St Petersburg University

SPD NICA

SPD On-line Filter: workflow and data management systems.

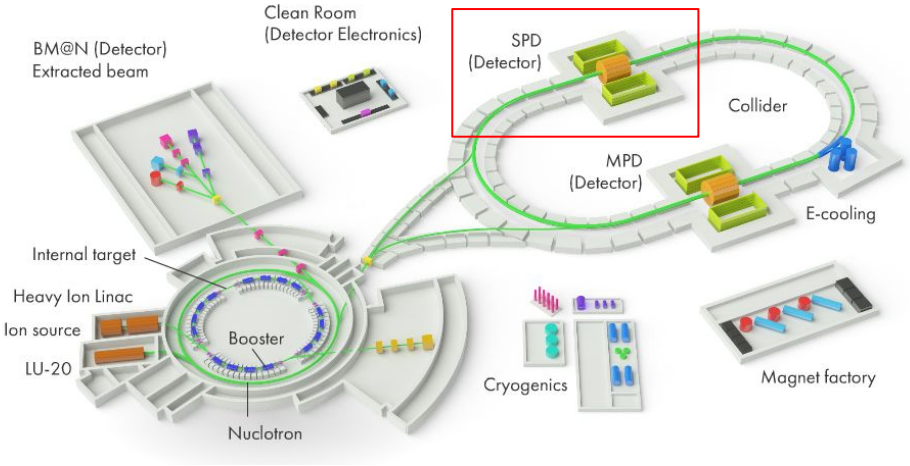
Tereschenko D.V., Ponomarev E.V., Oleynik D.A., Korkhov V.V.

St. Petersburg State University

GRID’23

SPD experiment at NICA collider

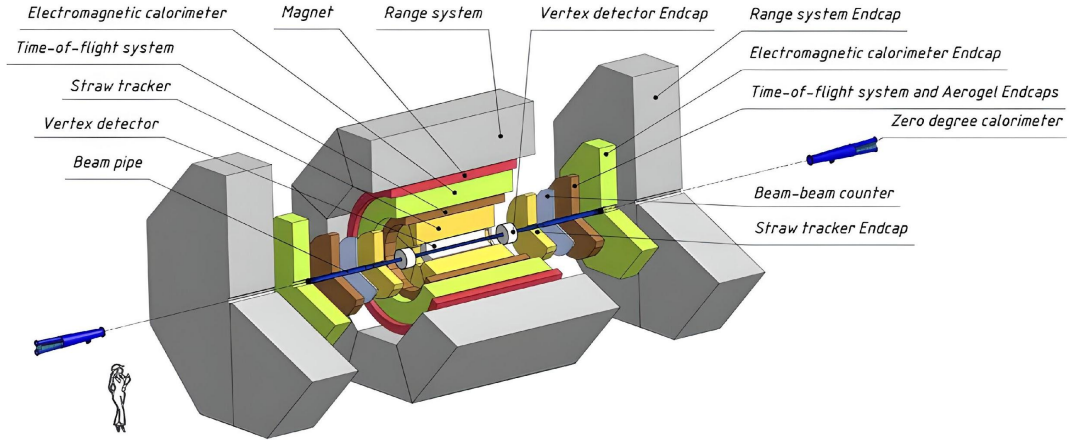
One of the strategically important infrastructure projects, from the point of view of the long–term scientific plan of JINR, is the NICA complex for spin physics on polarized beams - the SPD detector (Spin Physics Detector).



NICA Complex

- Polarized proton and deuteron beams
- Collision energy up to 27 GeV
- luminosity up to 10^{32} cm^{-2} s^{-1}
- Bunch crossing every 80 ns = crossing rate 12.5 MHz

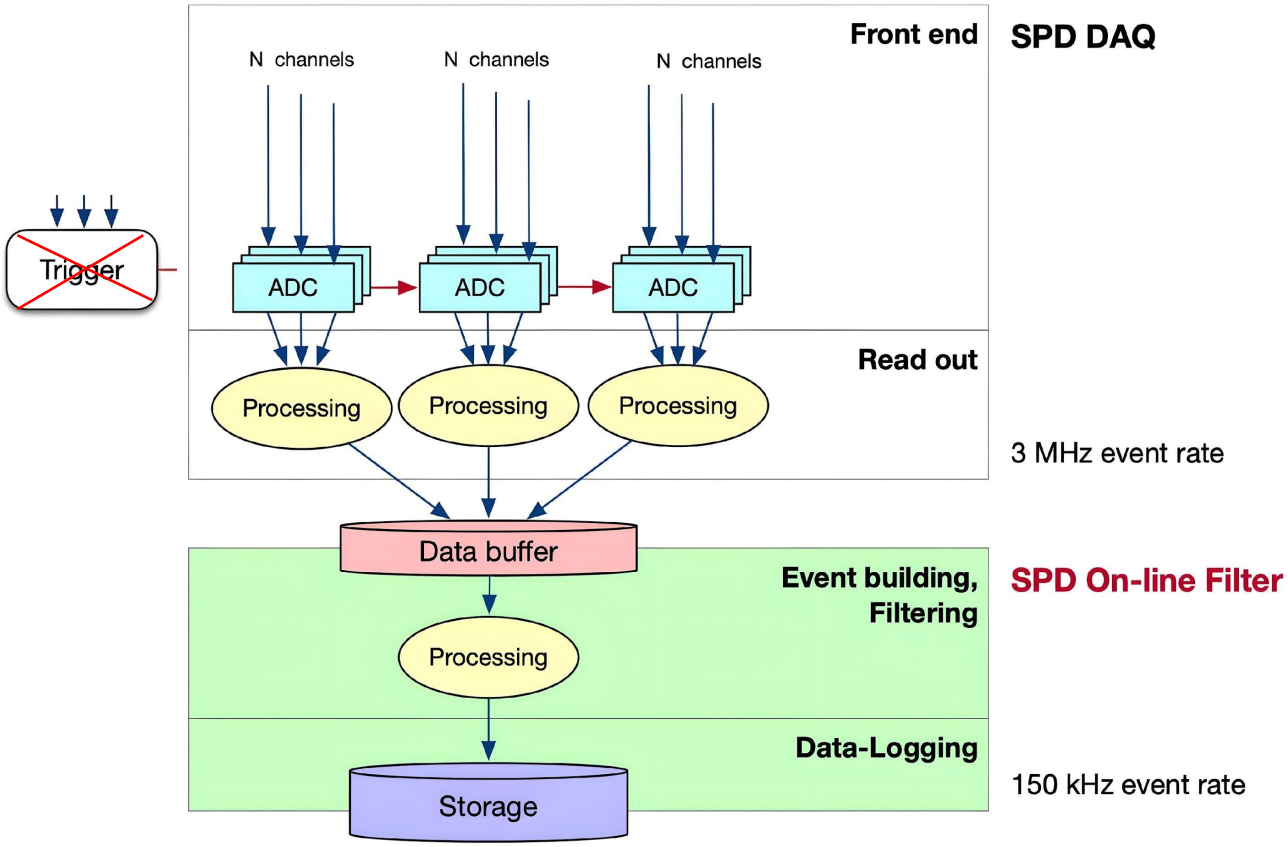
- Number of registration channels in SPD ~ 500000
- ~ 3 MHz event rate (at max luminosity) = pileups
  - ~ 20 GB/s (or 200PB/year) “raw” data
- Selection of physics signal requires momentum and vertex reconstruction
  - => no simple trigger is possible



