of Admin UI.

In figure 2.6 and 2.7 we can see an example of PostgreSQL connection and common configuration in Admin UI.

2.3.3.2 Launch extraction

Extraction is launched by the Process manager in Admin UI. The Process Manager works with workflows and scenarios. The source of information about the Process Manager in Admin UI is Orchestration and Monitoring of Manta Flow Processes master thesis [19]. For more information about the Process Manager in Admin UI see the thesis.

Workflow is a list of Phases, Technologies, Connections and Scenarios. It defines which Scenarios should be executed for which Phase, Technology and/or Connection, regardless of their Orchestration Constraints. The Workflow also contains metadata about itself, e.g. creation time and the creator or configurable environment variables.

Scenario is an execution unit for a Workflow. The Scenario may have Orchestration Constraints. In MANTA Flow CLI Scenario is represented by a Spring bean responsible for a step of MANTA Extract, Analyse and Export process.

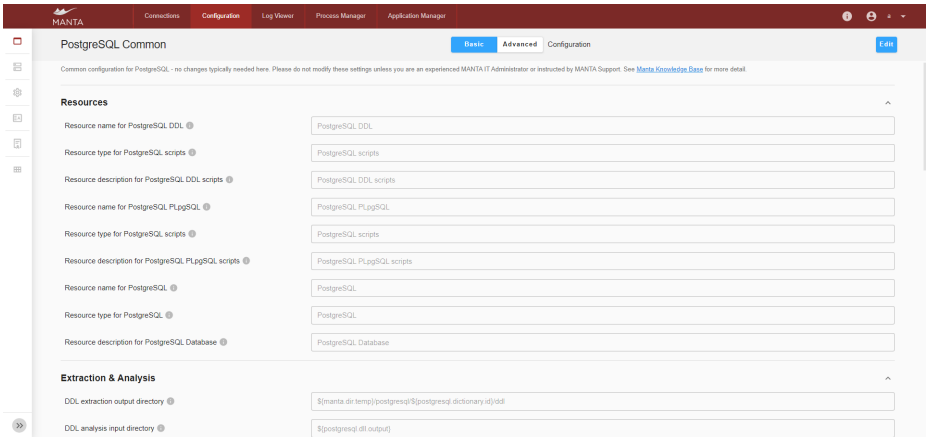


Figure 2.7 PostgreSQL common configuration in Admin UI

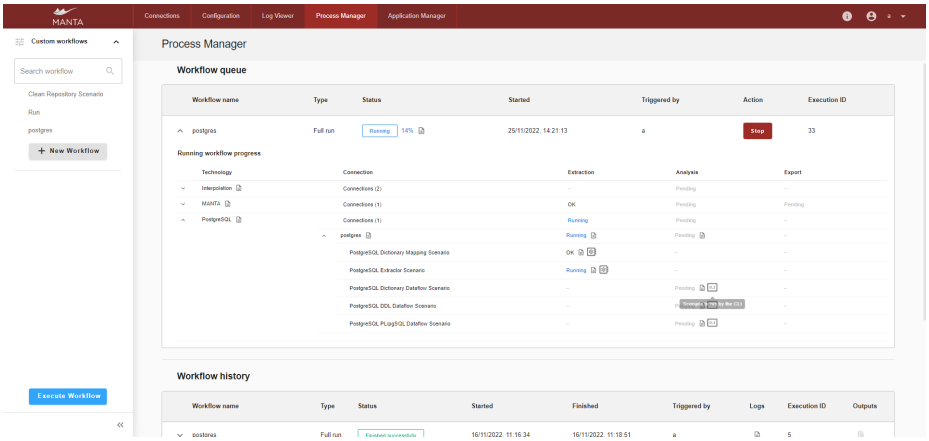


Figure 2.8 Main screen of the Process Manager in Admin UI

In figure 2.8 we can see the main screen of the Process Manager with workflow queue, workflow history, and the list of defined workflows. In figure 2.9 we can see a workflow editor in the Process Manager, which enables us to create new workflows and edit existing workflows.

2.3.3.3 Extraction

The extraction step for some technologies is divided into metadata extraction and parsing/processing substeps.

Metadata extraction The extractor scenario connects to the source system defined in the connection configuration and extracts metadata. Based on the source system, different metadata may be extracted.

For instance, the Microsoft SQL Server extraction scenario doing the following during the extraction:

- Extract names of all objects and their dependencies
- Topologically sort them
- For each object

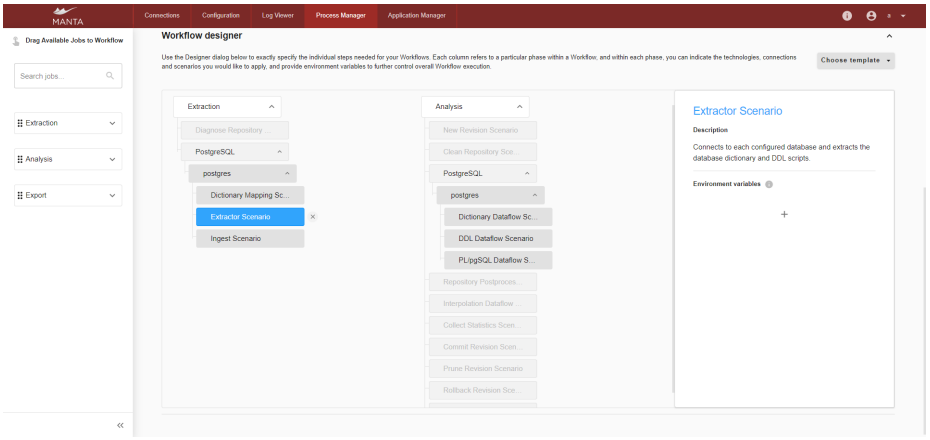


Figure 2.9 Process Manager workflow editor in Admin UI

- Extract DDL
- Write DDL to file (if requested)
- Parse DDL and write resulting object metadata to dictionary

Metadata parsing/processing Metadata parsing is required for some technologies before passing them to the analysis scenario. Mainly, parsing is required for database or programming language technologies. Mostly parsing or processing of metadata before analysis does not take a lot time and resources, but for some technologies it does. The technologies that are supported by MANTA and that takes a lot of resources and time are Microsoft SQL Server, Apache Hive and Oracle database.

2.3.3.4 Collect extracted files

All extraction outputs at the end of the extraction scenario are persisted in the temp folder in the MANTA Flow CLI temp directory. The extraction outputs are persisted in a particular format and structure, which expects an analysis scenario corresponding to the extractor scenario. For instance, the PostgreSQL extraction scenario persists extracted metadata in the format and structure expected by the following up PostgreSQL analysis scenario.

For instance, the directory structure of extracted PostgreSQL metadata stored in MANTA Flow CLI temp directory is the following:

```
temp
└── postgresql
    └── <connection id>
        ├── ddl
        │   └── <database name>
        │       └── <schema name>
        │           ├── views
        │           │   ├── <view name>.sql
        │           │   └── ...
        │           ├── functions
        │           │   ├── <function name>.sql
        │           │   └── ...
        │           └── <entity type>
        │               └── ...
        └── <connection id>.mv.db
```

```
    <connection id>
      ...
```

2.4 Used technologies

In this section are listed some of the technologies that were used during this diploma thesis writing.

2.4.1 Integrated development environment

An integrated development environment (IDE) is software for building applications that combines common developer tools into a single graphical user interface (GUI). An IDE typically consists of:

- *Source code editor*

  A text editor that can assist in writing software code with features such as syntax highlighting with visual cues, providing language specific auto-completion, and checking for bugs as code is being written.

- *Local build automation*

  Utilities that automate simple, repeatable tasks as part of creating a local build of the software for use by the developer, like compiling computer source code into binary code, packaging binary code, and running automated tests.

- *Debugger*

  A program for testing other programs that can graphically display the location of a bug in the original code.

[20]

IntelliJ IDEA Ultimate IDE by JetBrains was used during the development.

2.4.2 Java programming language

Java is a high-level, class-based, object-oriented programming language. For many years it was one of the most popular programming languages. It is used quite often by big companies like banks. [21]

2.4.3 Maven

"Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal, Maven deals with several areas of concern:

- *Making the build process easy*
- *Providing a uniform build system*
- *Providing quality project information*
- *Encouraging better development practices"*

[22]

2.4.4 JUnit

JUnit is a unit testing framework for the Java programming language. [23]

2.4.5 Spring Boot

"Spring Boot helps you to create Spring-powered, production-grade applications and services with absolute minimum fuss. It takes an opinionated view of the Spring platform so that new and existing users can quickly get to the bits they need.

You can use Spring Boot to create stand-alone Java applications that can be started using java -jar or more traditional WAR deployments. We also provide a command-line tool that runs Spring scripts.

Our primary goals are:

- *Provide a radically faster and widely accessible getting started experience for all Spring development.*
- *Be opinionated, but get out of the way quickly as requirements start to diverge from the defaults.*
- *Provide a range of non-functional features common to large classes of projects (for example, embedded servers, security, metrics, health checks, externalized configuration).*
- *Absolutely no code generation and no requirement for XML configuration."*

[24]

2.4.6 ActiveMQ Artemis

"Apache ActiveMQ Artemis is an open source project to build a multi-protocol, embeddable, very high performance, clustered, asynchronous messaging system. Apache ActiveMQ Artemis is an example of Message Oriented Middleware (MoM)." [25]

Artemis was selected as a message broker as it is able to transfer large files through it and it is already used in MANTA platform. For more information, see [26].

2.4.7 JavaScript programming language

JavaScript is a scripting language that enables to create dynamically updating content, control multimedia, animate images, and implement complex features on web pages. [27]

2.4.8 React

"React is a JavaScript library for building user interfaces.

- *Declarative*

  React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes. Declarative views make your code more predictable, simpler to understand, and easier to debug.

- *Component-Based*

  Build encapsulated components that manage their own state, then compose them to make complex UIs. Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep the state out of the DOM.

- *Learn Once, Write Anywhere*

  We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code. React can also render on the server using Node and power mobile apps using React Native."

  [28]

2.4.9 Liquibase

"Liquibase is an open-source database schema change management solution which enables to manage revisions of your database changes easily. Liquibase makes it easy for anyone involved in the application release process to:

- *Eliminate errors and delays when releasing databases*
- *Deploy and roll back changes for specific versions without needing to know what has already been deployed*
- *Deploy database and application changes together so they always stay in sync"*

  [29]

2.4.10 H2 database

H2 is a relational database management system written in Java. It can be embedded in Java applications or run in client-server mode. It is really good, when you need a small embedded database. H2 database may by running in two different modes: embedded and server. H2 database has small footprint: around 2.5 MB jar file size [30]

2.4.11 MyBatis 3

"MyBatis is a persistence framework with support for custom SQL, stored procedures and advanced mappings. MyBatis eliminates almost all of the JDBC code and manual setting of parameters and retrieval of results. MyBatis can use simple XML or Annotations for configuration and map primitives, Map interfaces and Java POJOs (Plain Old Java Objects) to database records." [31]

2.4.12 VMware InstallBuilder

"VMware InstallBuilder is a multiplatform installer development and automatic update tool that makes it easy to package and deliver updates to crossplatform software." [32]

Chapter 3
Design

In this chapter, we will talk about the design. The chapter is divided into a number of parts where we will look at a particular part of the design in each part. The chapter is divided into Agent integration to MANTA Flow, Agent design, Admin UI integration design and Extraction processor design.

3.1 Agent integration to MANTA Flow

Two new applications should be created; namely, Agent and Extraction processor, and certain changes should be made to the Admin UI to move extraction functionality from MANTA Flow CLI to the Agent. Also, the Agent Manager component should be added to the Admin UI.

All connection configurations for all technologies need to be modified to allow the selection of the extraction method that is used for metadata extraction. If the Agent extraction method is selected, it has to be possible to select an Agent that will perform the extraction.

Based on the extraction method selected in the connection configuration, the Process Manager should launch extraction through the Agent or CLI. If the Agent extraction method is selected, then the extraction scenario will be launched on the corresponding Agent.

The Agent will receive a command to perform an extraction scenario together with the connection configuration of the source system from which the metadata should be extracted. If required for the extraction, parsing and preparations must be done before sending extracted metadata for the following persisting.

MANTA Flow CLI persists extracted outputs directly to the file system within extraction scenarios. The following analysis scenario, which the MANTA Flow CLI also performs, analyses the previously persisted extracted outputs. When MANTA Flow CLI performs the extraction, there are no problems with persisting the extraction outputs to the place where the following extraction scenario can reach them. It is so, as extraction and analysis are performed on the same server and by the same application. However, when the Agent performs the extraction, the extracted output should be persisted in the place where the MANTA Flow CLI can reach them before the analysis scenario is started. A new service called Extraction Processor should be introduced, which will receive the extracted outputs and persist them in the format and place required by the extractions scenarios and MANTA Flow CLI. As some of the technologies require more resources and time, their parsing will be moved to the Extraction Processor to reduce resource requirements for the Agent. The parsing of the rest of the technologies that do not require resource-intensive parsing will be performed on the Agent. The extraction Processor will consist of two functional parts, Extraction Persistor and Extraction Parser. Extraction Processor will persist the outputs extracted by Agent. The extraction Parser will parse the

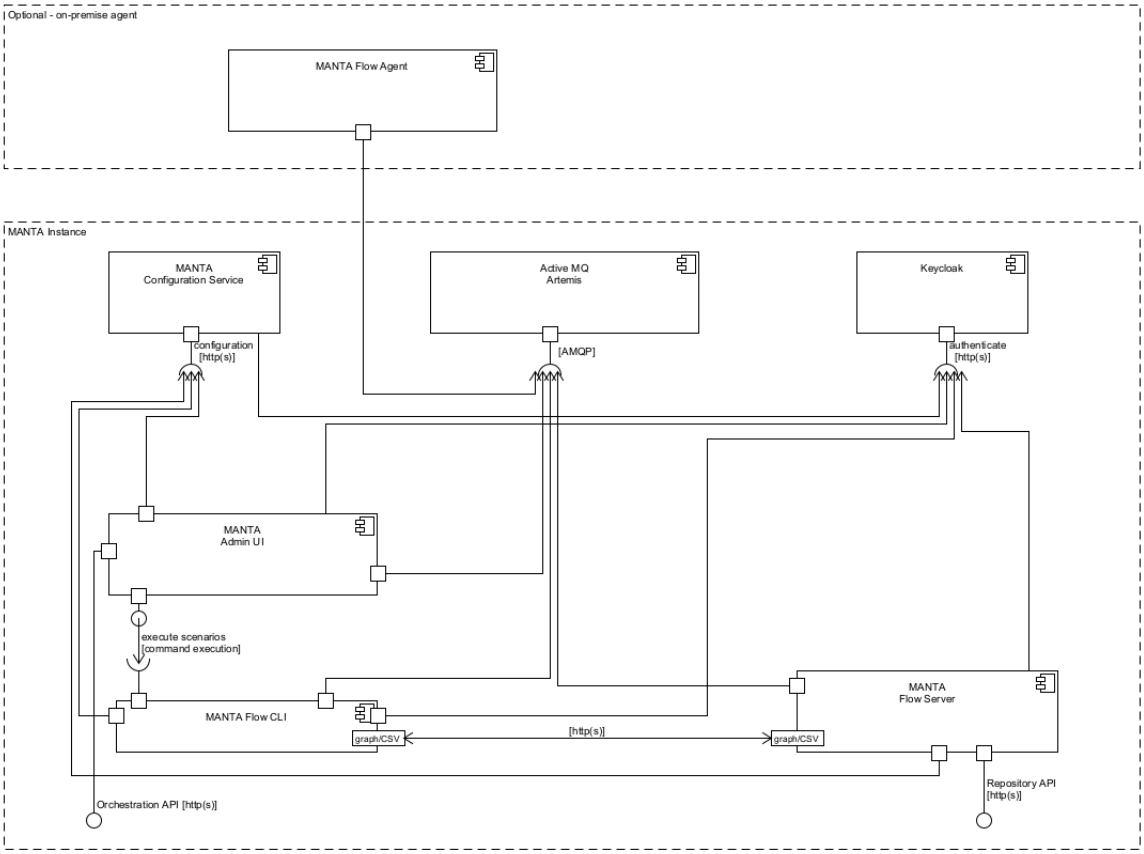


Figure 3.1 MANTA Flow architecture (source is internal MANTA documentation)

outputs extracted by the Agent and pass the parsing results to the Extraction Persistor for the following persistence.

In Admin UI, the new Agent Manager component should be added to manage all Agents that can perform extraction scenarios. The Agent Manager should enable registering new Agents, see Agent status, update Agent remotely, and configure it.

The MANTA Flow architecture with the MANTA Flow Agent integrated in it may be seen in in figure 3.1.

Sequence diagram of metadata extraction with Agent may be seen in in figure 3.2.

3.2 Sharing common code across applications

In this section we will take a look at how to share common code across different applications to reduce code duplicity as much as possible. Mostly, the code that should be shared is data transfer objects.

Sources of information on which the decision was made [33], [34] and [35].