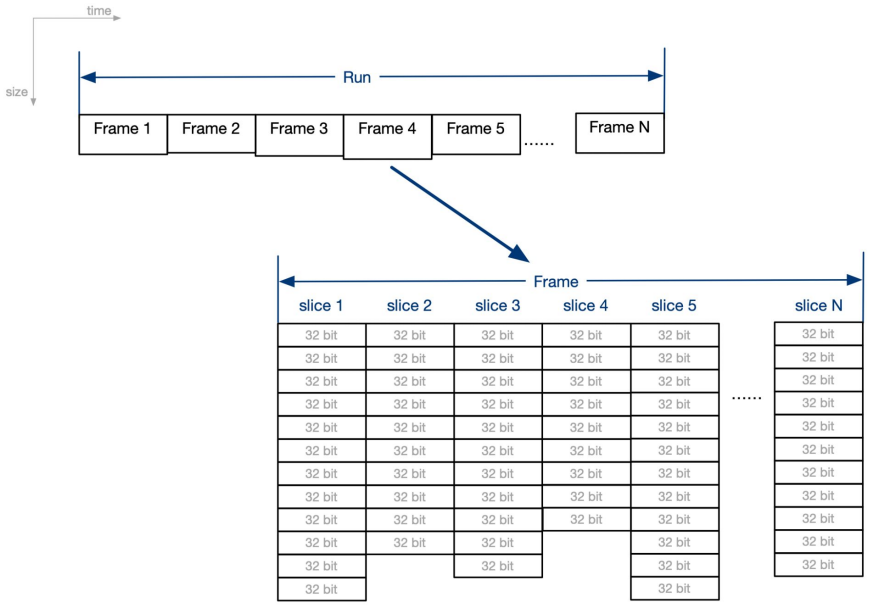
SPD facility

Trigger less DAQ on SPD

Triggerless DAQ, means that the output of the system will not be a dataset of raw events, but a set of signals from sub- detectors organized in time slices



«triggerless» DAQ

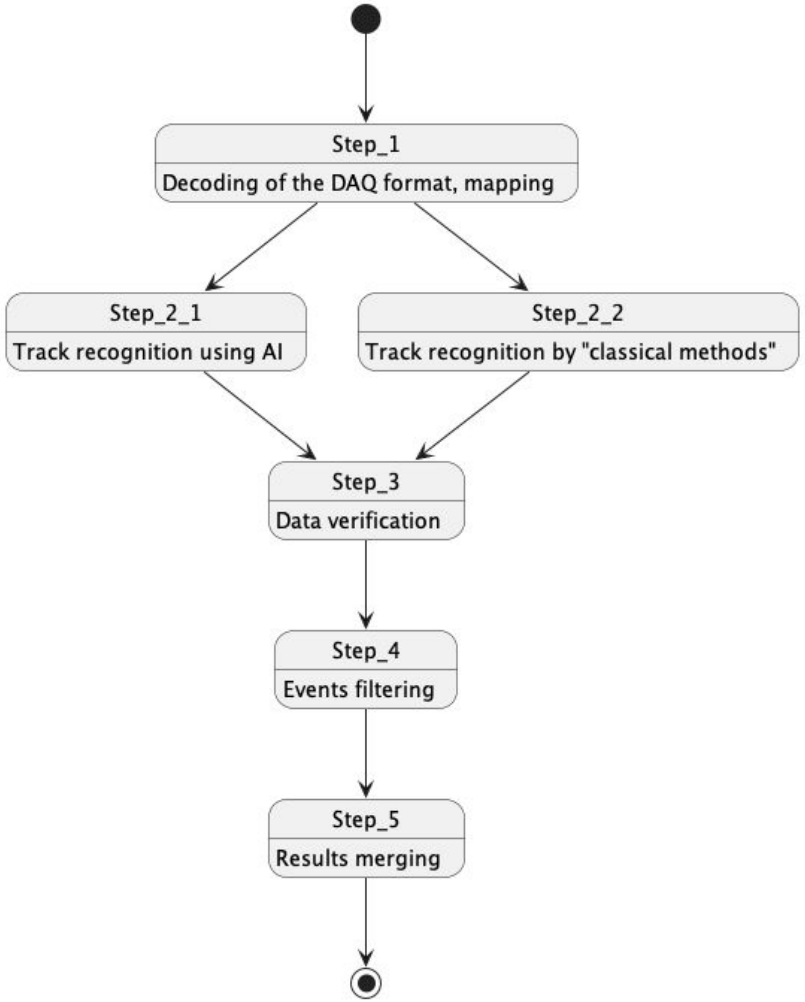


Struct of data from «triggerless» DAQ

- DAQ provide data organized in time frames which placed in **files** with reasonable size (a few GB)
- Each of these file may be processed independently as a part of top-level **workflow chain**
- No needs to exchange of any information during handling of each initial file, but results of may be used as input for next step of processing.

SPD OnLine Filter - High-throughput Computing system

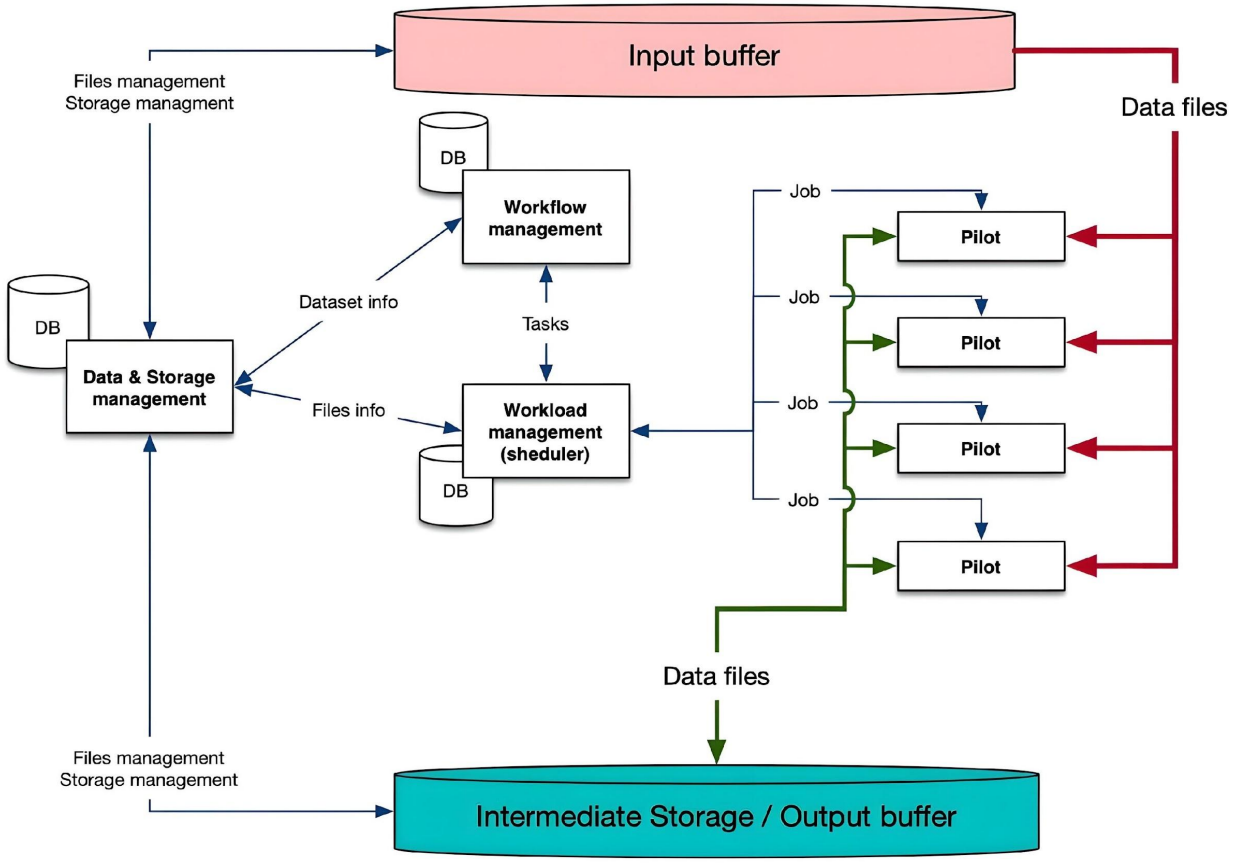
- **HTC** is defined as a type of computing that simultaneously executes numerous simple and computationally independent jobs to perform a data processing task.
- It can be applied for data aggregated by SPD DAQ because each particular frame can be processed **simultaneously**.
- Data processing should be **multi-staged** for providing of efficient usage of computing resources



Simplified multi-staged scheme

Middleware complex for SPD OnLine Filter

«SPD OnLine filter» – a hardware and software complex that provides multi-stage high-throughput processing and filtering of data for the SPD detector .



Architecture of SPD OnLine Filter

Data management

- Support of data lifetime (catalog, consistency check, cleanup, storage)

Workflow management

- Top level system
- Allow to define processing chains (workflows)
- Manage workflows execution
- Manage datasets

Workload management:

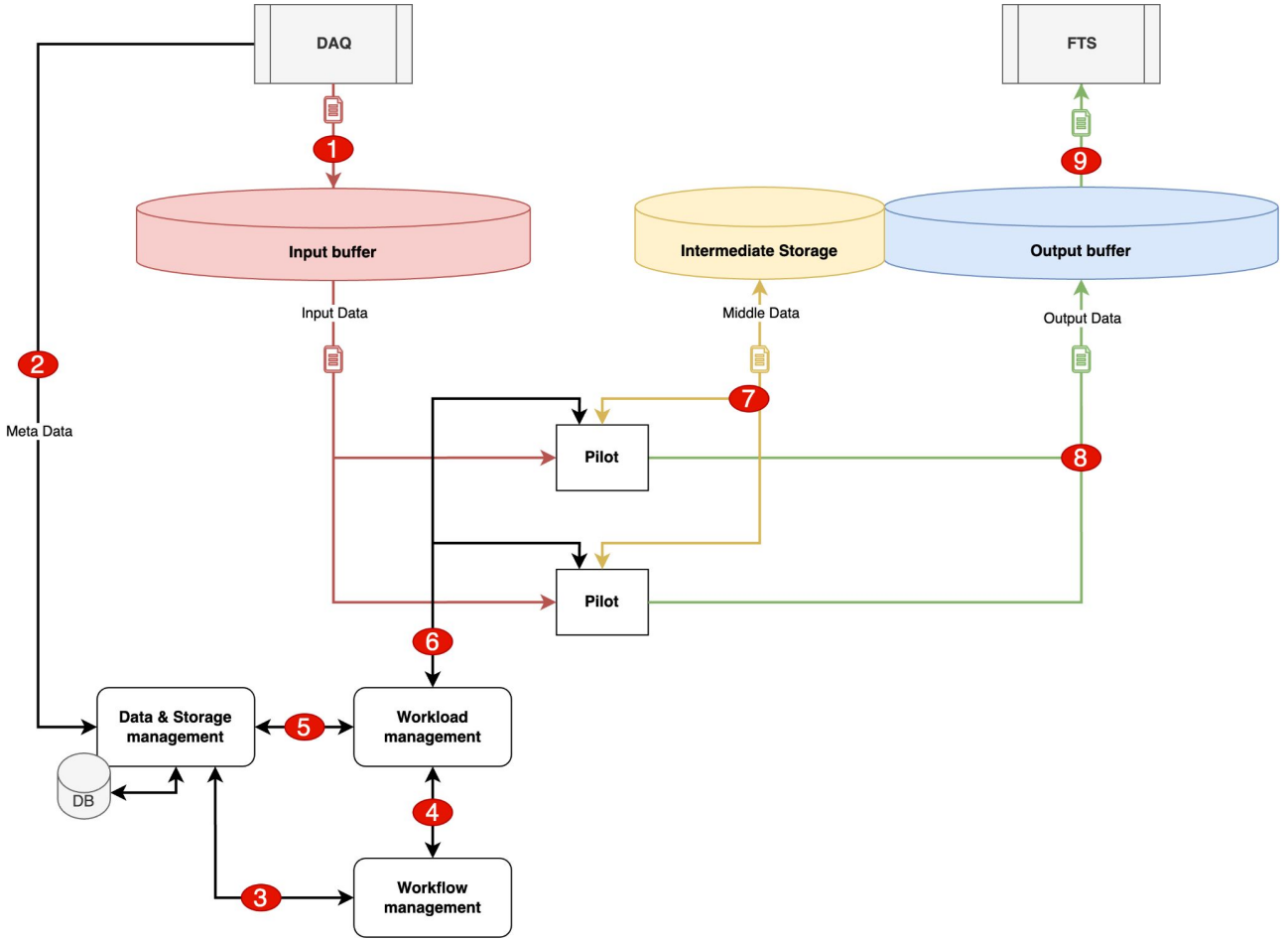
- Splitting of task to jobs;
- Dispatch jobs to pilots;
- Control of jobs executions;
- Control of pilots (identifying of "dead" pilots)

Data flow and concept of data processing

Main concept: High throughput data processing of independent blocks of data

Main data flows:

- SPD DAQ, after splitting into time blocks of signals from sensors, send data to the **input buffer** of SPD OnLine Filter as files of an agreed size
- The workflow management system creates and deletes intermediate and final **datasets**
- The workload management system **“fills” the datasets** with information about the resulting files
- At each step of data processing, Pilots will read and write files to the storage, create **secondary data**



Main data flows of SPD OnLine Filter

Requirements

The main requirement – the systems must meet the **high–throughput paradigm**.

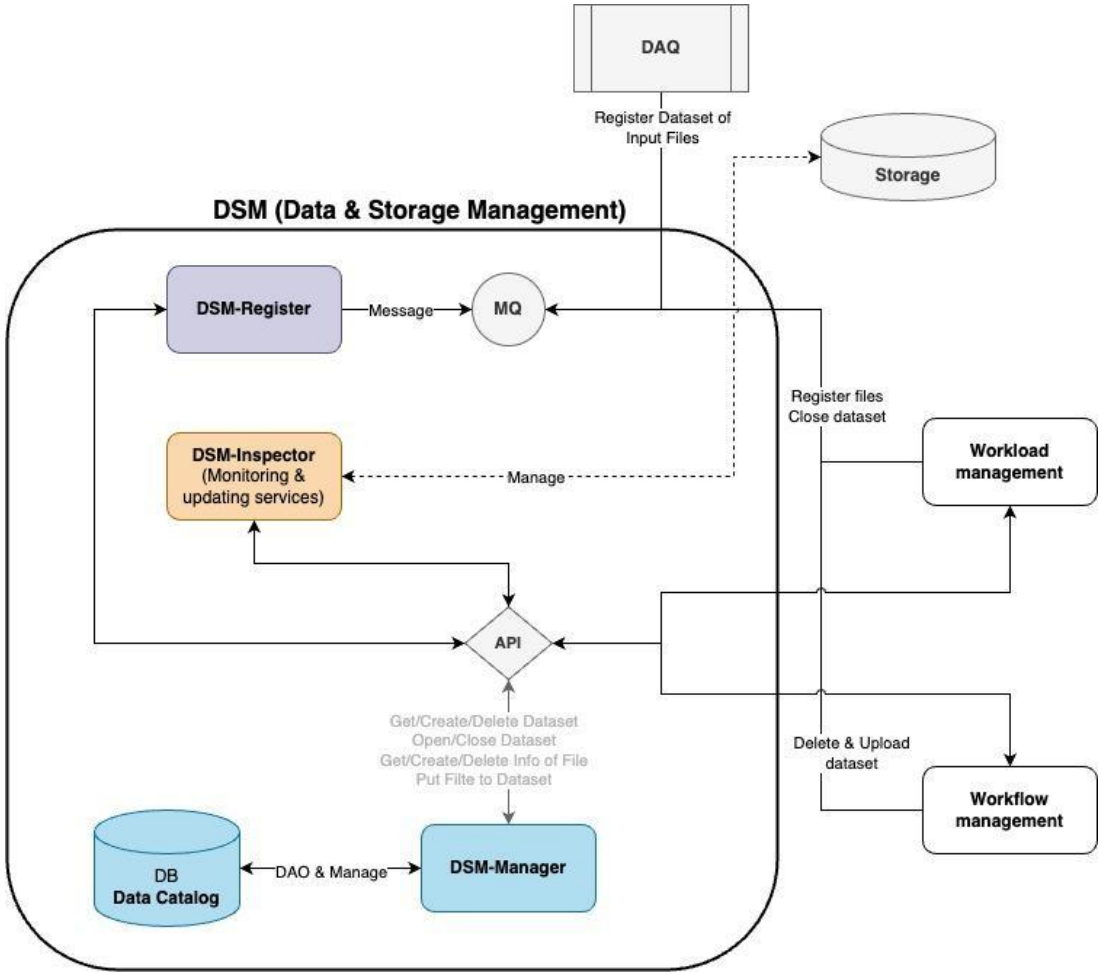
Functional requirements

Data management	Workflow management
❑ Abstracting from the data format of the DAQ; ❑ The possibility of logical grouping of data (not relevant to the level of physical storage); ❑ Lack of redundancy in the organization of datasets (control of unnecessary replicas); ❑ Separation of metadata from physical data storage (data catalog); ❑ Accounting for the state of data from the point of view of the processing process; ❑ Control of the consistency of information in the catalog in relation to the input and output storage.	❑ Define workflows which represents multi-stage processing ❑ Organization of data processing sequences (chains) ❑ Formation of a request for data processing according to a certain sequence ❑ Processing request execution