There are several options how to store DTOs:

- Store DTO copies both in Agent and in Admin UI
- Store DTOs in a separate module, which will be shared between Agent and Admin UI
- Store DTOs in the client libraries

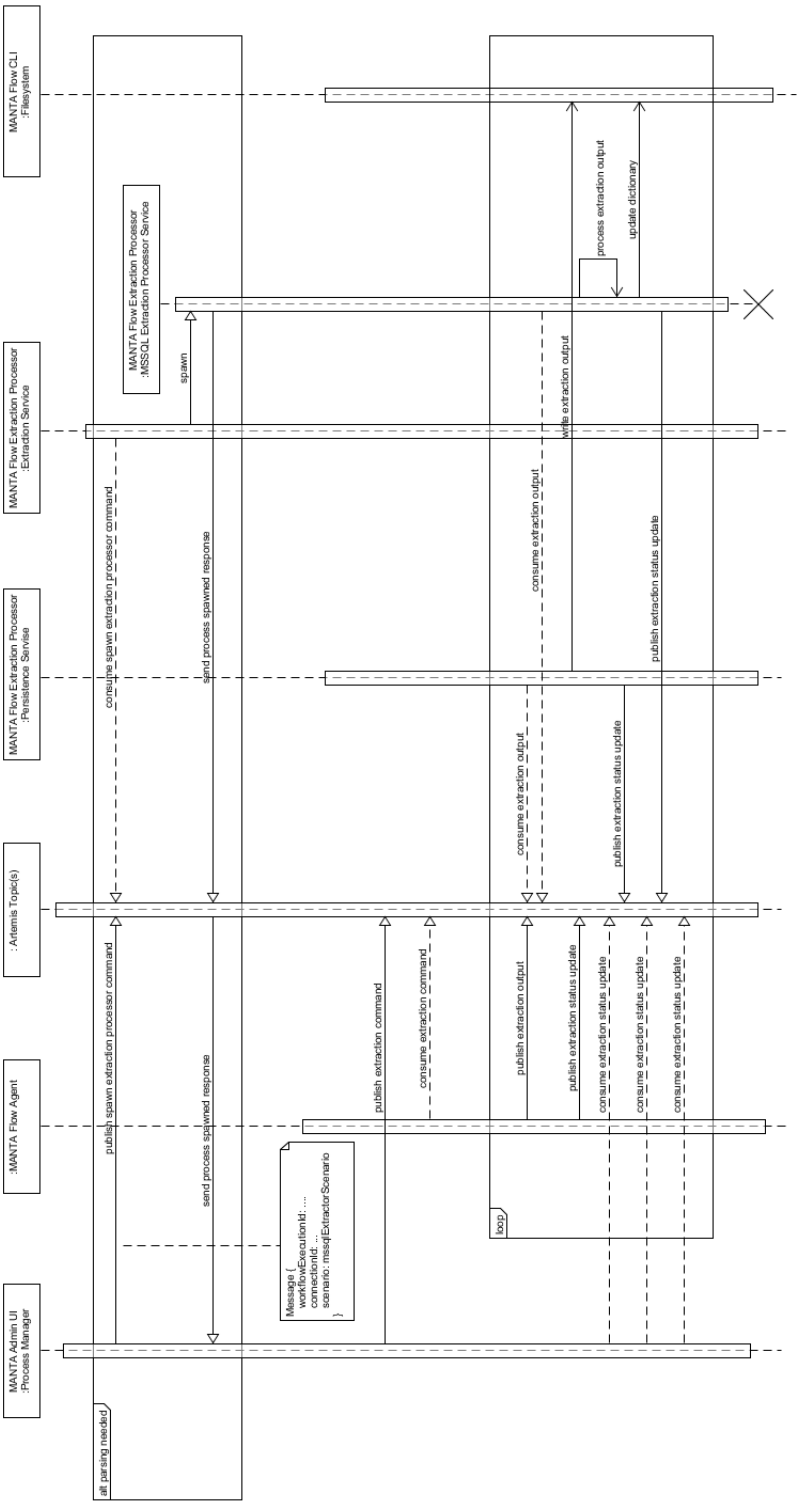


Figure 3.2 Sequence diagram of metadata extraction with Agent (source is internal MANTA documentation)

Below, we will discuss each option.

3.2.1 DTO copies in Admin UI and Agent

Mostly the duplicate copy-pasted code, which may differ across different versions of Admin UI and Agent.

Advantages

- Allows to avoid coupling

Disadvantages

- It leads to redundant code maintenance.
- It will be necessary to provide a mapping between DTOs in Admin UI and Agent, what may cause some problems (e.g., typos in names of DTOs, lack of DTO on one side, and presence of it on the other)
- To perform a validation on both producer and consumer sides, the validation also should be copy pasted.

3.2.2 DTOs in a separate module which will be shared between Agent and Admin UI

DTOs located in one separate module. It will be thin as there will be only DTOs without any logic.

Advantages

- No redundant code
- If breaking changes are not introduced, different versions of modules (client libraries) may be used
- Contract-breaking changes can be communicated via the library version easily
- Validation may be performed on the producer and consumer sides without the need for copy-pasting code

Disadvantages

- This may cause overly coupling a microservice and consumers such that any small change to the microservice itself can cause unnecessary changes to the consumer. For example, at one client, we had a library of common domain objects that represented the core entities in use in our system. This library was used by all the services we had. But when a change was made to one of them, all services had to be updated.
- Leads to model generalization across all the services (what is not a problem in our case, Admin UI and Agent is developed by the same team)
- This leads to model generalization across all the services, but it is not a problem in our case, as Admin UI and Agent are developed by the same team of developers.

3.2.3 DTOs in client libraries

Agent and Admin UI are separated into Client and Server parts. DTOs located in client parts of Admin UI and Agent, reused both by producer and consumer. It will be thin as there will be only DTOs without any logic.

Advantages

- No redundant code
- If breaking changes are not introduced, different versions of modules (client libraries) may be used
- Contract-breaking changes can be communicated via the library version easily
- Validation may be performed on the producer and consumer sides without the need for copy-pasting code
- Separation of DTOs into two client modules (separation of concerns), so specific DTOs for specific services are used

Disadvantages

- This may cause overly coupling a microservice and consumers such that any small change to the microservice itself can cause unnecessary changes to the consumer. For example, at one client, we had a library of common domain objects that represented the core entities in use in our system. This library was used by all the services we had. But when a change was made to one of them, all services had to be updated. But here the problem is solvable. The service doing the breaking changes must support both versions of the API, until all services dependent on it upgrade. Upgrading those dependent services is desirable, but not forced, unlike a shared artifact.

3.2.4 Conclusion

DTOs stored in client libraries should be used as they have the biggest number of advantages and the only disadvantage may be solved by making use of client libraries not mandatory, but hardly recommended.

In case of need, copy-paste method may be used. To make such flexibility possible, we will have to prepare consumers for situations when the path to class is not known and the class name is the same as the name of the copy-pasted class.

Most likely for message serialization/deserialization MappingJackson2MessageConverter (Spring Framework 6.0.2 API) will be used. Type of serialized object is sent as message property (the default property name is \type).

3.3 Agent design

In this section we will take a look at the MANTA Flow Agent design.

The Agent application can be logically separated into several logical and functional parts: Agent Dispatcher, Agent Extractor, and Agent Validator.

3.3.1 Extraction as process or thread?

One crucial moment to decide before implementing the Agent is how extractions and validations will run. If each extraction and validation should be running as a separate thread or a separate process.

3.3.1.1 Extraction as process

If extraction scenarios and validations run as a separate process, then Agent Dispatcher is the main Agent process that always runs and spawns Agent Extractor and Agent Validator processes when extraction or, respectively, validation requests are received from the Process Manager.

Advantages

- Easy to kill Agent Extractor process from Agent Dispatcher main process
- If some Agent Extractor JVM process is crushed, all other extraction processes will keep working
- Better resources assignment. It is possible to set limits by JVM command line run flags on the new process start.

Disadvantages

- Inter-process communication (e.g., Java RMI, JMX) is needed to pass messages from the Agent Dispatcher to the Agent Extractor JVM process. It is no problem if the Agent Extractor process is launched from Agent and then it directly communicates with MANTA Cloud if needed, but most probably it would not be needed. Where it is needed, JMS is used.
- If Agent JVM is killed, then it will be tricky to kill children processes. It may be solved by passing Agent PID (process identifier) to the child Extraction process and periodical checking if the parent Agent Dispatcher process is still running. If it is not running, then the extraction process will immediately kill itself.

Possible implementation

- Extraction running

  Each extraction is running in a separate process.

- Extraction launch

  The Agent Extractor process is launched from the Agent Dispatcher process. The Agent Dispatcher will handle PIDs (Process identifiers) of all launched extractions so that it can kill them in case of need. All children's processes will have the PID of the parent Agent process, so they can check if the parent is still alive, and if not, they will kill themself.

- Extraction configuration

  Extraction and Artemis connection configurations are passed to the extractor process as input arguments. The input argument may be a string with the whole configuration, environment variable, or path to file with configuration.

- Communication from Agent Extractor process to MANTA Cloud

  The extractor process sends extracted objects directly to MANTA Cloud to queue for exported objects, and there is no need to communicate with the Agent Dispatcher process. Each Agent Extractor will create its own connection to MANTA Cloud.

- Communication from MANTA Cloud process to Agent Extractor

  The first approach is that messages are listened to in the Agent Dispatcher process and, after, are passed to Agent Extractor processes using inter-process communication. Another possible approach is that messages are listened to directly in extraction processes, and then there is no need for inter-process communication. Unique extraction IDs and different message types may be used to filter messages for Agent and Extractor.

3.3.1.2 Extraction as thread

If extraction scenarios and validations run as a separate thread, then Agent Dispatcher is the main Agent process that always runs and spawns Agent Extractor and Agent Validator processes when extraction or, respectively, validation requests are received from the Process Manager.

Advantages

- Easy to pass configuration for extraction and commands to extraction thread
- If to use singleton beans as much as possible, all extractors may share the same beans, so it is possible to reuse network connections to Activemq Artemis.
- If the Agent process is killed, then all running threads are together with it. So there is no need to clean up leftover processes.

Disadvantages

- Proper stopping is problematic, as all resources should be closed and the thread should be finished properly, otherwise, memory leaks are possible.
- There is no granularity in resource monitoring. All running extractors and the Dispatcher will be hidden under the same process, so it would be more difficult to distinguish which resources are used by which application.

Possible implementation

- Extraction running

  Each extraction is running in a separate thread.

- Extraction launch

  The agent listens to messages from Cloud and dispatches each extraction task to a separate thread.

- Extraction configuration

  Extraction configuration is passed from the Agent Dispatcher to the Agent Extraction thread directly within one application.

- Communication from Extractor thread to MANTA Cloud

  The extractor thread sends extracted objects directly to MANTA Cloud, using the Agent service for communication with MANTA Cloud so that all extractors will use the same service, thus, the same connections to MANTA Cloud.

- Communication from MANTA Cloud to Extractor thread

  Messages are listened to in the Agent Dispatcher and then dispatched to extractor threads.

3.3.1.3 Conclusion

Given the above comparison, it is better that each extraction be a separate process. This option has significant advantages and no unresolvable disadvantages, while an option with a separate thread for each extraction does not have any significant advantages and has one significant disadvantage.

3.3.2 Agent common moments

In this subsection, we will discuss common moments for all Agent applications.

3.3.2.1 Communication

All communication from the MANTA Flow application to the Agent is performed through ActiveMQ Artemis. Such an approach eliminates the need to perform requests directly to the Agent and to have an opened input port on the server where the Agent is installed. Mutual Transport Layer Security secures communication with ActiveMQ Artemis, which allows for encrypted connection in which both parties authenticate each other using certificates. For more information on this topic, see [36].

Messaging configuration We use Spring functionality to work with JMS Messaging. Based on the message type, the appropriate method in CommandListener is called.

Message sending to Artemis is performed using ArtemisCommunicationService, which contains preconfigured JmsTemplate and provides maximum flexibility for message sending. JmsTemplate is configured to use the CachingConnectionFactory factory, which boosts performance.

DefaultJmsListenerContainerFactory should be provided for the proper work of JmsListeners. It should be initialized with the CachingConnection factory for a performance boost. If this factory is used, multiple sessions will be created within one connection, which decreases useless overhead. Using the CachingConnection factory with durable consumers is required, as each consumer should have a unique id. ID is assigned for each connection.

Reusing connections/sessions/consumers/producers as much as possible is essential. Otherwise, it may cause performance issues. Spring JMS Template is known to use resources poorly, so CachingConnectionFactory factory should be used with it.

JmsListener, which consumes messages with high priority, should always have available threads which may handle them. We want to avoid situations when all threads handle less critical messages and have no capacity for important messages. It may be solved with a concurrency property with JmsListener.

Messages All messages to/from Agent should be uniquely identified by ID to make it possible to track them across systems.

As messages will be consumed both from Extractor and Agent, they should be filtered using the following message properties: agentId, extractionId, and message type. (see Expression)

MappingJackson2MessageConverter may be used to convert an object to a message with a serialized object in JSON format. Information about the serialized object is added to message properties. Using it, a consumer will be possible to deserialize the object correctly. The information required for deserialization is a full path to a class that was serialized. It is stored in \type property. DTOs are stored in client libraries, and this functionality is not used, but it may be used if needed.

3.3.2.2 Agent installation structure

Agent structure after installation is the following:

```
agent
├── bin
│   ├── <agent version>
│   │   └── service
├── lib
│   ├── <agent version>
├── manta-flow-agent-dir
│   ├── conf
│   ├── data
│   ├── extraction-outputs
│   ├── lib-ext
│   ├── logs
│   └── tmp
└── tools
    └── <agent version>
```

The Agent distribution model is on-premise. The Agent should be installed on the client server to access the servers running in the client network. For now, Agent is distributed as Agent wrapped with an installer created with InstallBuilder software, but in the future, it also will be distributed as a docker image. To support more distribution formats in the future, the Agent folder structure should be better prepared for it right from the beginning. To have the Agent prepared, its structure should be basically separated into two categories: files that should not be changed, so they should not be persisted, and files that can be changed, so they should be persisted. Persisted means those files should be kept on the Agent restart and update, while others can be deleted or entirely replaced by new files. The manta-flow-agent-dir directory contains all files that should be persisted, when all other directories should not and can be safely deleted or replaced.

Versioned bin, lib, and tools directories are needed for remote upgrade implementation. During the remote update, a new version of Agent is installed right next to the old version of Agent, and when the installation of the new Agent is installed, the old Agent launches the new Agent, which will clean up leftovers of the old Agent. It will be discussed later in the section about remote Agent update 3.3.9.2.

3.3.2.3 Agent configuration

The Agent application configuration is separated on user and default configurations. The default configuration can be overwritten with user configuration. The default configuration will be stored inside Agent jars and will not be accessible from the outside for any changes. The user configuration will be

Configurations are stored in the configuration in YAML format. YAML is a human-readable data format that is often used for configuration files. Installer Builder does not work well with .properties files, but it has good support for YAML format, so YAML format is preferred. Spring supports YAML configuration files just fine, so it also would not be a problem.

Each process acquires properties from the configuration files on its startup. If to store extraction configurations on the Agent side, then to edit it, we will have to request configuration from the Agent, edit it and then send it back. So, to avoid it, as many as possible configurations required for extraction should be sent to the Agent together with the extraction command to prevent unnecessary complications.

The default Agent configuration stored inside of Agent JAR and not accessible from the outside is the following:

```
manta:
  dir:
    scenario: ${manta.agent.root-path}/manta-flow-agent-dir/data
```

```
    input: ${manta.agent.root-path}/manta-flow-agent-dir/data/input
    temp: ${manta.agent.tmp-path}
  artemis:
    connection-enabled: false
    port: 61616
    host: localhost
    mtls-enabled: false
    keystore-path:
    keystore-password:
    truststore-path:
    truststore-password:
  agent:
    common:
      id: Agent_default
      version: ${project.version}
    tmp-path: ${manta.agent.root-path}/manta-flow-agent-dir/tmp
    ext-lib-directory-path: ${manta.agent.root-path}/manta-flow-agent-dir/lib-ext
    jar-path: ${manta.agent.root-path}/lib/${project.version}
    extraction-file:
      tmp-folder: ${manta.agent.root-path}/manta-flow-agent-dir/tmp/extraction_file

logging:
  config: classpath:log4j2-spring.xml

spring:
  pid:
    file: ${manta.agent.root-path}/manta-flow-agent-dir/data/agent.pid
```

An example of the user configuration, that is set and created during Agent installation, is the following:

```
manta:
  artemis:
    port: 61616
    host: localhost
    mtls-enabled: true
    keystore-path: agent/manta-flow-agent-dir/conf/artemiskeystore
    keystore-password: really_secure_password
    truststore-path: agent/manta-flow-agent-dir/conf/artemistruststore
    truststore-password: another_secure_password
  agent:
    common:
      id: Agent_default
```

The user configuration supersedes the default configuration. Separation of configuration on default and user allows for simplifying the Agent upgrade process. The default configuration is entirely replaced on each upgrade when the user configuration is kept as it is. The default configuration contains reasonable defaults for all parameters, but for some parameters, it is only possible to decide with input provided by a user. Also, separation on user and default configurations is needed for future distribution of Agent as a docker image as it allows to upgrade Agent in a container by replacing default configuration and Agent files.