Technical requirements cannot be setup at the current stage. But in parallel, a simulation model of the installation is being built, which should allow determining the pre-production characteristics of the entire system.

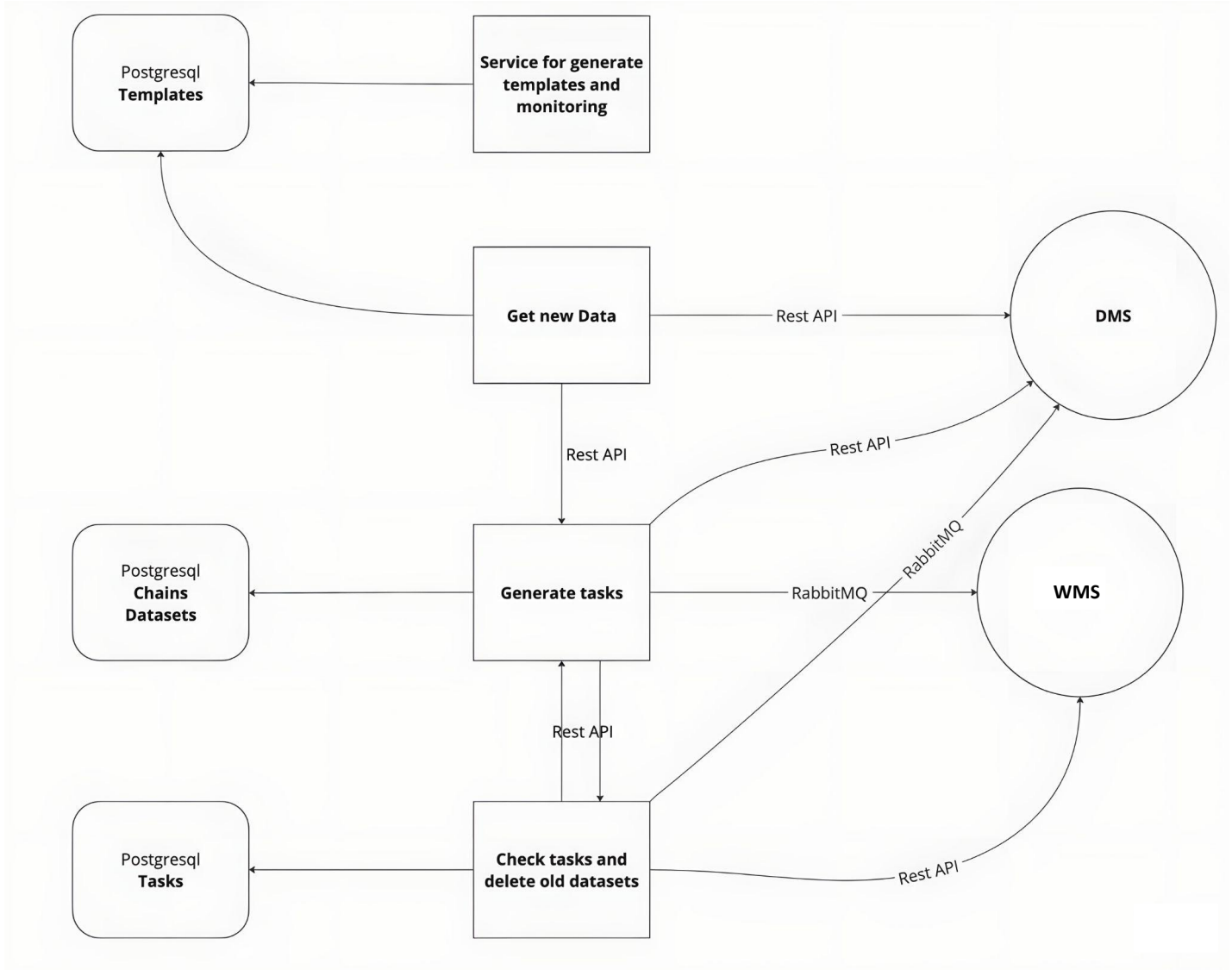
Architecture and functionality of Data Management

- **DSM-Register (Data Registration)**: A service that receive requests for adding/deleting data in the system asynchronously (via MQ). Then the service makes changes to the data catalog via the API of the *dsm-manager*
- **DSM-Manager (REST API of data catalog):**
  - File management: get info of organization the system data
  - Dataset management: create a dataset, add a file to the dataset, close the dataset; delete the dataset; provide information of contents of the dataset (files in the dataset)
- **DSM-Inspector (Daemon tasks)**: delete files on storage, check consistency of files, monitoring the use of storage (for example, "dark" data)



Architecture of Data Management

Architecture and functionality of Process Management



Services:

- Service with API for create templates of data processing chains
- Service for get new data from DMS
- Service for generate tasks and send to WMS
- Service for monitoring status of tasks

Databases:

- Store of templates
- Store of chains
- Store of tasks

Design of databases (Data Catalog)

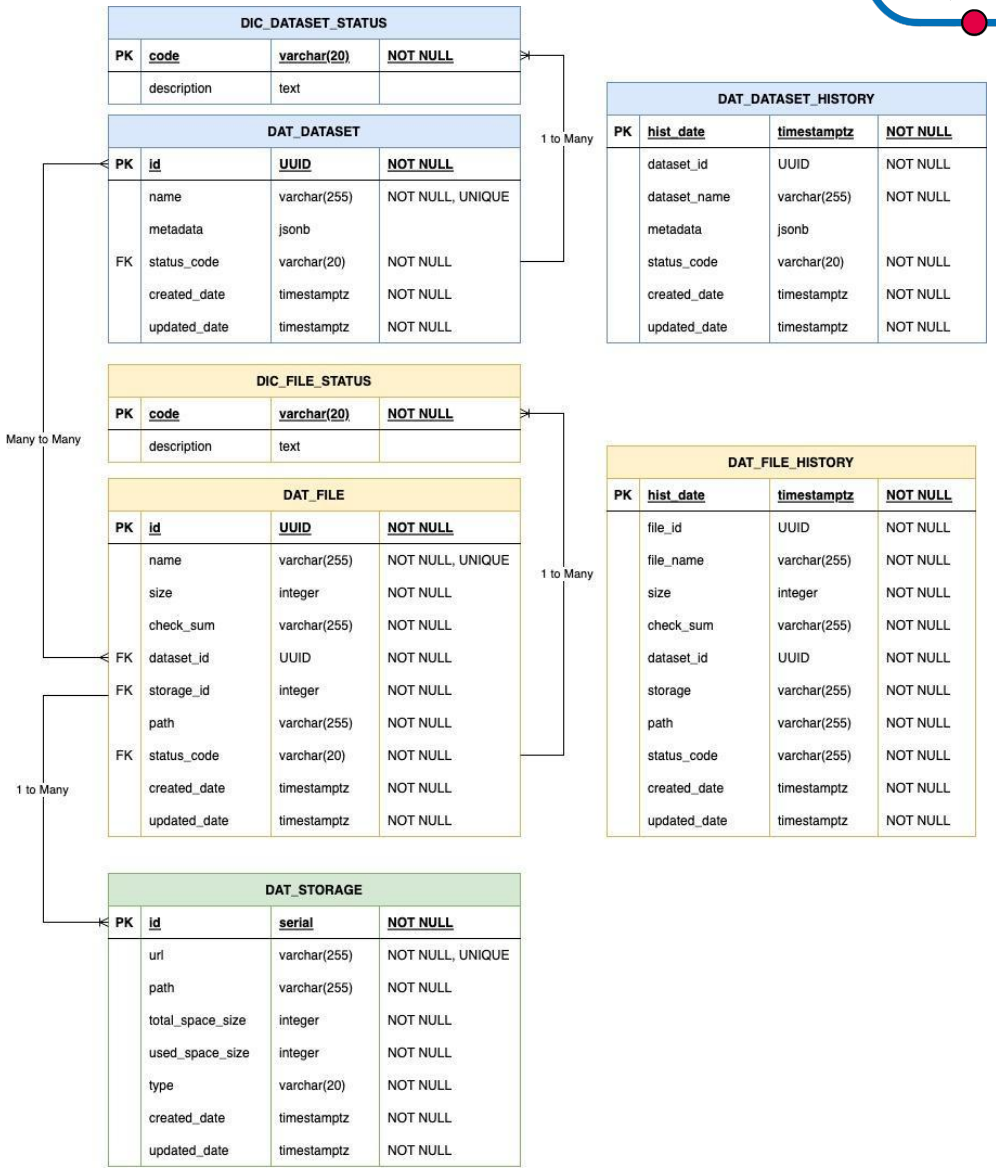
DBMS - relational database PostgreSQL 12

Tables:

- **DAT_FILE** - catalog of files
- **DAT_DATASET** - catalog of datasets
- **DAT_FILE_HISTORY и DAT_DATASET_HISTORY** - archive data
- **DIC_FILE_STATUS и DIC_DATASET_STATUS** - dictionary of statuses
- **DAT_STORAGE** - info of storages

Extra mechanisms:

- **Trigger** on catalogs – save data lifecycle for further monitoring
- **Indexes** on filter fields for optimization of operations
- **Partition** of history tables for optimization of indexes
- **Unique constraints** for support data rules on DB layer



ER-diagram of Data Catalog

Design of services (key results)

- ✔ Design of signature and work algorithm for all message gateways;
- ✔ Design of list necessary REST API methods and their signature (for example, CRUD and filter operation on data catalog);
- ✔ Design of work algorithm daemon tasks (usage statuses from data catalog)
- ✔ Design of scenarios of inter-service interaction

Prototype of services (key results)

- ✔ Choose required Tech Stack
- ✔ Developed the logic of build app based on IoC pattern
- ✔ Developed the mechanism for declare model of data in database based on ORM and migration scripts
- ✔ Developed the basic CRUD operations on data as REST API
- ✔ Developed the mechanism for declare message queues of RabbitMQ. Also setup of dead letter queues.
- ✔ Setup of tools for CD (building and deploying) on JINR LIT infrastructure.

Now all services deploying. Testing in progress.

Tech Stack

Common ➔ Python 3.11 ➔ Docker ➔ Docker-compose	**Frameworks** ➔ FastAPI + Unicorn ➔ Pika (RabbitMQ) ➔ Celery (Daemon tasks)
DB ➔ SQLAlchemy (PostgreSQL) ➔ Alembic (Migration) ➔ Asyncio + Aiopg	**Extra** ➔ Aiohttp ➔ Aiofiles ➔ Pydantic ➔ Pickle

Use cases

Developed use cases for feature full-scale testing of the entire system.

- Receiving input files from DAQ
  - Action: Manually place files in the input directory. Send information about input files to the dsm-register.
  - Expected result: The dsm-register service received the message successfully. The data is located in the data catalog (check via the REST API of dsm-manager): information about the input dataset, as well as a list of all files.
- Starting of process (receive new data, creating tasks by chains)
- Opening the new datasets
- Registering the new files into open datasets
- Closing the datasets after filling
- Deleting the not actual datasets
- Uploading the the resulting dataset

Conclusion

As a result, for process and data management systems in «SPD OnLine filter»:

- Choose required architecture of the systems;
- Designing the internal components of each systems;
- Designing internal and external interfaces of interaction;
- Choose Tech stack;
- Partial prototyping;
- Deploy on JINR LIT infrastructure and preparing for testing.

Next, we plan to finish prototyping the system, test the system, and then proceed to integrate the system with the application software on the SPD installation.

Доп. Проектирование DSM-Register

*Сервис должен слушать очередь сообщений **RabbitMQ** и обрабатывать заявки на добавление/удаление данных в системе.*

Exchange	**Routing Key**	**Msg**	**Алгоритм**
dsm.register (direct)	file.input	Информация о файлах во входном наборе • Путь до хранилища • Путь на хранилище • Имя файла • Размер • Контрольная сумма	Создаём набор по номеру frame-а со статусом OPEN. Добавляём файлы с привязкой к этому набору, устанавливаем первичный статус CREATE.
	file.process	Информация о наборе • Имя набора / UID набора Информация о файлах в наборе (опционально) • –//– • ID хранилища	Создаём набор (если его нет) в статусе OPEN, добавляем файлы в набор, устанавливаем статус CREATE.
	dataset.close	Информация о наборе • UID набора • Контрольный список файлов (имена файлов)	Запрашиваем зарегистрированные файлы в наборе. Если совпадает с контрольным списком, то устанавливаем статус CLOSED. Иначе возвращаем сообщения обратно в очередь для отложенного исполнения.
	dataset.upload	UID набора	Помечаем набор на выгрузку (статус TO_UPLOAD)
	dataset.delete	UID набора	Помечаем набор на удаление (статус TO_DELETE)

Сигнатура и алгоритм работы шлюзов приёма сообщений для сервиса dsm-register

Доп. Проектирование DSM-Manager

*Сервис, предоставляющий **API** к каталогу данных (размещение данных в каталоге, обращение к каталогу, изменение данных в каталоге)*

Получения данных в унифицированном и структурированном виде:

- Для каждой сущности должен быть задан набор **CRUD операций + операции выборки**
- Сами данные должны выдаваться в формате **JSON**
- Сигнатура интерфейсов должна отвечать архитектурному стилю **REST API**

Функциональность	**API**	**Контракт запроса**	**Контракт ответа**
Internal API • Управление информацией о наборах (CRUD API) • Управление информацией о файлах (CRUD API) • Управление информацией о хранилище (CRUD API) • Получение информации для мониторинга (Filter API)			
External API			
Взаимодействие с системой управления нагрузкой			
Содержание набора	GET /dataset/<id>/file	ID набора	Общее кол-во файлов в наборе (total) + короткая информация о файлах (data)
Информация о хранилище	GET /storage/<id>	ID хранилища	Информация о хранилище
Взаимодействие с системой управления процессом			
Список наборов в определенном статусе	GET /dataset?status=<>&timestamp=<>	Статус набора + Отметка времени	Информация о наборах
Некоторые API управления наборами (создание выходного набора, получение статуса выходного набора)			