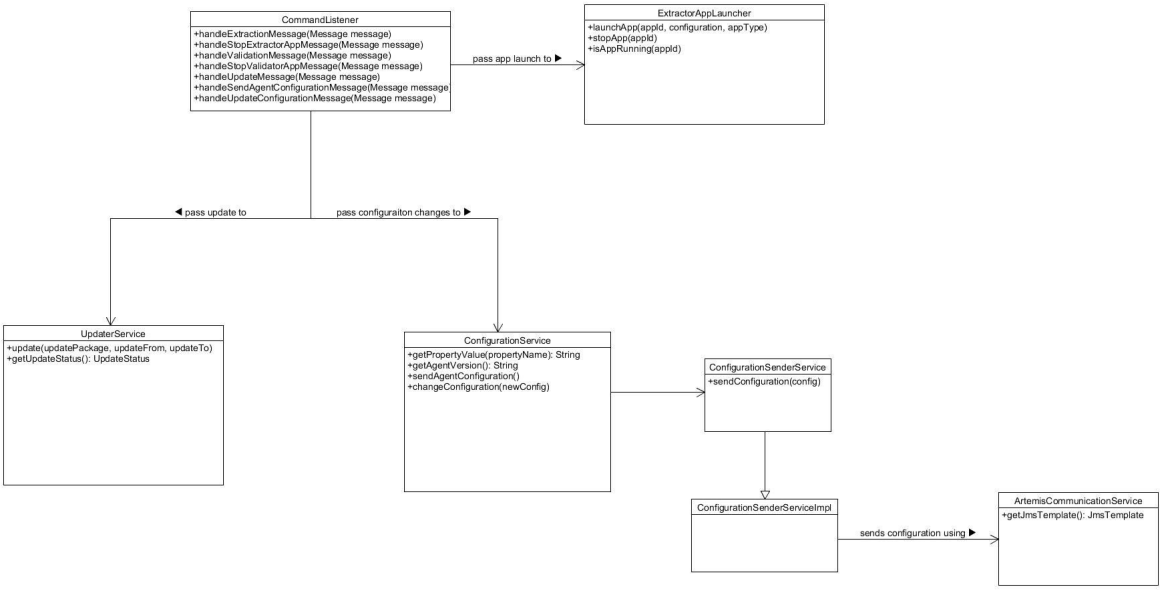


Figure 3.3 MANTA Flow Agent Dispatcher design

3.3.3 Agent Dispatcher

Agent dispatcher serves as a dispatcher that spawns application processes and dispatches tasks to them. The Agent dispatcher is the only Agent application that is running all the time when the Agent is running. It listens to requests about Agent status and responds to them.

In figure 3.3 we can see the Agent Dispatcher design.

3.3.4 Spawned processes abort

After launching the extraction process, the dispatcher will store the PID of the process. Using this PID, it would be possible to kill the process later if the dispatcher receives a terminate extraction command.

If the dispatcher process is stopped, then all spawned processes should also be stopped. All children's processes will validate if the parent process is running. If the parent process is not running, the children's process will be self-destroyed.

3.3.5 Agent Extractor

Agent extractor performs extraction from source system based on configuration that was passed to it from Agent dispatcher.

Each extractor produces different types of outputs like files, different objects. Outputs will be unified (file, string), so we can use general implementations of extractor persister which may be used from any extractor instead of writing different Extraction Persister for each Extractor.

All extractions are already implemented in MANTA Flow CLI and they just should be moved to the Agent and used from it.

In figure 3.4 we can see Agent Extractor design.

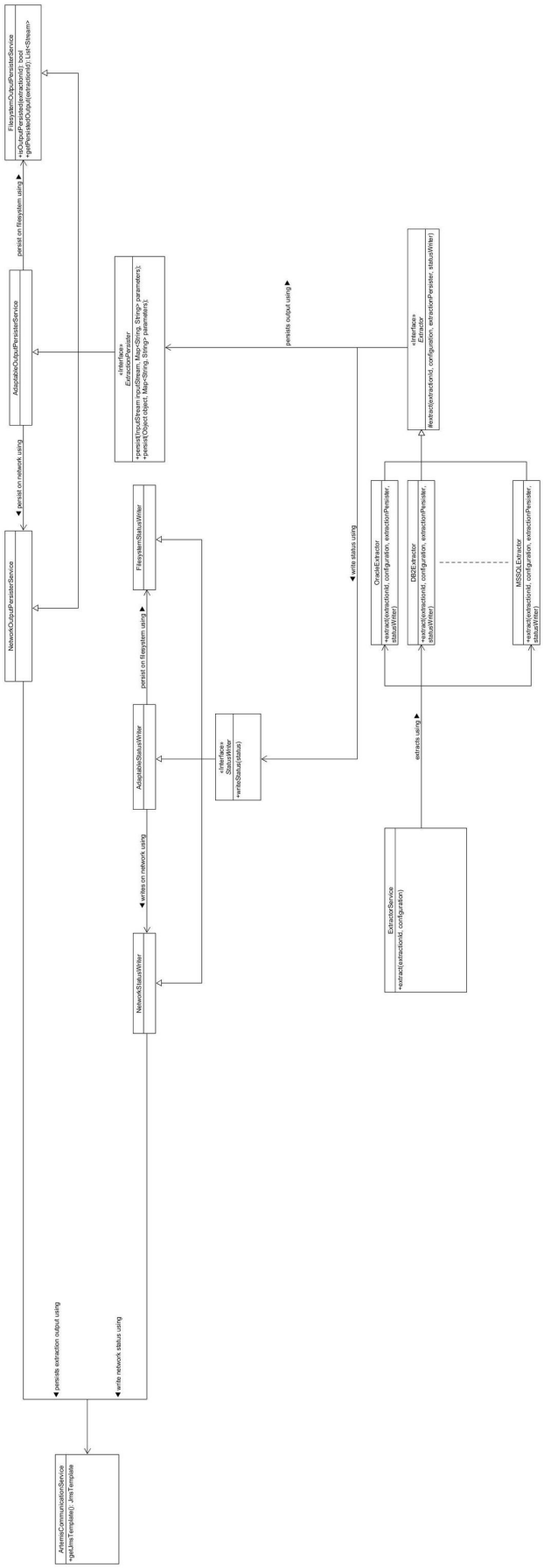


Figure 3.4 MANTA Flow Agent Extractor design

3.3.5.1 Persisting extracted outputs

Extracted outputs are persisted by output persister, which has a different implementation. The used implementation is configured and received during each extraction command together with the extraction configuration. The output persister will have several implementations: Network, Filesystem, and Adaptable persister (more may be added later). All persisters persist data gradually, each persister is responsible for the data, and nothing can be lost. Some scanners require the extracted data in the order they were extracted, so the order of data must be kept as it is.

- Network persister

  Network persister sends extracted data to Extraction Processor Persistor and Extraction Processor Parser (in case of extractions where a parser participates)

- Filesystem persister

  Filesystem persister stores extracted data to the filesystem, it is a fallback option in case of problems with connection to the message broker, or it may be chosen as the primary option if so desired. The amount of data Filesystem Persister may store on the filesystem is limited. As extracted data should be stored in the order coming from the extractor, it is also essential to store this order in the filesystem. The H2 database may be used to store messages in the correct order, which should be sent later.

- Adaptable persister

  Adaptable persister adapts to the current situation with connection to MANTA Cloud. When the connection to MANTA Cloud is working, then Network persister is used. In case of connection loss, Filesystem persister is used. If the connection to MANTA Cloud is established again, it uses Network persister again and, in the first place, sends data that were before persisted on the filesystem.

3.3.5.2 Extraction status

An extractor will send precise information about the extraction process using the StatusWriter. The status writer has several implementations, the same as the output persister, so Network status writer, Filesystem status writer, and Adaptable status writer (more may be added later).

- Network status writer

  The network status writer sends extraction status to MANTA Cloud.

- Filesystem status writer

  Filesystem status writer stores statuses to the filesystem. It is a fallback option in case of problems with connection to MANTA Cloud, or it may be chosen as the primary option if so desired.

- Adaptable status writer

  Adaptable status writer adapts to the current situation with connection to MANTA Cloud. A Network status writer is used when the connection to MANTA Cloud is working. In case of connection loss, Filesystem status writer is used. If the connection to MANTA Cloud is established again, it uses Network status writer again and, in the first place, sends data that were before persisted on the filesystem.

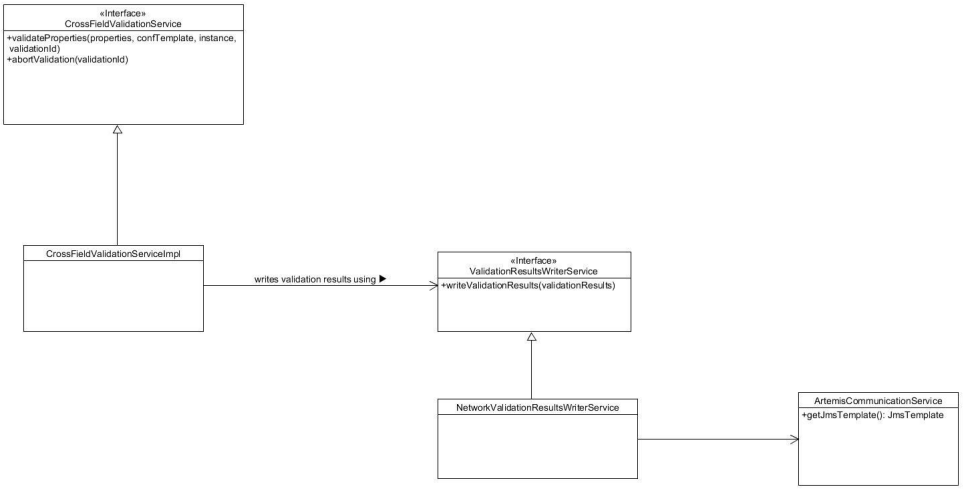


Figure 3.5 MANTA Flow Agent Validator design

3.3.6 Agent Validator

Validation of connection configurations was performed before directly from Admin UI. Now, as Admin UI may not have access to the source system, connection configuration should be performed by Agent. Validation will be run as extraction in a separate process. Extractions are already implemented, so they just should be reused within Agent. Validation results status writer will be used for writing validation results. In the first iteration, validation results may be sent as one message, but later it will be better to send validation results gradually.

In figure 3.5 we can see Agent Validator design.

3.3.7 Agent registration

The Agent has to be authenticated with the Artemis broker to be able to receive and send messages. Artemis supports a wide range of security techniques. For communication encryption and authentication of both parties, TLS will be used. Authentication is performed during the SSL/TLS handshake.

In Admin UI Application Manager will be a possibility to register a new Agent. After registering a new Agent in Admin UI, a user will obtain a configuration for the added Agent. During Agent installation, a user will be asked to provide the Agent configuration. It will be possible to change the configuration read from the configuration. Agent private key may be empty in configuration if the user has chosen to generate key pair by himself and did not provide a private key during the Agent registration in the Agent Manager to include it in the configuration. In this case, a user should provide the private key during Agent installation.

Agent registration is finished during the Agent startup by sending a startup message to Artemis. Connection and provided private key should be verified during the installation process of the Agent. It should be checked if the provided private key has a pair in Artemis truststore. Installer Builder allows running a program or script during installation and this feature may be used to validate the connection to the Artemis server. A simple Java program should be run to validate a provided certificate and connection to Artemis. It will try to create a connection to the Artemis server. If the connection is successfully created, then validation is succeeded, otherwise, it is failed. Even if it is impossible to establish the connection or the private key does not have a pair in Artemis, we should let a user install the Agent. It is a fallback in case of a bug or some problems with the connection. After the validation, the Artemis address is saved

to Agent configuration, and the private key is stored in the keystore.

In figure 3.6 we can see Agent registration process sequence diagram.

3.3.8 Agent installation

Installation is performed using the installation wizard, leading a user through the installation process. The installer will be implemented using VMware InstallBuilder.

Installation activity diagram is shown in figure 3.7.

Installation process wireframes step by step are shown in figures 3.8, 3.9, 3.10, 3.11, 3.12, 3.13, 3.14, 3.15, 3.16, 3.17.

3.3.9 Agent update

The update of the Agent may be performed in two different ways:

- Local update

  A user obtains an installer from Manta, launches it on the server with an installed agent, and, using an update wizard, updates the Agent.

- Remote update

  Admin UI proposes to perform the update, a user initiates the update using the Agent Manager, and the update is handled entirely by Admin UI.

The remote update is the preferred way of how to update the Agent. A local update is a backup option for cases when it is impossible to do the remote update.

Admin UI will perform compatibility control with each message from Agent. The appropriate Agent status will be shown in the Agent Manager according to the compatibility state. Semantic versioning will be used to detect breaking changes, it will allow us to define only major versions with which a service is compatible, and all minor and patch releases will be taken by the major versions. Everything with the same major version is compatible, and everything with a different major version is not.

3.3.9.1 Local Agent update

The local update will be done using an installer generated by InstallBuilder. The same installer which is used for the Agent installation, will be used for local update of the Agent. After the launch of the installer, an update wizard will appear, leading a user through the update process.

- If the Agent is running, then it should be stopped
- Replace old libraries with the new ones
- Update Agent configuration
  - Add new properties with default values if there are some
  - Remove deprecated properties
- Start the updated Agent

Agent local update activity diagram is shown in figure 3.18.

Agent local update process wireframes step by step are shown in figures 3.19, 3.20, 3.21, 3.22, 3.23, 3.24.

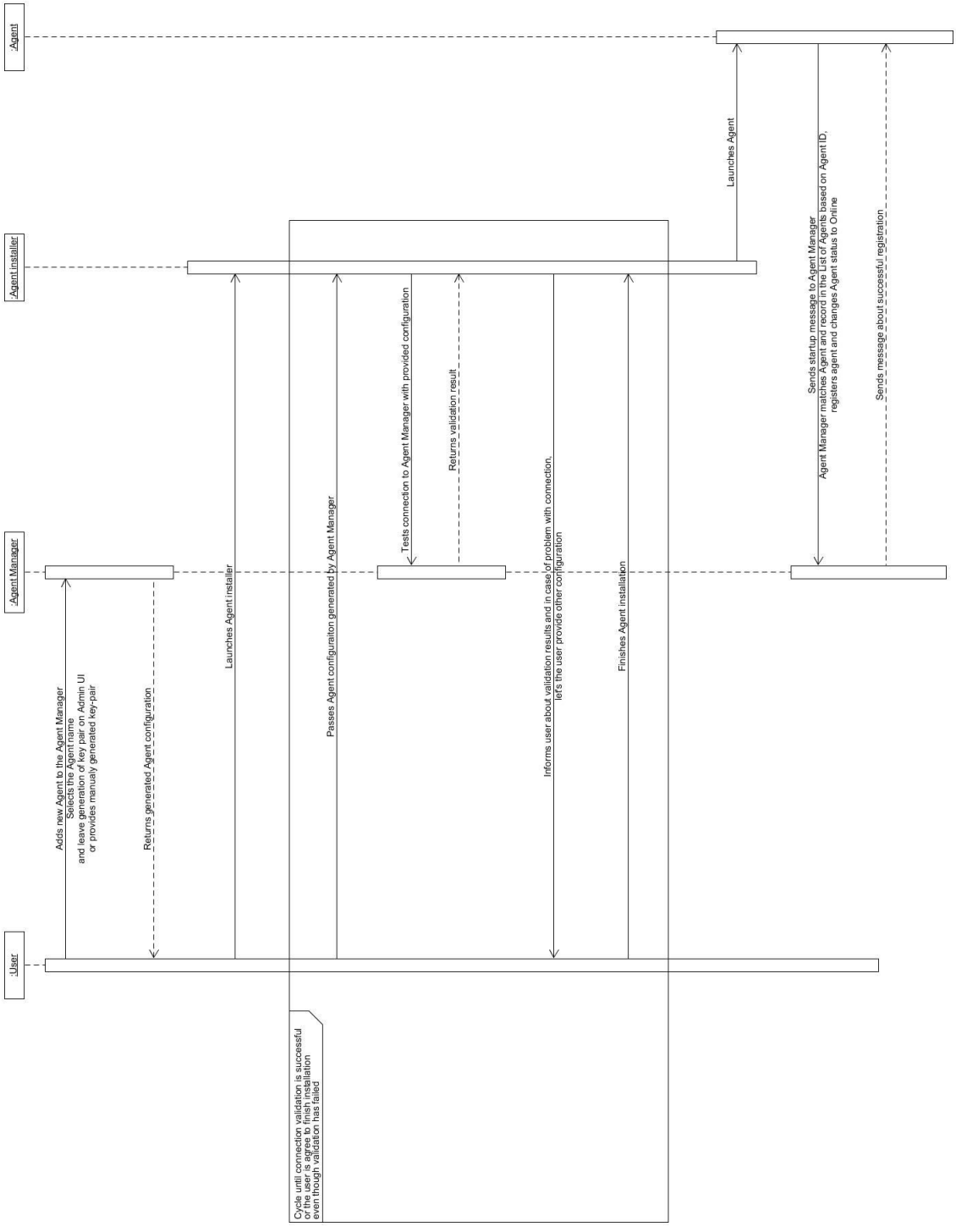
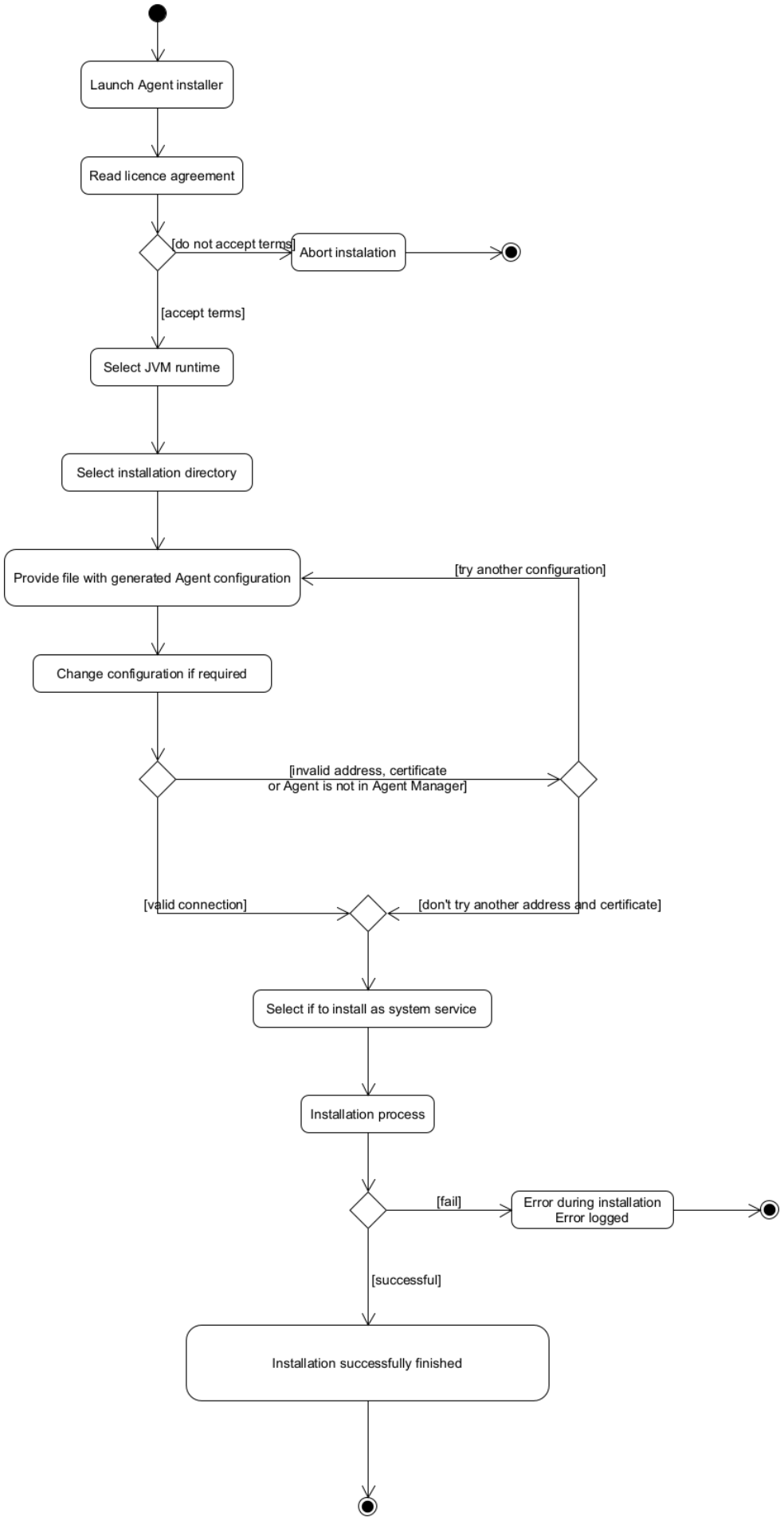


Figure 3.6 Registration process sequence diagram


Launch Agent installer
Read licence agreement
[do not accept terms]
Abort instalation
[accept terms]
Select JVM runtime
Select installation directory
Provide file with generated Agent configuration
[try another configuration]
Change configuration if required
[invalid address, certificate
or Agent is not in Agent Manager]
[valid connection]
[don't try another address and certificate]
Select if to install as system service
Installation process
[fail]
Error during installation
Error logged
[successful]
Installation successfully finished

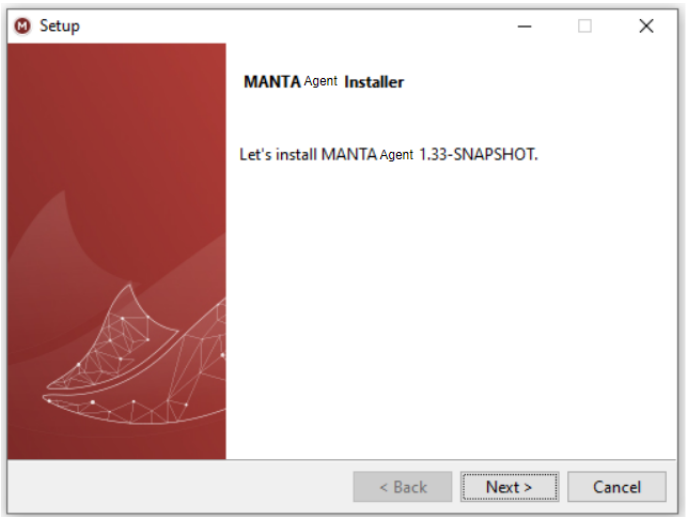


Figure 3.8 Agent installation step 1

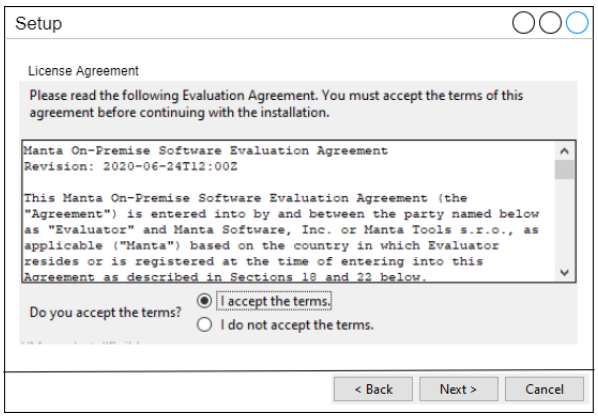


Figure 3.9 Agent installation step 2

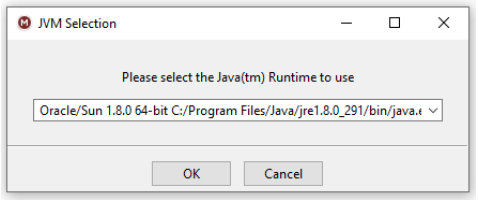


Figure 3.10 Agent installation step 3

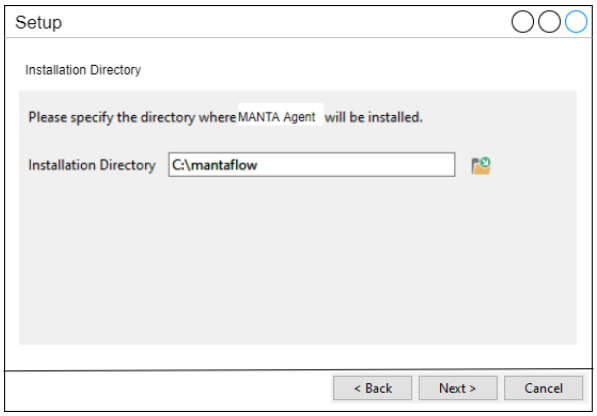


Figure 3.11 Agent installation step 4

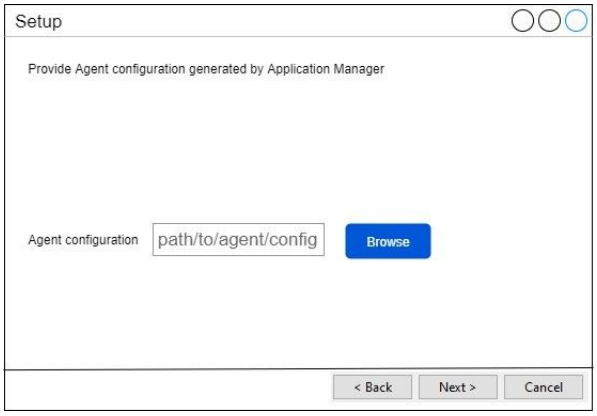


Figure 3.12 Agent installation step 5

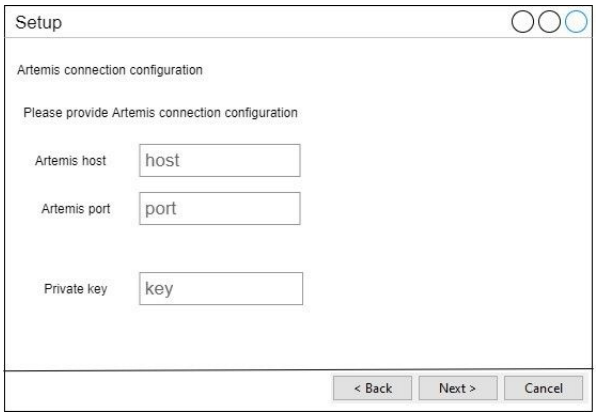


Figure 3.13 Agent installation step 6

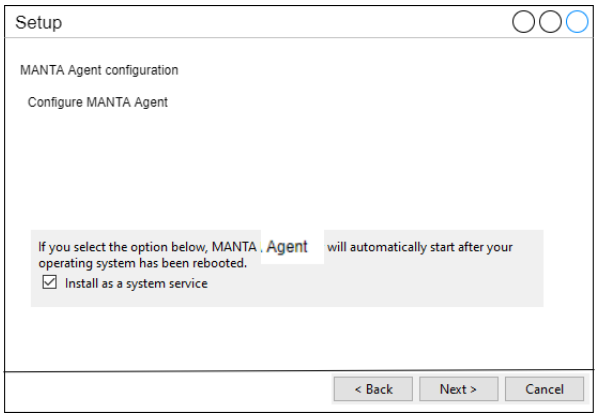


Figure 3.14 Agent installation step 7

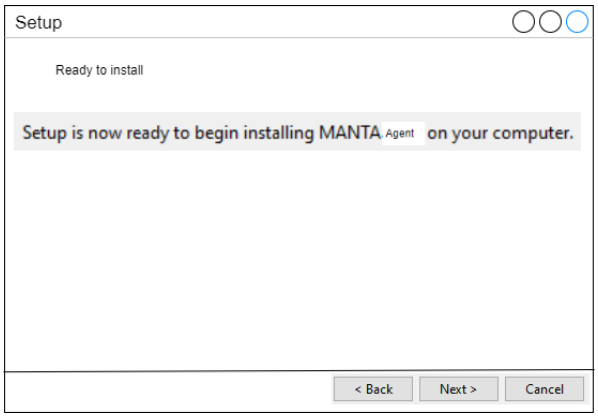


Figure 3.15 Agent installation step 8

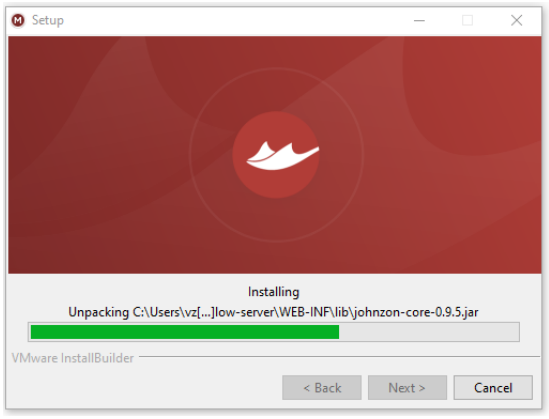


Figure 3.16 Agent installation step 9

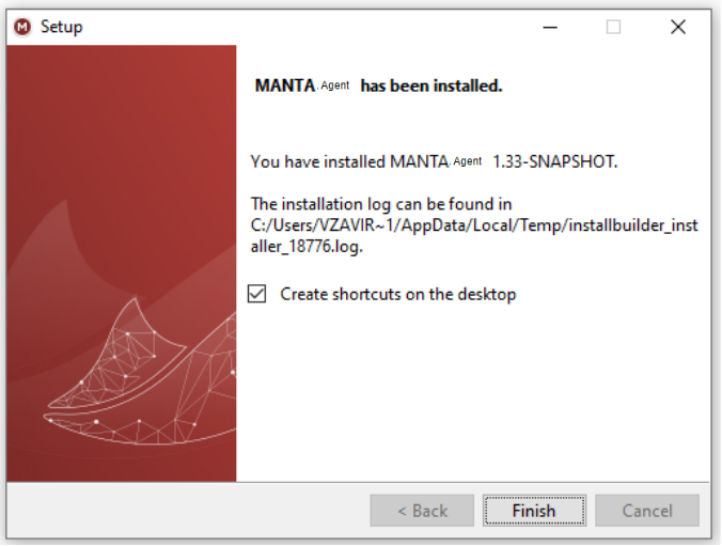


Figure 3.17 Agent installation step 10

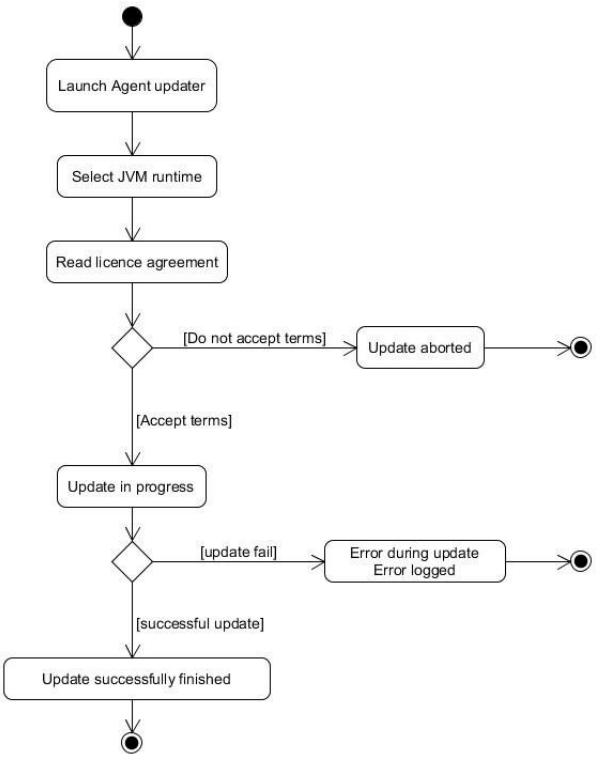


Figure 3.18 Agent local update activity diagram

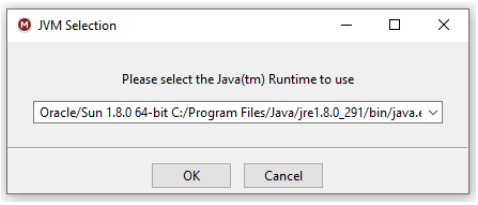


Figure 3.19 Local Agent update step 1

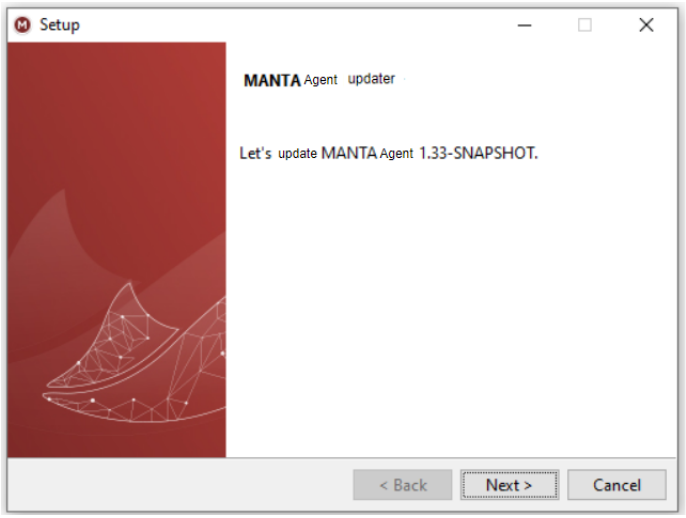


Figure 3.20 Local Agent update step 2

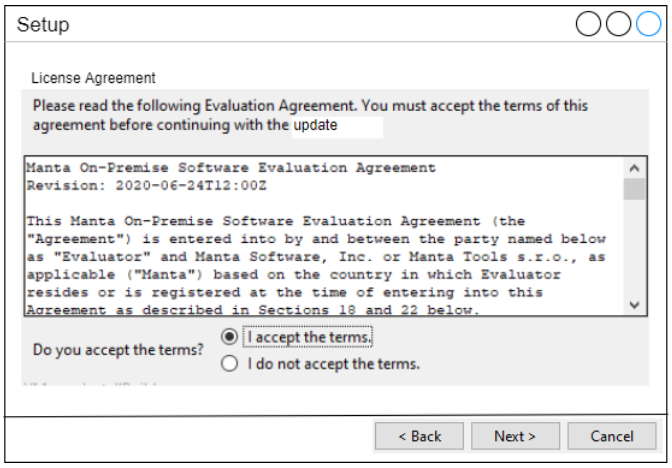


Figure 3.21 Local Agent update step 3

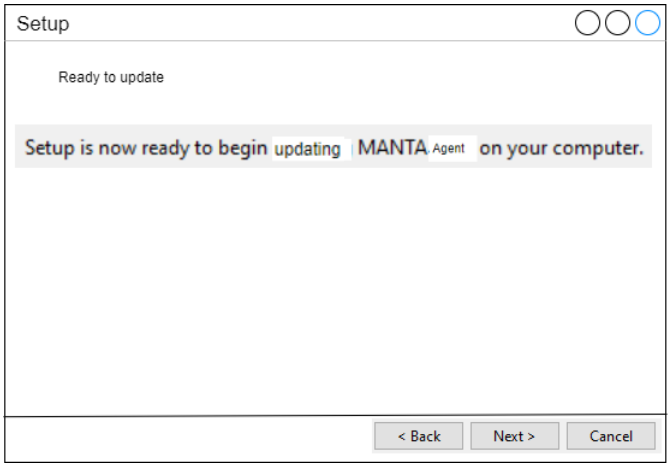


Figure 3.22 Local Agent update step 4

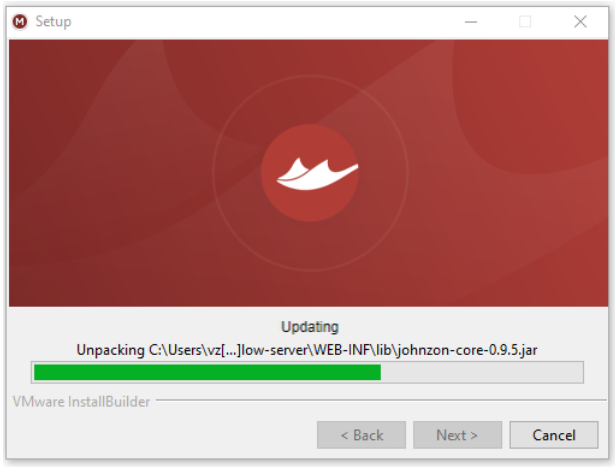


Figure 3.23 Local Agent update step 5

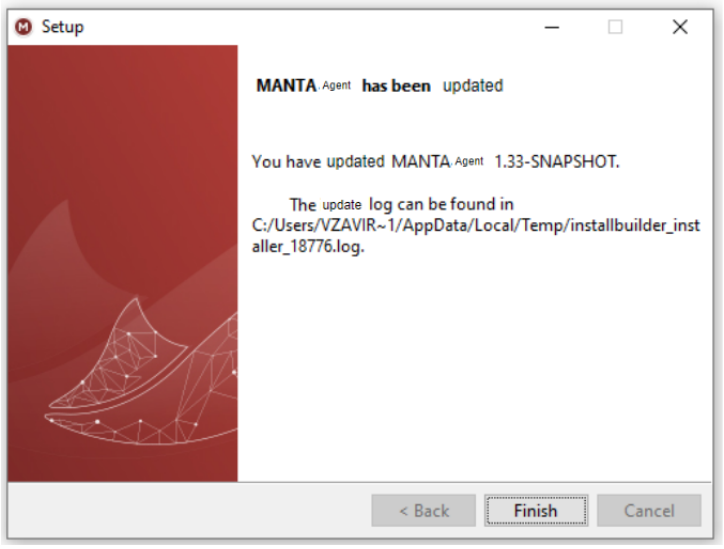


Figure 3.24 Local Agent update step 6

3.3.9.2 Remote Agent update

The update will be launched from Agent Manager by a user command. The possibility to make an update will be shown in Agent Manager in case if some Agent update is available. The update will be started immediately when it is launched by the user. When the update is launched, Agent Manager sends an update command to the Agent together with the new version of the Agent installer.