Список и сигнатура необходимых REST API методов сервиса dsm-manager

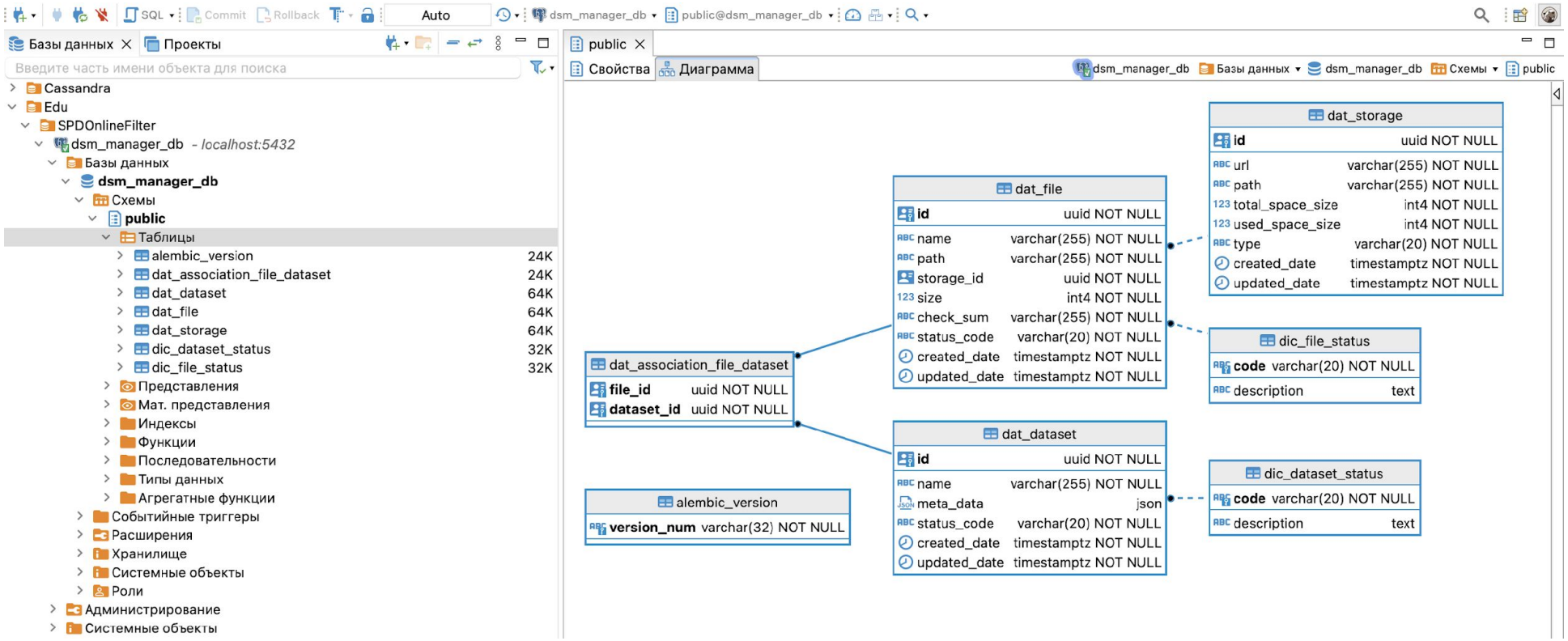
Доп. Проектирование DSM-Inspector (status система)

*Набор фоновых **задач** для мониторинга и контроля состояния данных в хранилище*

- **Удаление файлов на хранилищах**
  - Одним процессом получаем наборы файлов со статусом TO_DELETE. По каждому файлу в наборе проверяем статус.
    - Если есть не DELETED, то ставим статус TO_DELETE.
    - Иначе устанавливаем для набора статус DELETED.
  - Вторым процессом получаем файлы со статусом TO_DELETE. Проверяем, что все его наборы находятся в статусе TO_DELETE. Удаляем файл на хранилище. Устанавливаем статус DELETED.
- **Контроль выгрузки данных во внешнюю систему**
- **Проверка целостности файлов**
- **Контроль использования хранилища**

Доп. Прототипирование DSM-Manager (БД)

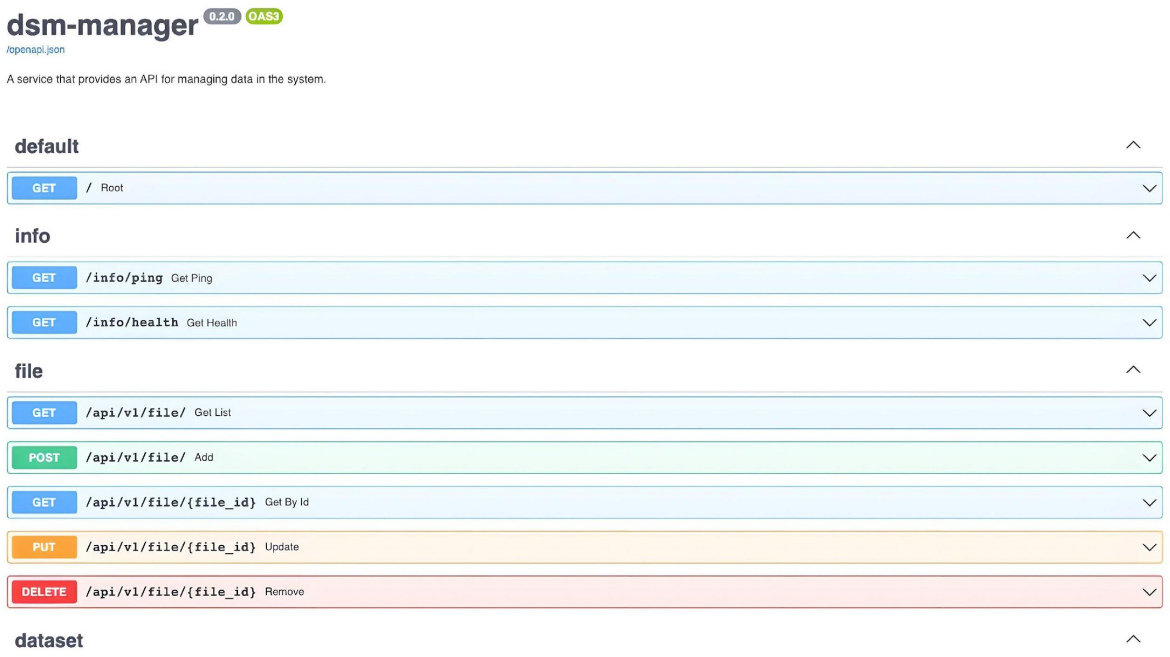
- В качестве СУБД выбрана **PostgreSQL 12**
- Схема БД была определена с помощью **ORM моделей** библиотеки SQLAlchemy (подход “code first”)
- Скрипты с DDL командами для построения таблиц были составлена инструментом построения **миграций** Alembic
- **Развёртка таблиц** в БД осуществлялась с помощью применения скриптов при запуске Docker контейнера



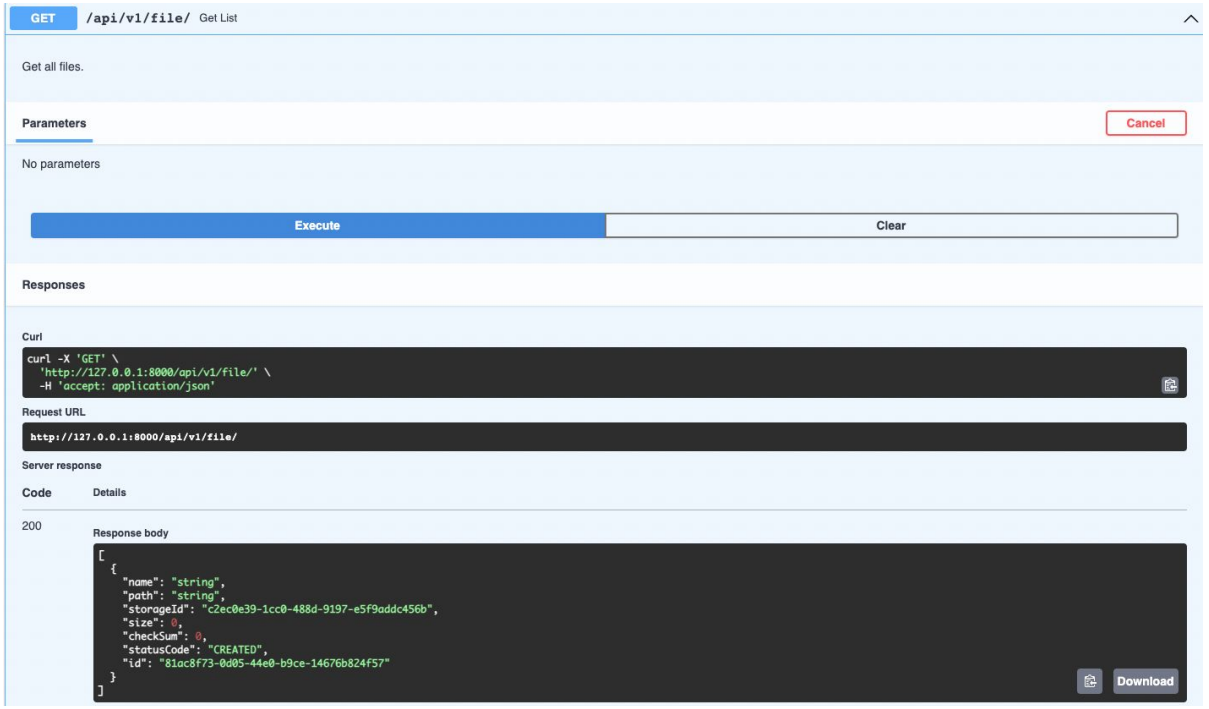
Развёрнутая схема таблиц в БД сервиса dsm-manager (интерфейс Dbeaver)

Доп. Прототипирование DSM-Manager (API)

- Выбран каркас для построения сервиса в виде асинхронного фреймворка **FastAPI + Unicorn**
- Всё построение приложения выполняется на основе одного из принципа SOLID - DIP (dependency inversion principle) и реализуется при помощи инструмента **Dependency Injector**
- Разработан базовый набор CRUD операций над данными в виде REST API.
- Реализована автогенерация описания API по спецификации OpenAPI 3.0 (доступно в интерфейсе **Swagger UI** по адресу *<адрес сервера>/docs*)



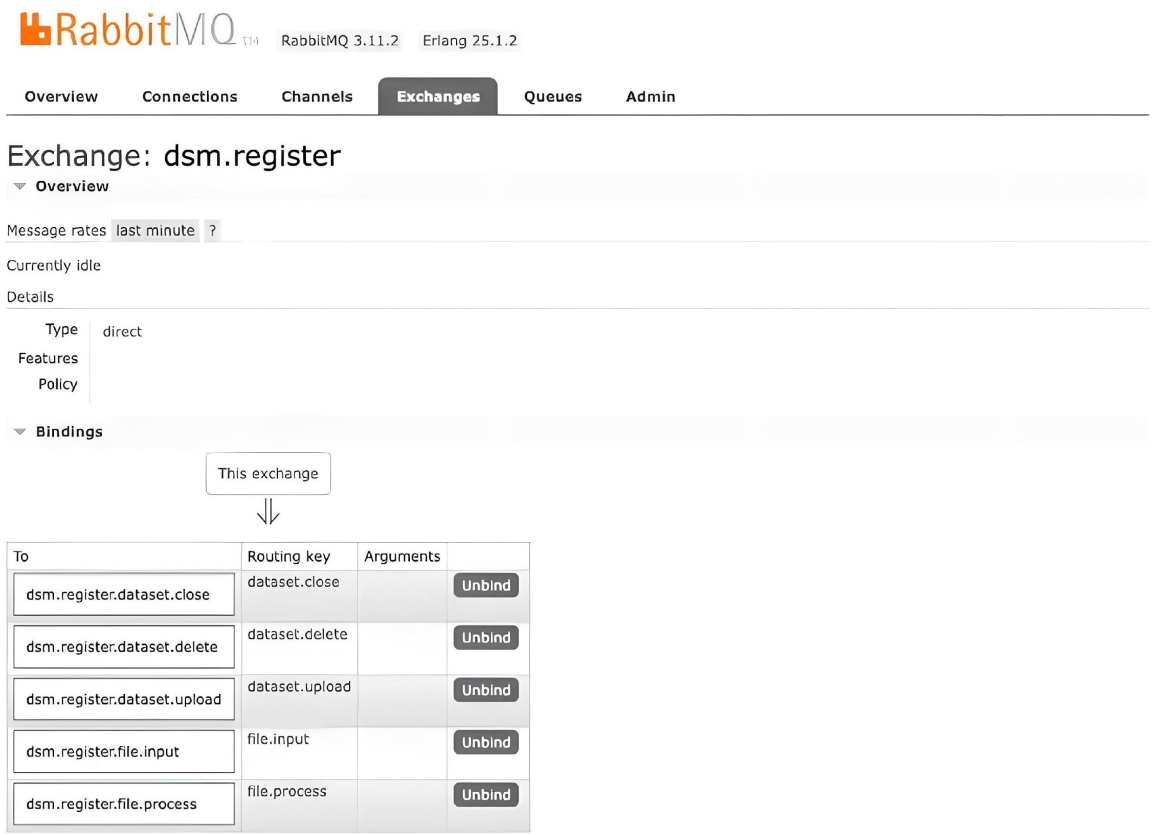
Swagger UI с описанием API сервиса dsm-manager



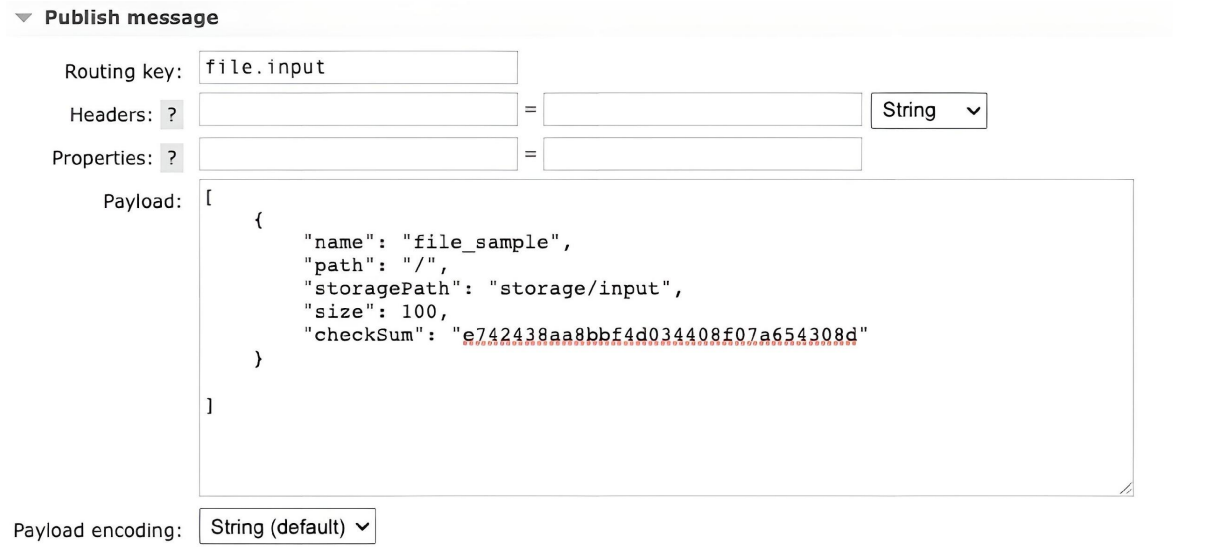
Пример обращение к сервису для получения списка файлов

Дом. Прототипирование DSM Register (Очереди RabbitMQ)

- В качестве брокера сообщений выбран **RabbitMQ**
- **Определение очередей** осуществляется при помощи декларативной нотации инструмента Pika
- На старте приложения идёт их **развёртка**