The same installer which is used for the Agent installation and Agent local update will be used during the remote update process. The installer is launched in the unattended mode so no user interaction is needed during upgrade. When the installation of the new versions of the Agent is launched by the old version of the Agent, the configuration of the old version is passed to the new version installer as arguments by the old Agent Together with the configuration.

During Agent installation connection to Artemis server will be tested and a test connection message to Agent Manager will be sent. The version of the currently installing Agent will be sent in the test connection message to Agent Manager and Agent Manager will perform Agent and Admin UI compatibility test, if the test is failed, then connection validation is also failed.

There are the following options after the new Agent installation and connection test:

- The Agent installation was successful, and the connection test was successful

  The old Agent switches to the non-accepting state, rejecting all incoming commands with an “Updating in progress“ response. When the old Agent has finished all running tasks, it launches a newly installed Agent. The old Agent stops itself after the new Agent is launched. The new Agent will delete the old Agent on the startup. A success message is shown in Agent Manager.

- The Agent installation was not successful, or the connection test was NOT successful.

  Agent installer automatically removes newly added files. An error message about the failed update is sent to the Agent Manager. The error message is shown in Agent Manager.

The remote update is initiated by a user in the Agent Manager in Admin UI. The Agent Manager proposes to update the Agent if an update is available.

Agent remote update from Agent Manager activity diagram is shown in figure 3.25.

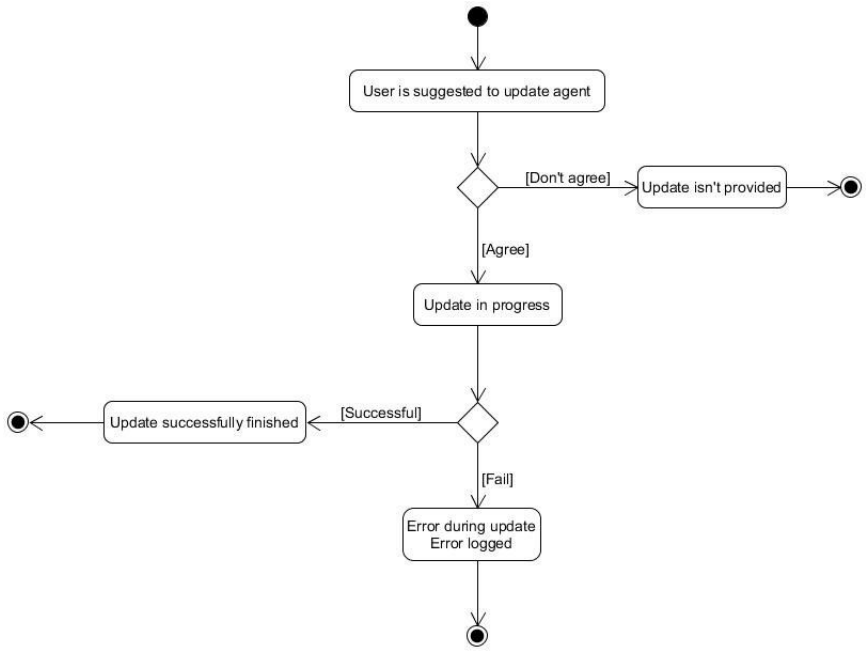


Figure 3.25 Agent remote update from Agent Manager activity diagram

3.4 Admin UI integration design

In this section, we will take a look at changes that should be done to the MANTA Flow application to integrate Agent.

3.4.1 Configurator

The Agent stores the essential minimum configuration required for the Agent functionality. The agent receives most of the configurations from command messages, which allows us to reduce the number of actions required to adjust the configuration.

3.4.1.1 Connection configuration

All connection configurations for technologies for which MANTA supports automatic export will get two new properties that will configure how the extraction is performed. The properties will be placed in the "Extraction" property group.

Newly added properties are:

- Extraction Method

  It defines if CLI or an Agent should be used for extraction. This property will be deleted later when only the Agent is used for extraction.

- Extraction Agents

  This property will be used to determine which agents are installed "near" the server with the technology and can be used for extraction. It is a multivalue select with dynamic values of registered agents.

In the initial implementation, we will use any working agent listed in the "Extraction Agents", but later it is possible to come up with some approaches how to select the best Agent based on the location, connection stability, etc.

The "Extraction Agents" property will contain a list of preferred agents. The list of preferred agents instead of a single agent will allow us to have some backup variants in case of connection problems and to perform some smart Agent selection.

During connection validation, agents listed in the "Extraction Agents" property will be used to perform validation. Each Agent will get a configuration required for validation, and validation will be run separately. Validation results will be shown separately for each Agent. A UX design for this will be required if we decide to use this approach.

3.4.1.2 Connection configuration validation

Suppose the Extraction Method property value is Agent. In that case, connection validation is launched for each Agent listed in the "Extraction Agents" property, but if the property is CLI, then validation is performed through CLI. Suppose the property is not present in the connection configuration for some technology. In that case, it means that validation is not implemented in Agent for this technology, and CLI should be used.

The Agent will perform the whole validation, and Admin UI BE will only pass validation to the Agent and collect validation results. The validation code should be moved to a separate module and shared with the Agent and Admin UI. In the first iteration, it will be possible to run validation in a separate thread in the Agent dispatcher. However, later, it would be better to run each validation in separate processes as it will be done with extractions.

A new instance of CrossFieldValidationService should be created in Admin UI BE, which will pass validation to Agents and collect validation results from them. The new instance of CrossFieldValidationService will use Spring JMS for all communication with Agents, so it will be used for message sending and receiving. We will not use JmsListener here, as we do not need the listener all the time, but only during connection validation. Instead of JmsListener, we will use preconfigured JmsTemplate for message sending and receiving, allowing us to block the validation thread in Admin UI until the validation results are not received.

In figure 3.26, we can see the Admin UI configurator class diagram with the most important moments to the Integration of Agents to Admin UI.

3.4.2 Process manager

Process Manager was developed as a master thesis [19] which servers as a good source of information about it.

Scenario execution should be slightly adjusted as currently, all scenarios are launched as a separate process on the same server as Admin UI is running. Now, all extraction scenarios should be launched differently, as Agents will perform extractions. However, the old way of using CLI for extraction should be kept for cases when Agent is not prepared to extract some technology.

In common configurations, an overridable binary property should be added that will define if CLI or an Agent should be used for extraction. An agent and the extraction method which should be used during extraction will be acquired together with connection configuration.

A new implementation of MantaPlatformExecutor will be added. It will send ExecuteScenarioPhaseCommand messages for the spawn of the Persistence service, Extraction processor, and the start of extraction in an Agent. All ExecuteScenarioPhaseCommand messages should be acknowledged with ScenarioPhaseExecutedResponse messages by all services. If they are not acknowledged for some time (e.g., 10 seconds, should be configurable), then the launch has failed.

Statuses will be collected by the ScenarioProgressListener listener, which will consume ScenarioStatusUpdate messages from all services.

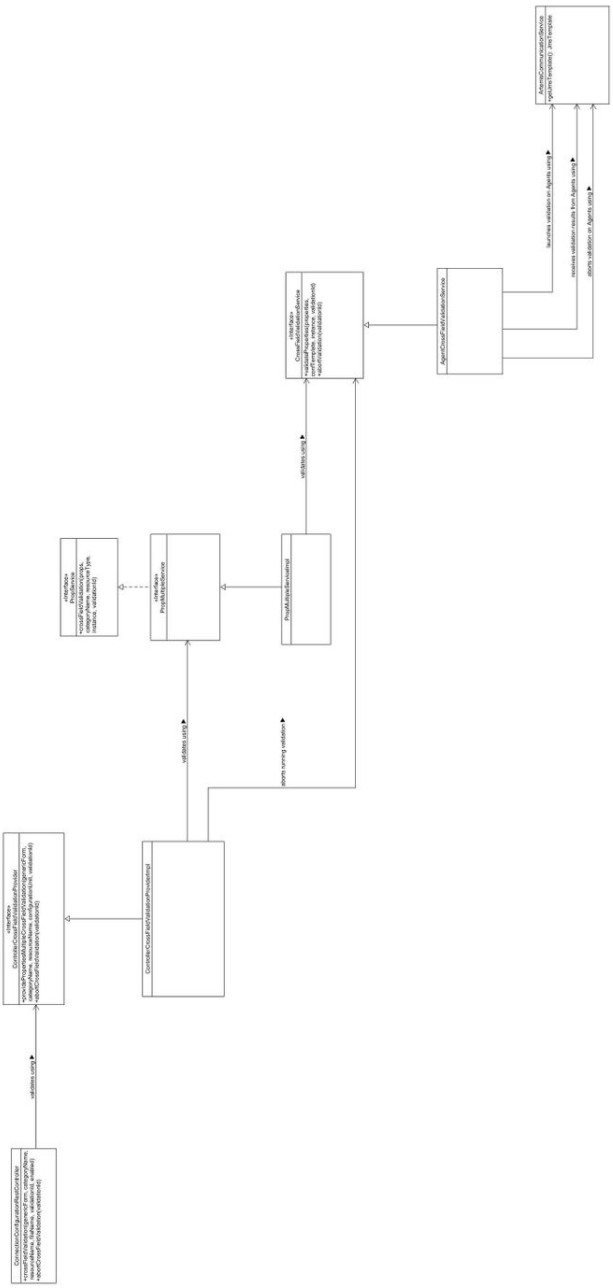


Figure 3.26 Admin UI configurator class diagram

A new implementation of ScenarioExecutionInstance will be added, which will wrap all communication with services through Artemis. Its implementation should mimic the behavior of a locally running process. This approach will allow us smoothly add a code for running scenarios remotely.

New checked exceptions should be created for communication errors, and each of the methods should throw them in case of communication issues. The code that uses ScenarioExecutionInstance must know network communication is used and must be able to handle failures.

ScenarioExecutionInstance interface methods with ScenarioExecutionRemoteInstance implementation:

- isAlive - false if execution is finished or terminated, otherwise true

  All services which are participating in execution will be requested about their status. If there is no response from some of the services for some constant time (10 seconds, for instance), there are taken as not running, and the method returns false. If there is no response from some of the services, TerminateScenarioExecutionCommand is sent to all services participating in the execution. If all the services respond within the timeframe, then true is returned. If the ScenarioFinished message with SUCCESS or FAILURE outcome was registered by WorkflowExecutionScenarioStatusStore, then false is returned.

- waitFor - blocking call until execution is not finished or until timeout

  WorkflowExecutionScenarioStatusStore service will store a semaphore for each currently running remote extraction scenario. The semaphores will allow us to provide a blocking waitFor functionality. If the ScenarioFinished message with SUCCESS or FAILURE outcome was received, then waitFor is unblocked immediately using Semaphore release by a thread that received the message.

  Periodically performs isAlive checks until the ScenarioFinished with SUCCESS outcome message is received from all services, the ScenarioFinished with FAILURE outcome message is received from any service, or isAlive returns false. The thread called this method is blocked using Semaphore acquire

- terminate - Forcibly terminates execution

  Sends TerminateScenarioExecutionCommand and ScenarioFinishedUpdate messages to all services participating in the execution.

  Process Manager integration class diagram is shown in figure 3.27.

3.4.3 Agent manager

Agent Manager shows all connected agents, and their statuses and provides a possibility to provide different interactions with agents (e.g., update, disconnecting). All actions which may be performed with an agent are hidden under three dots.

Agent Manager class diagram is shown in figure 3.28.

3.4.3.1 Agent status

On the startup, the Agent will send a message about its startup, so the Agent Manager can take the Agent as alive (status in Agent Manager is changed to online) and consider the Agent as successfully installed. The Agent will respond to regular heartbeats requests from the Agent Manager to inform it that it is still available. The Agent is automatically marked as offline if there is no response from it for some period of time. The startup message will contain general information about the Agent.

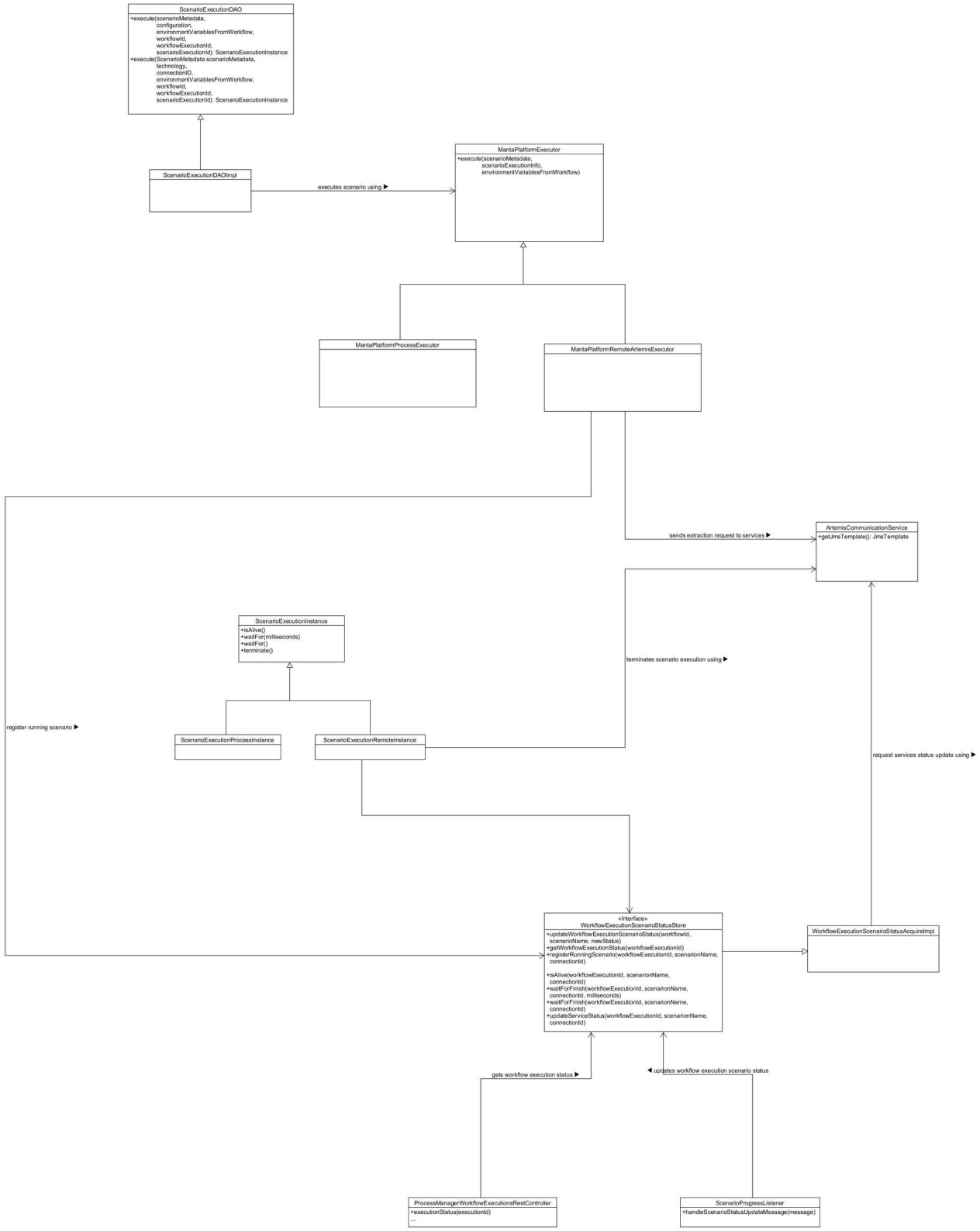


Figure 3.27 Process Manager integration class diagram

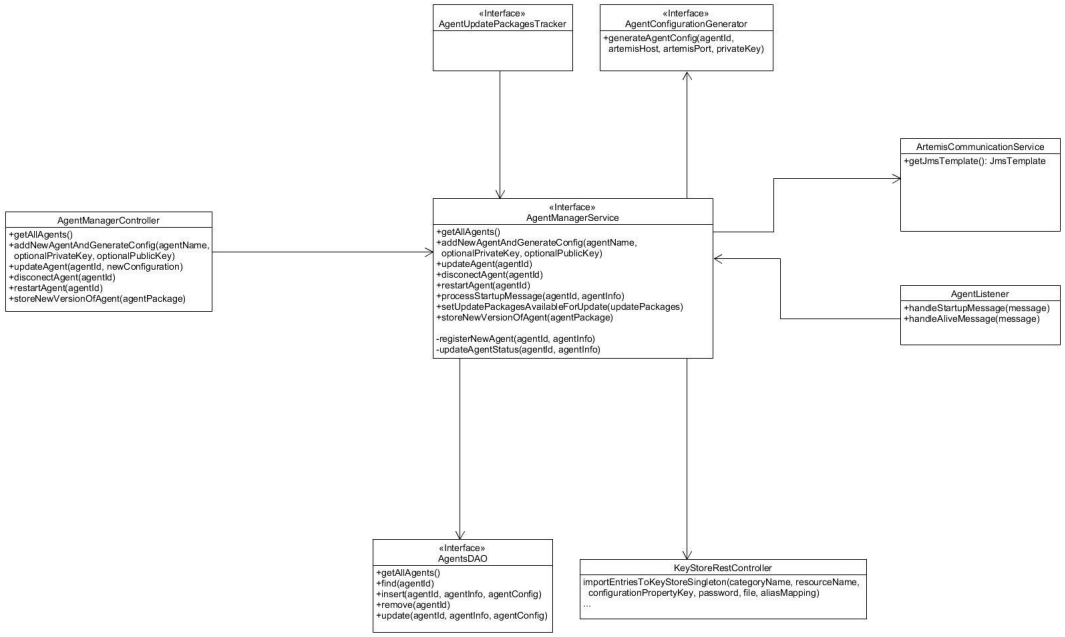


Figure 3.28 Agent Manager class diagram

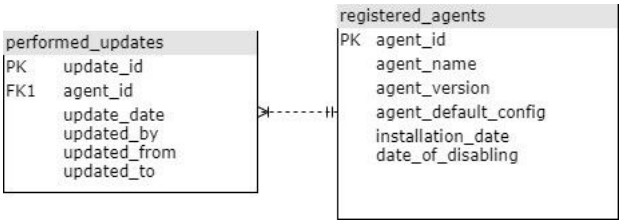


Figure 3.29 Agent Manager database diagram

3.4.3.2 Storing information about Agents

All registered Agents will be stored in a table in the H2 database. The table will consist of Agents added to the List of agents in Application Manager previously. There will be all Agents, registered and unregistered.

The table will contain the following columns: agent id (generated during adding the Agent to the List of Agents), agent name (selected by a user), agent version, agent configuration, registration date, and unregistration date.

Agent statuses (online or offline) will be stored on the heap, as statuses will be updated frequently, and there is no need to store them in the database. The information about Agent statuses does not have to be persisted on the Admin UI shutdown as on the next startup, it will be outdated and will be updated anyway.

Information about performed remote Agent updates will be stored in the table in the H2 database. The table will contain the following columns: update date, updated by, updated from, updated to.

In figure 3.29 we can see Agent Manager database diagram.

Id	Name	Version	IP address	Status	...	
1	prod-oracle-agent	1.0	149.154.192.176	Blocked, update required	...	⋮
2	prod-mssql-agent	1.1	149.154.192.177	Update available	...	⋮
2	prod-db2-agent	1.2	149.154.192.178	Active	...	⋮

Figure 3.30 Agent Manager view

3.4.3.3 Generating Agent configuration

Agent configuration generating will simplify the installation process for users, as all required information will be stored in one configuration file, which should be passed to Agent installation during installation. Agent configuration is a ZIP file with a YAML configuration file, Artemis keystore, and truststore for mTLS communication with Artemis.

In the YAML configuration file will be the following information:

- Agent ID
- Artemis host
- Artemis port
- Keystore password
- Truststore password

3.4.3.4 Unregistering Agents

It will be possible to unregister a registered Agent. It will cause the stop of all running processes in the Agent and its shutdown. If the Agent is not currently running or there are some problems with the connection, the Agent will be marked as unregistered in the database, and once it is back online, the unregister message will be sent to it.

The unregistered Agent will be soft-deleted from the database when the Agent confirms receipt of the unregister message. Soft delete means not real delete of the record, but setting the unregister date column in the database to the date when the record was soft-deleted. An Agent who was once unregistered can not be registered again.

3.4.3.5 Agent Manager wireframes

The wireframes of Agent Manager are shown in figures 3.30, 3.31, 3.32, 3.33.

3.5 Extraction Processor design

MANTA Flow Extraction Processer is a component responsible for receiving and processing the outputs of the MANTA Flow Agent. The component is divided into two subcomponents:

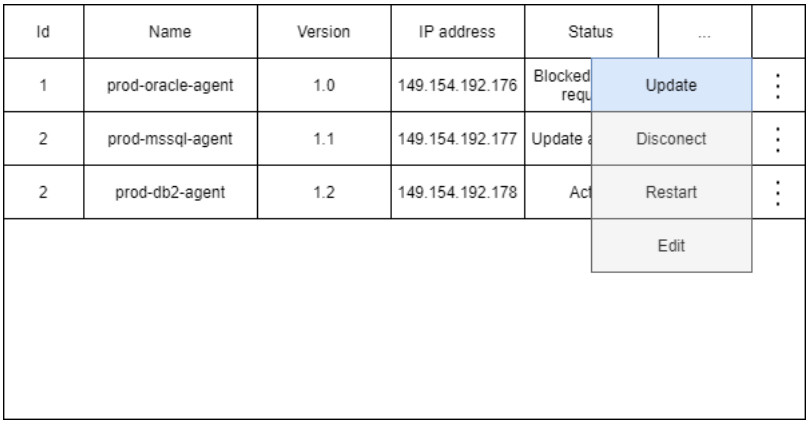


Figure 3.31 Action that can be performed with each Agent

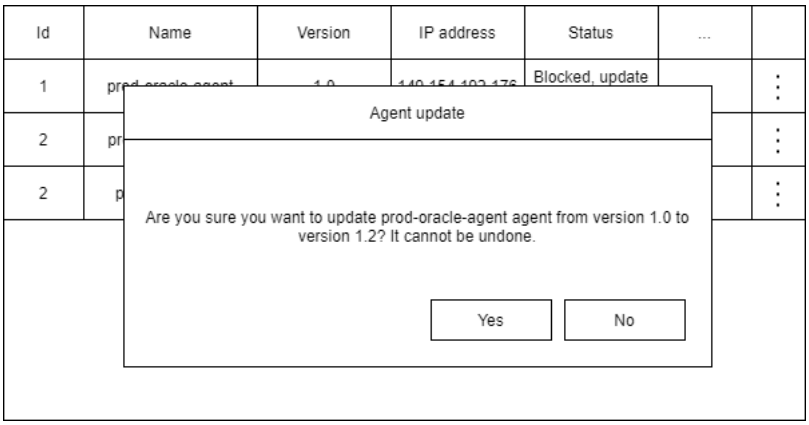


Figure 3.32 Warning before remote Agent update launch

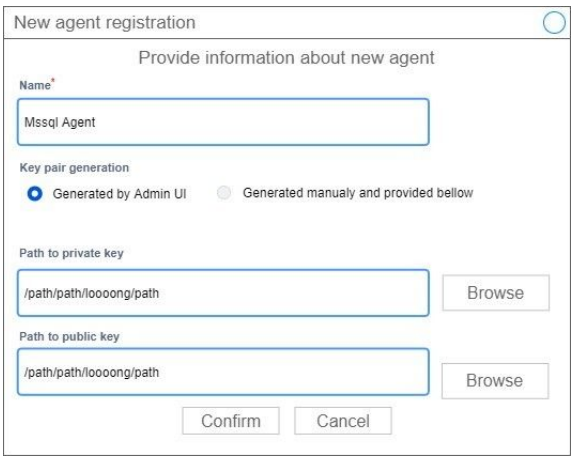


Figure 3.33 New Agent registration form

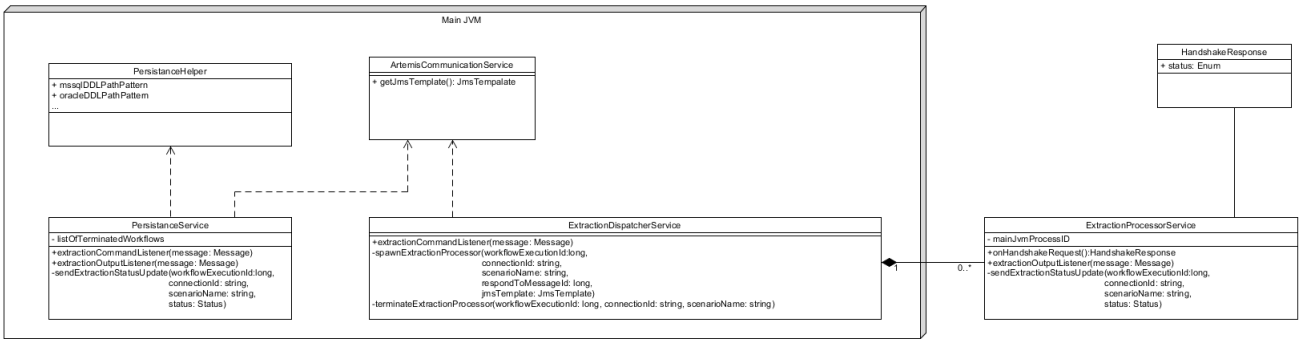


Figure 3.34 Extraction processor class diagram (source is MANTA internal documentation)

- Extraction Processor Persistor - persist extraction outputs
- Extraction Processor Parser - parses and processes outputs for some technologies where it is needed. It passes the results of the output processing to the Persistor

For now, the Extraction Processor will be a part of the Admin UI application and will be running on the same application server. However, it will be designed and implemented as a separate application so that it may run so in the future.

In figure 3.34, we can see class diagram of Extraction Processor with Persistor and Parser parts.

3.5.1 Extraction Processor Persister

The MANTA team already implemented Extraction Processor Persister, so there is no need to design and implement anything in this part. Here will be mentioned some important moments about its design based on the MANTA internal documentation.

Extraction Processor Persister will listen to new extracted outputs sent by the Agent and Extraction Processor Parser (if applicable for technology). Each of the outputs will be processed by a separate thread. A listener should be registered for each extraction output type. The persistence service must listen to and be aware of termination commands/terminated workflows to ensure not to process inputs that will arrive after the termination command is received.

3.5.2 Extraction Processor Parser

The extraction Processor Parser still needs to be implemented, but it was already designed, so there is nothing to design here. Here will be mentioned some important moments about its design based on the MANTA internal documentation.

The same as with the Agent Extractor component running as a separate process, the Extraction Processor Parser will also be running as a separate process. Parsers are, by design, tied to one connection, and having each parser as a separate process makes it easier to manage resources used by the spawned process. The newly spawned process will register the JMS consumer.

After Extraction Processor Parser is spawned, it will listen to the outputs extracted by the Agents that should be parsed, parse the outputs and send the data created after the parsing to

the Extraction Processor Persister. When the extraction is finished, the parser process will be self-destroyed.

Chapter 4
Implementation

In this chapter, we will talk about the final implementation. The chapter is divided into a number of parts where we will look at a particular part of the implementation in each part. The chapter is divided into Agent implementation, Admin UI integration implementation and Extraction processor implementation. Due to the huge amount of code, only essential parts of the implemented solution will be discussed.

4.1 Agent implementation

The Agent is implemented using the Spring Boot framework. The application is distributed as a fat JAR. Fat JAR is a JAR file that contains the Java program and all its dependencies. Agent Dispatcher, Agent Extractor, and Agent Validator program files are located within the fat JAR. The same JAR file is used to launch all the applications. In order to run a specific program, we provide the corresponding path to the main class at startup. The main class is the entry point of all Java programs. One far JAR for all applications allows us to share common dependencies among them, which allows us to reduce the final size of the Agent.

4.1.1 Agent Dispatcher

In code listing 4.1 we can see a simplified version of JMS listener that listens to extract scenario messages on topic/scenarioCommands topic. JmsListener annotation is used to register the handleExtractionMessage method as a JMS listener. A selector is defined to select only messages intended for the particular Agent. The selector selects only messages that are for this particular Agent. The filtering is done by the Agent ID parameter. Also, the selector selects messages only of a particular type and for a particular workflow phase. The messages are also filtered by version to deal with possible breaking changes. The Agent should not accept messages that were produces by a producer with a different major version than has the Agent. The message contains information about extraction scenario and extraction configuration.

```
1 @JmsListener(
2     destination = "topic/scenarioCommands",
3     selector = "serviceID = '${manta.agent.common.id}' "
4             + "AND messageType = 'ExecuteScenarioPhaseCommand' "
5             + "AND phaseName = 'EXTRACTION' "
6             + "AND " + "senderServiceVersion LIKE ' "
7             + "#{@agentProperties.common.appMajorVersion}"
8             + " .%' "
```

```
)
public void handleExtractionMessage(
    extractionConfiguration,
    scenarioInformation
) {
    try {
        appLauncher.launchExtractorApp(
                scenarioInformation,
                extractionConfiguration
        );
    } catch (Exception e) {
        scenarioStatusUpdatesProducer
            .extractorAppLaunchFailed(scenarioInformation, e);
    }
}
```

Code listing 4.1 Listener to extract scenario messages

When the message is accepted, the extraction scenario information and extraction configuration are passed to app launcher service, that will launch Agent Extractor process.

In code listing 4.2 we can see the simplified launchExtractorApp method that launches the Agent extractor process. To launch the Agent Extractor, the same Java is used that was used to launch Agent Dispatcher application. It is an important moment, as on the same machine several Javas may be installed. The method is gradually constructing command that will be launched using the ProcessBuilder class from the Process API.

```

public void launchExtractorApp(
    scenarioInformation,
    extractionConfiguration,
) {
    List<String> processParameters = new ArrayList<>();
    processParameters.add(javaPath);

    processParameters.add("-cp");
    processParameters.add(agentExtractorClassPath);

    // start Spring Boot application with non-default class
    processParameters.add("-Dloader.main=eu.profinit.manta.flow.agent.
        extractor.ExtractorApp");
    // start Spring Boot application with non-default class
    processParameters.add("org.springframework.boot.loader.
        PropertiesLauncher");
    processParameters.add(String.format("--spring.config.additional-
        location=%s",
    // pass application.yml configuration
    agentProperties.getSpringConfigLocation()));
    // set configuration file name for Agent extractor configuration
    processParameters.add("--spring.config.name=application-extractor");

    LOGGER.info("Starting extraction process with following parameters:
        {}", processParameters);

    DisposableProcess extractorProcess = new DisposableProcessImpl(
            processParameters.toArray(new String[0])
    );

    // add input parameters
```

```
29
30      Process process = extractorProcess.spawn();
31  }
```

Code listing 4.2 Extractor application launcher

4.1.2 Agent Extractor

In code listing 4.3, we can see an extraction start point in Agent Extractor. As it is a Spring Boot application, the main method starts the Spring Boot application. When the application is fully initialized, it starts the execution of extraction from this point. The parentProcessValidatorRunnable service periodically validates that the parent Agent Dispatcher process is running, and if it is not running, then it stops the Agent Extractors process. The extractionService service starts extraction by selecting a particular extractor based on extraction configuration. When the extraction is finished, the Agent Extractor process is self-destroyed.

```
1
2   try {
3       parentProcessValidatorRunnable = parentProcessValidatorProvider.
            getParentProcessValidator();
4
5       // Start parent process watcher
6       parentProcessValidatorRunnable.start();
7
8       // Run the extraction
9       extractionService.extract();
10
11      LOGGER.info("Extraction has successfully finished. Stopping Agent
            Extraction process.");
12  } catch (Exception e) {
13      LOGGER.log(Categories.extractorErrors().errorDuringExtraction()
14                              .catching(e));
15      LOGGER.info("Extraction has finished with fatal error. Stopping
            Agent Extraction process.");
16  } finally {
17      if (parentProcessValidatorRunnable != null) {
18          // Stop parent process watcher and finish the whole processing
19          parentProcessValidatorRunnable.stop();
20      }
21
22      shutdownService.shutdownApplication();
23  }
```

Code listing 4.3 Agent Extractor application extraction start

In code listing 4.4, we can see implementation of Extraction service. The service obtains all required inputs for extraction and launches the extraction using a particular extractor.

```
1
2   /**
3    * Extractor chosen based on extraction scenario name
4    */
5   @Autowired
6   private AgentExtractor extractor;
7
8   @Override
9   public void extract() {
```

```
10      Map<String, String> extractionProperties = System.getProperties().
            entrySet().stream()
11              .filter(entry -> entry.getKey() instanceof String && entry.
                    getValue() instanceof String)
12              .collect(Collectors.toMap(entry -> (String) entry.getKey(),
                    e -> (String) e.getValue()));
13
14      extractor.extract(new ExtractionConfiguration(extractionProperties),
             persistor, statusWriter);
15  }
```

Code listing 4.4 Extraction service implementation

The extractor is selected using the Spring functionality. Adding Autowired annotation allows Spring to resolve and inject the Agent extractor bean into the ExtractionServiceImpl bean. Each Agent extractor is a Spring bean. Technology Agent extractor bean is created only if it is required based on the provided extraction configuration.

In code listing 4.4, we can see the implementation of the Postgresql Agent extractor. PostgresqlDatabaseAgentExtractor bean is created only if the scenarioName parameter is equal to postgresqlExtractorScenario.

```
1
2   @Component("postgresqlExtractorScenario")
3   @ConditionalOnProperty(
4           value="scenarioName",
5           havingValue = "postgresqlExtractorScenario"
6   )
7   public class PostgresqlDatabaseAgentExtractor implements AgentExtractor
        {
8
9       // Agent extractor parameters
10      ...
11
12      @Override
13      public void extract(ExtractionConfiguration configuration,
14                          ExtractionPersistor extractionPersistor,
15                          StatusWriter statusWriter
16      ) {
17          PostgresqlExtractorImpl extractor
18              = new PostgresqlExtractorImpl(dao,
19                  // persists DDLs
20                  ddlWriter, new PostgresqlDictionaryWriterImpl());
21
22          // extractor configuration code
23          ...
24
25          extractor.extract();
26
27          // sending extraction outputs and removing files created during
                extraction
28      }
```

Code listing 4.5 Postgresql Agent extractor implementation

Already implemented extractors that were used before in MANTA Flow CLI were reused here.

4.1.3 Agent Validator

The Agent Validator process launched similarly to the Agent Extractor, and its inner structure is also similar. The only difference is that connection validation is performed in the Agent Validator.

4.1.4 Agent installation

Agent installer was implemented by the MANTA team. It was implemented using the VMware InstallBuilder tool.

4.1.5 Agent update

The Agent update may be performed in two ways: locally and remotely. In both cases, the Agent installer participates in the update. Agent updater is bundled together with Agent installer to the same executable. The main difference between local and remote updates is how the installer is transferred to the server where the Agent is running and who is launching the installer on the server to perform the update. In the case of the local update, the installer is transferred to the server and launched by a user. In the case of remote update, the installer is transferred automatically, and the Agent launches the installer.

4.1.5.1 Local Agent update

Local Agent update is was also implemented by the MANTA team. The Agent update is also implemented using the VMware InstallBuilder tool.

4.1.5.2 Remote Agent update

When the remote update is triggered, the Admin UI sends the Agent installer to the Agent together with a command to perform the remote update. The Agent receives the command and saves the Agent installer to a temporary folder to prepare it for the update.

In code listing 4.6, we can see the simplified implementation of the launch remote agent update listener. We store the received Agent installer on the file system, and after it, we pass doing the remote upgrade to the agentRemoteUpdateService.

```
1
2  Path agentInstallerPath = null;
3  try {
4      // store Agent installer sent as stream on the filesystem
5      agentInstallerPath = storeAgentInstaller(message);
6
7      agentRemoteUpdateService.doRemoteAgentUpdate(agentInstallerPath,
           mantaCorrelationID, agentInstallerVersion);
8  } catch (Exception e) {
9      AgentRemoteUpdateResultDTO agentUpdateResult
10             = new AgentRemoteUpdateResultDTO(AgentUpdateResultType.
                   FAILED, e.getMessage());
11     agentRemoteUpdateService.sendRemoteUpdateFinishedMessage(
           agentUpdateResult);
12
13     throw new IllegalStateException("An exception has occurred during
           remote Agent update", e);
14 }
```

Code listing 4.6 Launch remote agent update listener implementation

In code listing 4.7, we can see the simplified implementation of the doRemoteAgentUpdate method in the AgentRemoteUpdateService. We are launching the Agent installer to install a new version of the Agent right next to the old version of the Agent. It is different then local Agent upgrade. During the local update, we can just replace the Agent application files as the Agent is not running. But during the remote update, the Agent is still running as we want to be sure the remote Agent update will finish successfully. So we are stopping the old Agent only when the installation of the new Agent was successfully finished, otherwise the old Agent is keep running and we are showing a message about the unsuccessful remote update in the Agent Manager. With this approach, we reduce a window when the Agent was broken during the remote update and a user intervention is required.

```
1
2  // run Agent installer and retrieve agent update results
3  AgentRemoteUpdateResultDTO agentUpdateResult = runAgentInstaller(
       agentInstallerPath, mantaCorrelationID);
4  sendRemoteUpdateFinishedMessage(agentUpdateResult);
5
6  // run migration script only if new Agent installation has finished
       successfully
7  if (agentUpdateResult.getUpdateResultType() == AgentUpdateResultType.
       SUCCESSFUL) {
8      // we should run migration script and immediately after it shut down
           this instance of Agent
9      runMigrationScript(agentInstallerVersion);
10
11     logger.info("Shutting down instance of old Agent");
12     shutdownService.shutdownApplication();
13 }
```

Code listing 4.7 Simplified implementation of the doRemoteAgentUpdate method in AgentRemoteUpdateService service

4.2 Admin UI integration implementation

The existing Admin UI application was modified and extended to integrate MANTA Flow Agent.

4.2.1 Configurator

New parameters were added to connection configurations of all technologies that are already supported by the Agent. In figure 4.1 we can see the extraction configuration. The Extraction Agents parameters allow us to select the Agent performing the extraction. Next to each registered Agent, the Agent status is shown. It is possible to select multiple Agents, and then the first Agent that is online and has a compatible version, and is ready for the extraction will be selected.

4.2.2 Process manager

The Process manager was extended to enable the execution of scenarios not only with MANTA Flow CLI but also with the Agent. Based on the connection configuration, the Process manager decides how a scenario should be executed and who should be the scenario runner.

In code listing 4.8, we can see the simplified implementation of the execute method in the MantaScenarioRemoteArtemisExecutor service. The ID of the Agent that should perform an extraction is retrieved from the extraction configuration. The temporary folder where the extraction output will be stored in MANTA Flow CLI is prepared in advance. Parser service

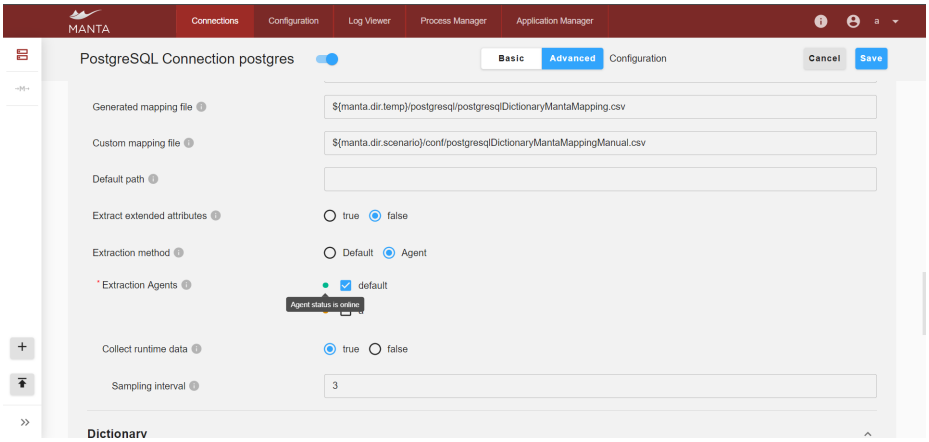


Figure 4.1 Extraction configuration

is spawned only if it is needed. The extraction is launched, and right after, an instance of the ScenarioExecutionRemoteArtemisInstanceImpl object is returned. Using this instance, it is impossible to determine the current state of extraction.

```
1
2  extractionAgentId = getExtractionAgentId(scenarioExecutionInfo.
       getScenarioRunnerInfo());
3
4  prepareTempFolderForScenario(scenario, connectionConfigurationProperties
       , environmentVariablesFromWorkflow);
5
6  boolean doesParserParticipates = determineIfParserParticipatesInScenario
       (scenario.getScenarioName());
7
8  // spawn outputs parser if it is participates in scenario
9  if (doesParserParticipates) {
10     // wait till it is not initialized (until successful response
           message is not received)
11     scenarioCommandsProducer.spawnOutputsParserAndWait(
12             correlationId,
13             otherParameters
14     );
15
16 }
17
18 scenarioCommandsProducer.launchExtractionScenarioOnAgent(
19         extractionAgentId,
20         otherParameters,
21         extractionConfiguration
22 );
23
24 return new ScenarioExecutionRemoteArtemisInstanceImpl(
25         extractionAgentId,
26         otherParameters
27         scenarioCommandsProducer
28 );
```

Code listing 4.8 Simplified implementation of the execute method in MantaScenarioRemoteArtemisExecutor service

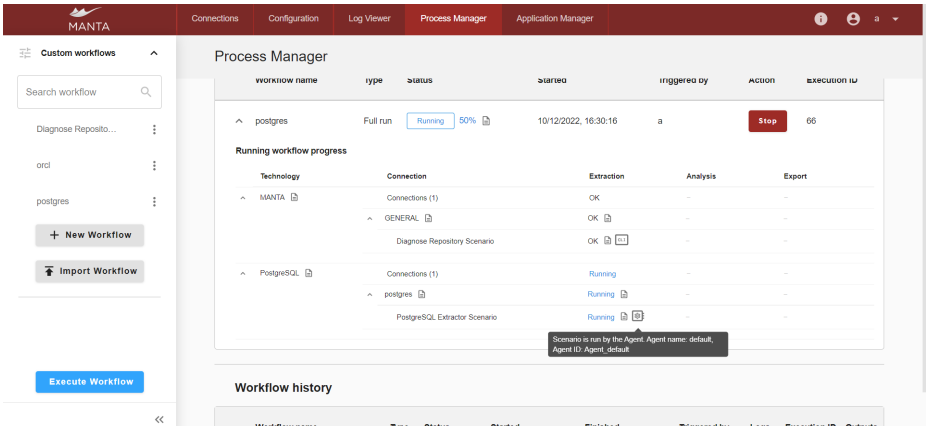


Figure 4.2 Information about scenario runner in Admin Process Manager

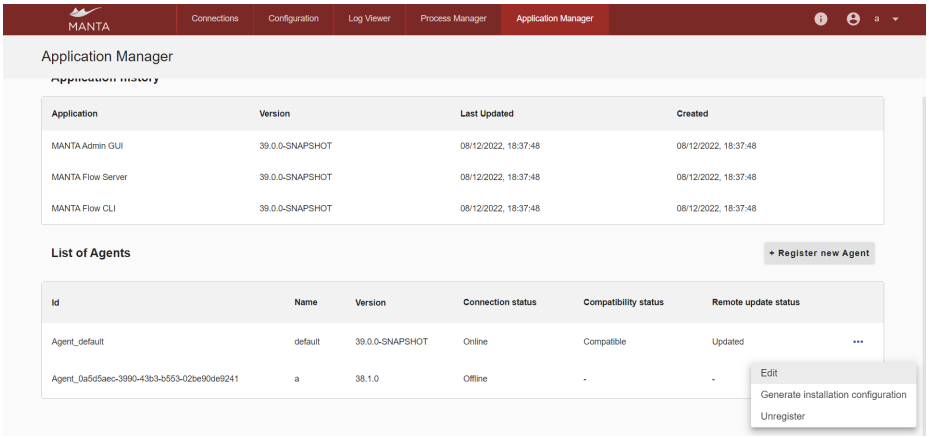


Figure 4.3 Agent Manager general view

To the Process Manager dashboard was added an information about a scenario runners. In figure 4.2 we can see an information about the scenario runner that is performing a scenario.

4.2.3 Agent manager

The Agent Manager back-end was implemented as a new module that is now is a part of Admin UI. The Agent Manager front-end was implemented in React framework and now it is a part of Admin UI web application.

In figure 4.3 we can see the Application Manager general view. In the List of Agents table we can see the list of registered Agents with information about each Agent.

A new Agent can be registered using the Register new Agent button that is located right above the table. In figure 4.4 we can see the page where it is possible to register a new Agent. Using the similar page, it is also possible to edit an already registered Agent configuration.

In figure 4.5 we can see the page where the Agent configuration may be generated for the newly registered Agent. A user is redirected on this page right after the new Agent registration, but it is also possible to generate this configuration later if it is needed. The mTLS configuration may be generated by MANTA or generated manually and provided by a user.

A user in Agent Manager can initiate the Agent remote update. Agent installer is bundled together with each installation of Admin UI. The bundled Agent installer has the same version

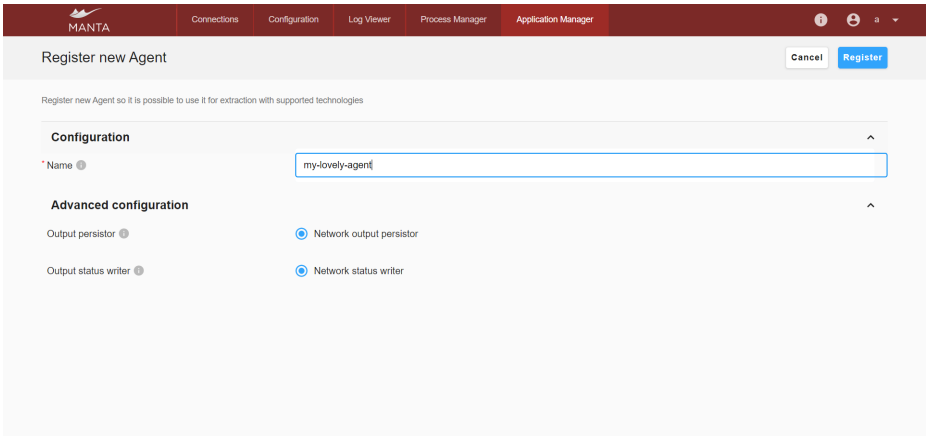


Figure 4.4 Register new Agent page

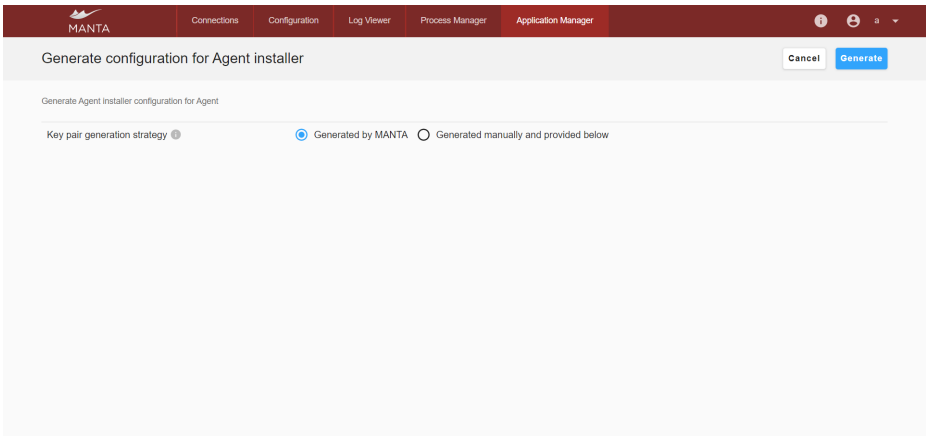


Figure 4.5 Generate Agent configuration

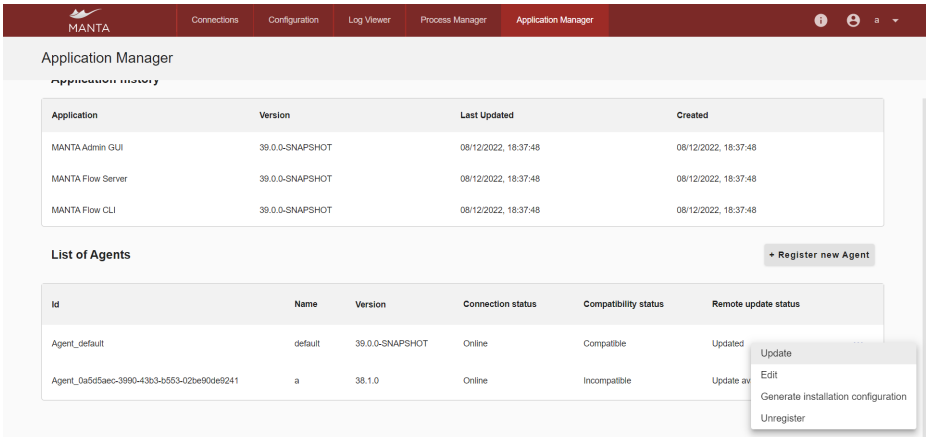


Figure 4.6 Update in Admin UI Agent Manager

as the Admin UI, and with each upgrade of Admin UI, the Agent installer is replaced by a new one. The Agent installer ZIP is signed during its build on the Jenkins server, and the certificate is verified in Admin UI during its startup. If certificate validation is failed, then such the Agent installer cannot be used for the remote update.

In figure 4.6 we can see the Agent Manager table with opened context menu where it is possible to launch the Agent update.

4.3 Extraction Processor implementation

Extraction Processor is now integrated into the Admin UI, but in the future, it can be separated and become an individual service.

4.3.1 Extraction processor persister

Extraction process persister was implemented by the MANTA team.

4.3.2 Extraction processor parser

The extraction Processor Parser process is spawned similarly to how it is done with Agent Extractor and Agent Validator application processes in the Agent. The parser process is running until the explicit command about the end of parsing is received. At the end of parsing, the parser sends the output of parsing to the Extraction Processor Persistor service for persisting. Already implemented parsers that were used before in MANTA Flow CLI were reused here.

Chapter 5

Testing, documentation, CI/CD

In this chapter, we will take a closer look at how the Agent and related components were tested, the Agent documentation, and the Continuous Integration/Continuous Delivery practices used during the Agent's development and delivery.

The Agent development and testing was done according to the Manta development standards. The Agent was not tested only by its developer but also by Manta testers. Other developers did code reviews of code before it was merged into the main code branch in the GIT tool.

5.1 Testing

The JUnit and integration tests were created during the implementation of all related parts. There are over 150 tests that cover most of the Agent related code base.

Before the module's beta release and again before the official MANTA release, the Quality Assurance team performed end-to-end tests with different scenarios. Also, according to the MANTA policy, regular code reviews were done before the merge of each pull request. Other developers perform code reviews to assure code quality.

Integration tests were implemented that test the whole Agent's functionality simulating Agent's production usage. The only difference is that the installer does not install the Agent, and communication with the ActiveMQ Artemis server is not encrypted. The integration tests are written for each technology supported by the Agent.

In figure 5.1 and 5.2 we can see the overview of Agent tests on the Jenkins automation server.

5.2 Documentation

All code that was implemented as a part of this work is completely documented with Javadoc. Comments enrich the most complicated parts to make them easier to understand. The whole design is documented on MANTA's internal Confluence website. Also, documentation for programmers about how to add an extractor to the Agent was created. One more manual that was created is a manual for users on how to use the Agent to perform an extraction with it.

5.3 Continuous Integration/Continuous Delivery

Builds are performed periodically on Jenkins and they are triggered by each new commit committed to the GIT server. When builds are finished, artifacts are deployed on Nexus server. In figure 5.3 we can see the Nexus repository website.

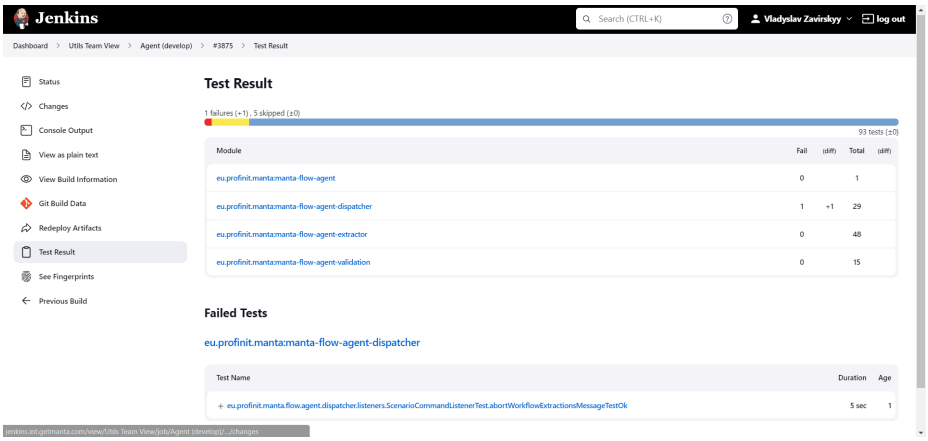


Figure 5.1 Agent tests overview in Jenkins

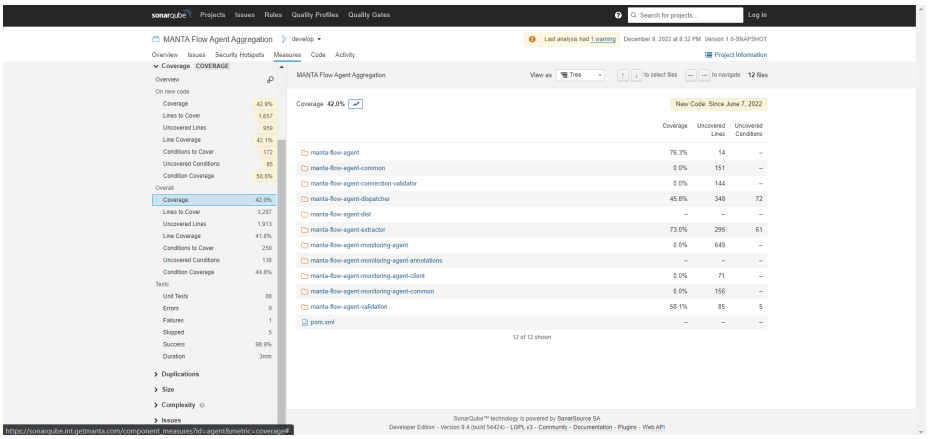


Figure 5.2 Agent tests overview in Jenkins

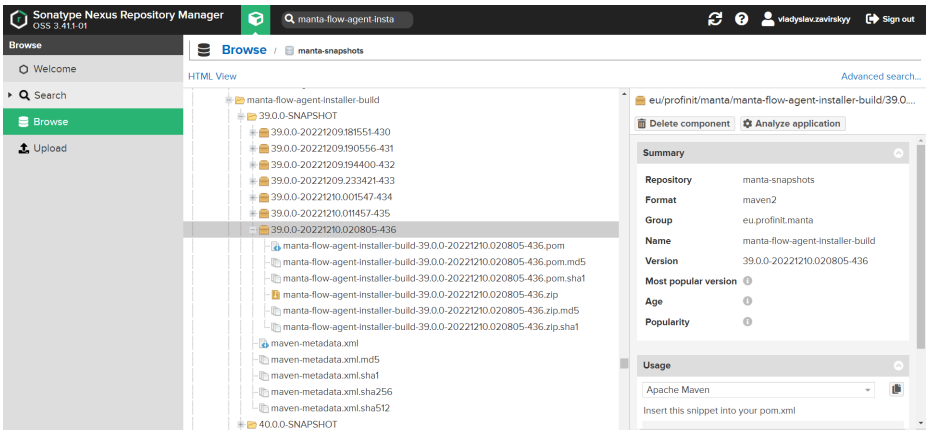


Figure 5.3 Nexus repository

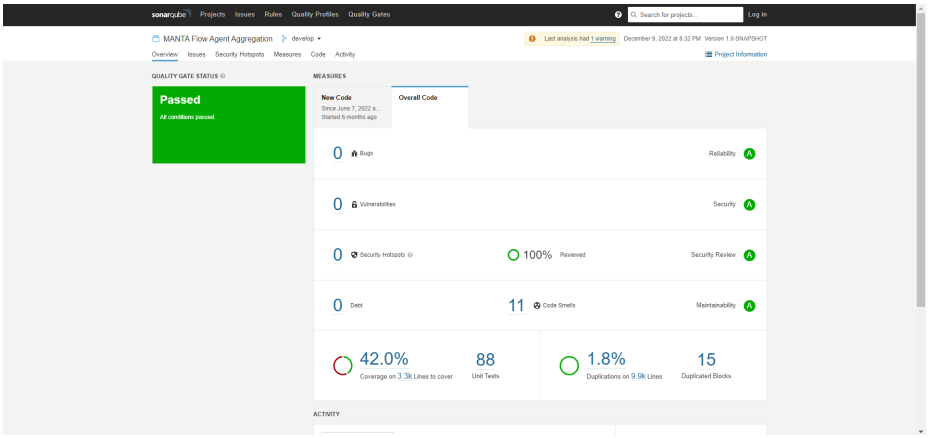


Figure 5.4 SonarQube overview of Agent

The SonarQube tool is used in MANTA for code quality assurance. The SonarQube tool performs continuous inspection of code quality. It perform automatic reviews with static analysis of code to detect bugs and code smells. Also, it is counts code coverage. In figure 5.4 we can see the SonarQube overview of Agent application.

Chapter 6
Conclusion

The MANTA ecosystem was analyzed together with its processes during the preparation period before actually designing and implementing the solution.

As a final solution, an in-house developed Thick Agent application was selected. The Thick Agent allows us to prepare extracted metadata before sending them to the MANTA Flow and perform metadata parsing in simple cases. In-house developed Thick Agent allows us to be more flexible with it and tailor its implementation to our needs. It will allow us to not depend on third-party organizations with such an essential component of the MANTA Flow tool. In case of problems with the Agent in production, they may be solved faster without communicating them with the third-party company.

Similar problems definitely may be solved differently, and the approach chosen within this work is one of many possible. Some general solution by a third-party company may be enough for this, especially if the extraction component is not critical for the company.

The Agent application was designed and implemented based on the existing initial analysis. Together with the Agent, integration of the Agent into the existing MANTA infrastructure was designed and implemented. Namely, the following integration was done: the Agent management through Admin UI Agent Manager, the extraction output parsing for selected technologies by Extraction Processor Parser, extraction configuration with Admin UI Configurator, and extraction orchestration with Admin UI Process Manager.

Both user and programmer documentation for all modules was prepared. Together with it, a user manual about how to use the Agent for metadata extraction and a programmer manual about how to add a new extractor to the Agent was prepared. All modules were tested with unit tests, and integration tests that test the whole metadata extraction flow were implemented.

All functional and non-functional requirements listed in section 2.1 have been implemented and met.

The Agent developed as a part of this thesis is tailored for the MANTA needs, but after some generalization of its code, it should not be a problem to use the Agent not only within the MANTA context.

Currently, the Agent is under active development by the MANTA team, as many extractors should be added there. The Agent is gradually replacing the MANTA Flow CLI extraction functionality. The Agent is now part of the MANTA Flow platform, and it is bundled into it. Metadata extraction functionality for new scanners is already added only to the Agent and is not added to the MANTA Flow CLI, which is definitely a good indicator of the Agent's importance in the MANTA platform. The current official release date for the Agent is Q2 of 2023. The Agent should also be distributed as a docker image in the future.

The Agent will always be under development as the MANTA platform supports more and more technologies with each release, so more and more extractors will be added to the Agent in

the future.

Appendix A

Acronyms

- XML – Extensible markup language
- ETL – Extract Transform Load
- IDE – Integrated Development Environment
- SQL – Structured Query Language
- PL/SQL – Procedural Language/Structured Query Language
- CSV – Comma-separated values
- PID – Process identifier
- JVM - Java virtual machine

Bibliography

1. *About Us — Automatically Scans Your Data Environment* [MANTA] [online]. 2020-03-27. [visited on 2022-10-16]. Available from: https://getmanta.com/about-us/.
2. *What is Data Lineage? - Definition from Techopedia* [Techopedia.com] [online]. [visited on 2022-10-16]. Available from: http://www.techopedia.com/definition/28040/data-lineage.
3. KRATKY, Tomas. *Data Lineage in 2022 Guide — Most Updated Guide* [MANTA] [online]. 2022-09-19. [visited on 2022-10-21]. Available from: https://getmanta.com/ultimate-guide-to-data-lineage/.
4. *What is data lineage and why is it important?* [Collibra] [online]. [visited on 2022-10-29]. Available from: https://www.collibra.com/us/en/blog/what-is-data-lineage.
5. *Use Cases & Case Studies* [MANTA] [online]. [visited on 2022-10-30]. Available from: https://getmanta.com/blog/category/use-cases-case-studies/.
6. ZAVIRSKYY, Vladyslav. *Design and implementation data flow analysis of jobs in IBM DataStage for Manta project*. Prague, Czech Republic, 2019. Available also from: https://dspace.cvut.cz/handle/10467/83016. Bachelor's thesis. Czech Technical University, Faculty of Information Technology.
7. MARTIN, Matthew. *Functional vs Non Functional Requirements* [online]. 2020-02-15. [visited on 2022-11-17]. Available from: https://www.guru99.com/functional-vs-non-functional-requirements.html.
8. BC. FIRMENT, Roman. *Monitoring Support for Manta Flow Agent in Cloud-Based Architecture*. Prague, Czech Republic, 2022. Available also from: https://dspace.cuni.cz/bitstream/handle/20.500.11956/173531/120418500.pdf?sequence=1. Master's thesis. Charles University, Faculty of mathematics and physics.
9. *Logical Architecture • Talend Cloud Physical Reference Architecture • Reader • Welcome to Talend Help Center* [online]. [visited on 2022-11-17]. Available from: https://help.talend.com/r/en-US/Cloud/talend-cloud-physical-reference-architecture/logical-architecture.
10. *Working with Talend Remote Engines • Talend Remote Engine User Guide for Linux • Reader • Welcome to Talend Help Center* [online]. [visited on 2022-11-17]. Available from: https://help.talend.com/r/en-US/Cloud/remote-engine-user-guide-linux/remote-engine-ug.
11. *Hybrid Cloud Network Connections - AWS Direct Connect - Amazon Web Services* [Amazon Web Services, Inc.] [online]. [visited on 2022-11-17]. Available from: https://aws.amazon.com/directconnect/.

12. *AWS Direct Connect - Amazon Virtual Private Cloud Connectivity Options* [online]. [visited on 2022-11-17]. Available from: https://docs.aws.amazon.com/whitepapers/latest/aws-vpc-connectivity-options/aws-direct-connect.html.
13. *What Is a VPN? - Virtual Private Network* [Cisco] [online]. [visited on 2022-11-17]. Available from: https://www.cisco.com/c/en/us/products/security/vpn-endpoint-security-clients/what-is-vpn.html.
14. MIQUELLADEBOER. *On-premises data gateway architecture* [online]. [visited on 2022-11-17]. Available from: https://learn.microsoft.com/en-us/data-integration/gateway/service-gateway-onprem-indepth.
15. BERNIER, Hugo. *Accessing Your On-Premises Data Using the On-Premises Data Gateway* [Tahoe Ninja] [online]. [visited on 2022-11-17]. Available from: https://tahoeninja.blog/posts/accessing-your-on-prem-data-using-on-prem-data-gateway/.
16. SERVERLESS360. *Azure Logic Apps Integration Service Environment* [Serverless360] [online]. 2018-11-07. [visited on 2022-11-17]. Available from: https://www.serverless360.com/blog/azure-logic-apps-integration-service-environment.
17. *Data Integration Tools, Connectors & Solutions — Progress DataDirect* [Progress.com] [online]. [visited on 2022-11-17]. Available from: https://www.progress.com/datadirect-connectors.
18. *What is the difference between MANTA, MANTA Live, MANTA Flow, MANTA Admin UI, and Open MANTA?* [MANTA] [online]. [visited on 2022-10-21]. Available from: https://getmanta.com/faq/how-to-use-manta/what-is-the-difference-between-manta-manta-live-manta-flow-manta-admin-ui-and-open-manta/.
19. BC. GONDEK, Petr. *Orchestration and Monitoring of Manta Flow Processes*. Prague, Czech Republic, 2020. Available also from: https://dspace.cvut.cz/handle/10467/89915. Master's thesis. Czech Technical University, Faculty of Information Technology.
20. *What is an IDE?* [online]. [visited on 2022-11-04]. Available from: https://www.redhat.com/en/topics/middleware/what-is-ide.
21. *What is Java and why do I need it?* [online]. [visited on 2022-11-04]. Available from: https://www.java.com/en/download/help/whatis_java.html.
22. *Maven – Introduction* [online]. [visited on 2022-11-04]. Available from: https://maven.apache.org/what-is-maven.html.
23. *JUnit 5* [online]. [visited on 2022-11-04]. Available from: https://junit.org/junit5/.
24. *Spring Boot* [online]. [visited on 2022-11-04]. Available from: https://spring.io/projects/spring-boot#overview.
25. *ActiveMQ* [online]. [visited on 2022-11-04]. Available from: https://activemq.apache.org/components/artemis/.
26. KOVAC, Jakub. *Design and prototype implementation of a logging framework for Manta Flow*. Prague, Czech Republic, 2020. Available also from: https://dspace.cvut.cz/handle/10467/87669. Bachelor's thesis. Czech Technical University, Faculty of Electrical Engineering.
27. *What is JavaScript? - Learn web development — MDN* [online]. [visited on 2022-11-04]. Available from: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript.
28. *React – A JavaScript library for building user interfaces* [online]. [visited on 2022-11-04]. Available from: https://reactjs.org/.
29. KALAR, Erika. *How Liquibase Works — Advanced Database Schema Change — Liquibase* [Liquibase.com] [online]. [visited on 2022-11-04]. Available from: https://www.liquibase.com/how-liquibase-works.

30. *H2 Database Engine* [online]. [visited on 2022-11-04]. Available from: https://www.h2database.com/html/main.html.

31. *mybatis – MyBatis 3 — Introduction* [online]. [visited on 2022-11-04]. Available from: https://mybatis.org/mybatis-3/.

32. *Cross Platform VMware InstallBuilder: Multiplatform Installer Tool* [online]. [visited on 2022-12-02]. Available from: https://installbuilder.com/.

33. NEWMAN, Sam. *Building Microservices: Designing Fine-Grained Systems*. 1st edition. Beijing Sebastopol, CA: O'Reilly Media, 2015. ISBN 978-1-4919-5035-7.

34. *MicroServices - How To Share DTO (Data Transfer Objects)* [Vinsguru] [online]. [visited on 2022-11-29]. Available from: https://www.vinsguru.com/microservices-architecture-how-to-share-dto-data-transfer-objects/. Section: Architecture.

35. BAELDUNG. *How to Share DTO Across Microservices — Baeldung* [online]. 2020-06-27. [visited on 2022-11-29]. Available from: https://www.baeldung.com/java-microservices-share-dto.

36. *Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE [Book]* [online]. [visited on 2022-12-02]. Available from: https://www.oreilly.com/library/view/advanced-api-security/9781430268178/. ISBN: 9781430268178.

Content of the attached media

- `readme.txt` ..... a brief description of the content of the medium
- `exe` ..... directory with executable form of implementation
- `src`
  - `impl` ..... implementation source codes
  - `thesis` ..... source form of the work in format LaTeX
- `text` ..... the text of the thesis
  - `thesis.pdf` ..... text of the thesis in PDF format