Сконфигурированные очереди RabbitMQ

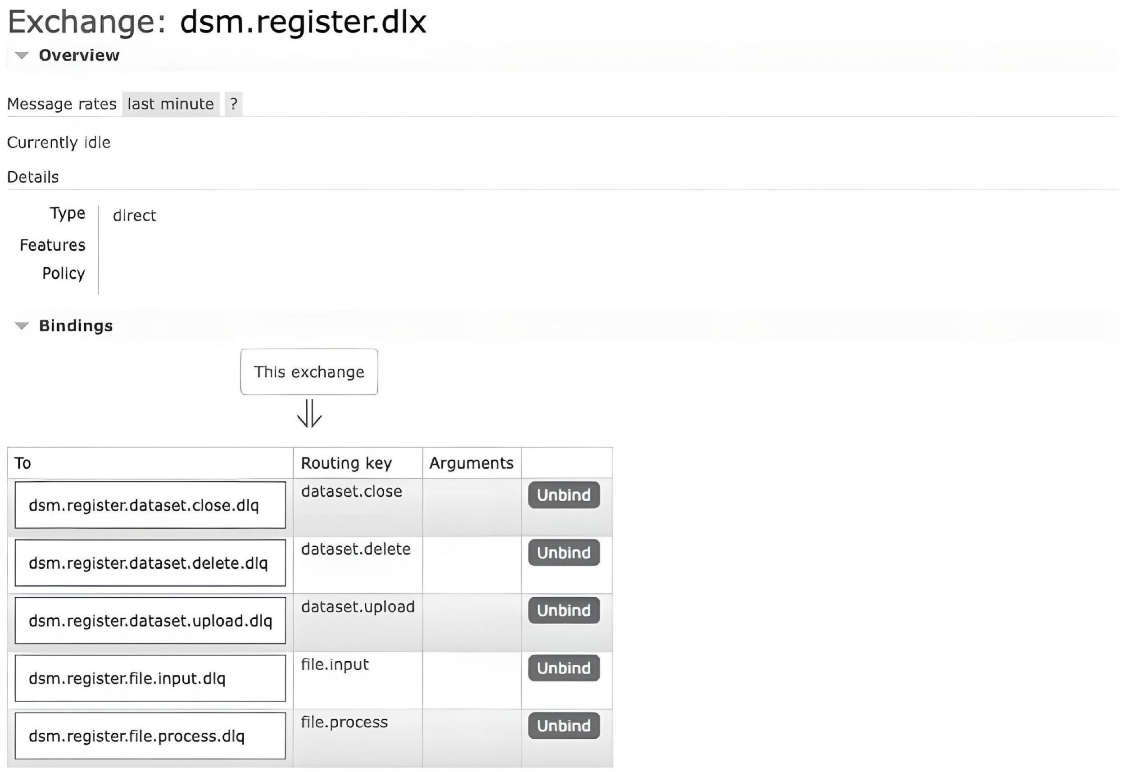


2023-05-10 17:23:28 INFO: [DSM-REGISTER-I004] [CONSUMER-HANDLER] **Received message** [FileInputDto(name='file_sample', path='/', storage_path='storage/input', size=100, check_sum='e742438aa8bbf4d034408f07a654308d')] from queue 'dsm.register.file.input' with delivery tag 1 [in /src/app/executer/rabbit/consumers/base/consumer_handler.py:48

Отправка и приём сообщения в очередь RabbitMQ входных файлов

Доп. Прототипирование DSM-Register (DLQ)

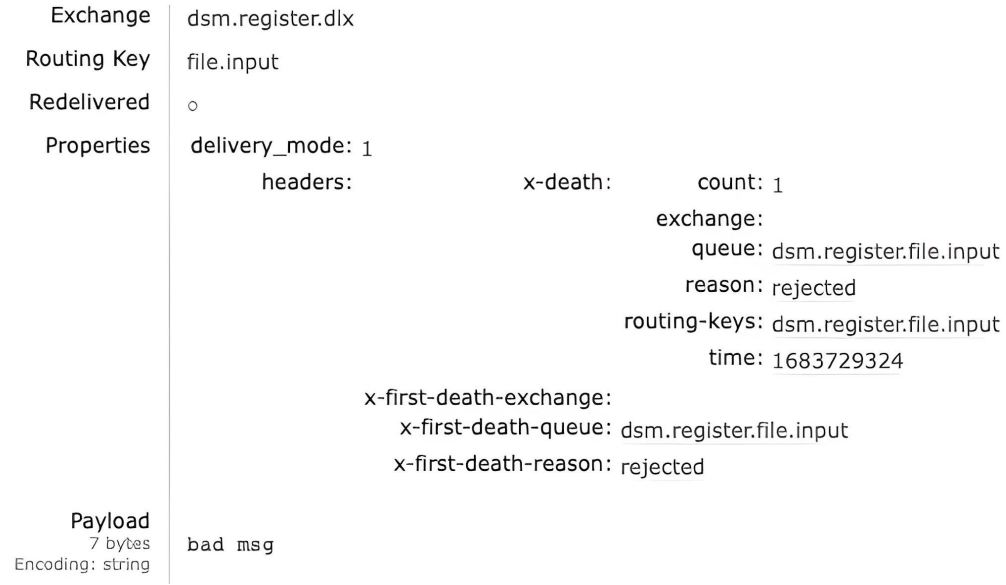
- При ошибке обработки сообщения оно отклоняется (**basic reject**)
- Для каждой очереди реализована доп. очередь “мёртвых сообщений” (**DLQ**/Dead letter queue)



Очереди “мёртвых сообщений”

В очередь *dsm.register.file.input* отправлено сообщение “bad msg”

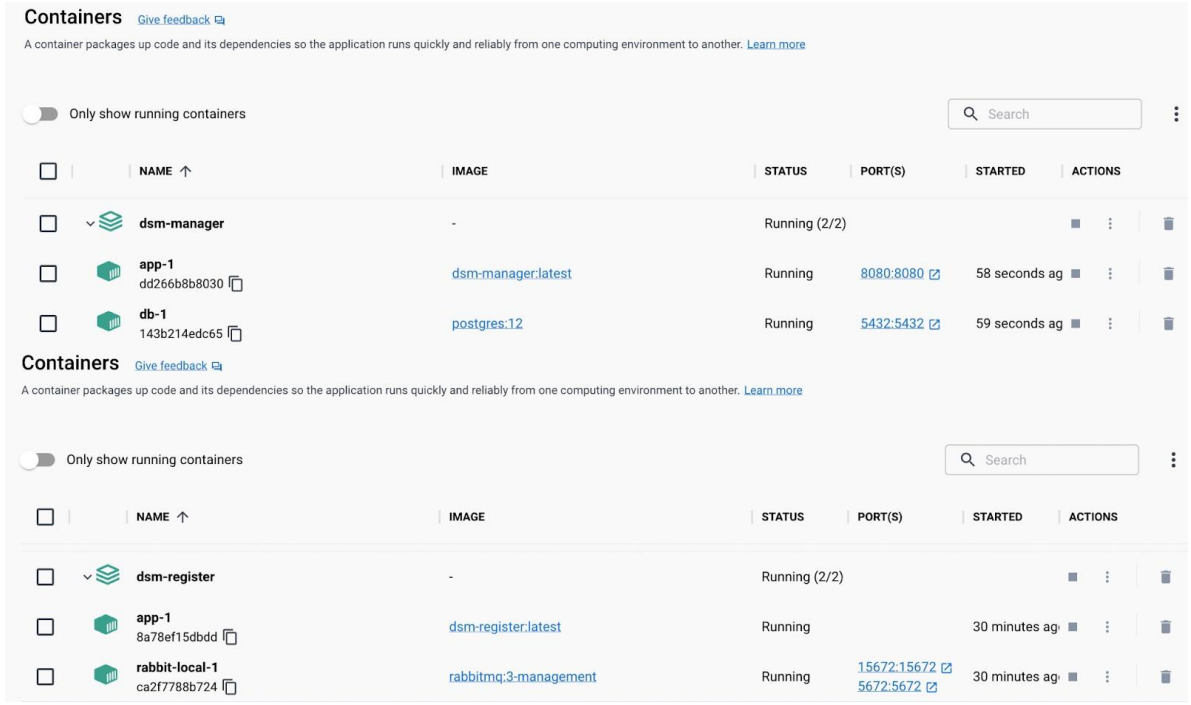
2023-05-10 17:35:24 ERROR: [DSM-REGISTER-E001] [CONSUMER-HANDLER] **Error on consume msg** b'bad msg': Expecting value: line 1 column 1 (char 0) [in /src/app/executer/rabbit/consumers/base/consumer_handler.py:30



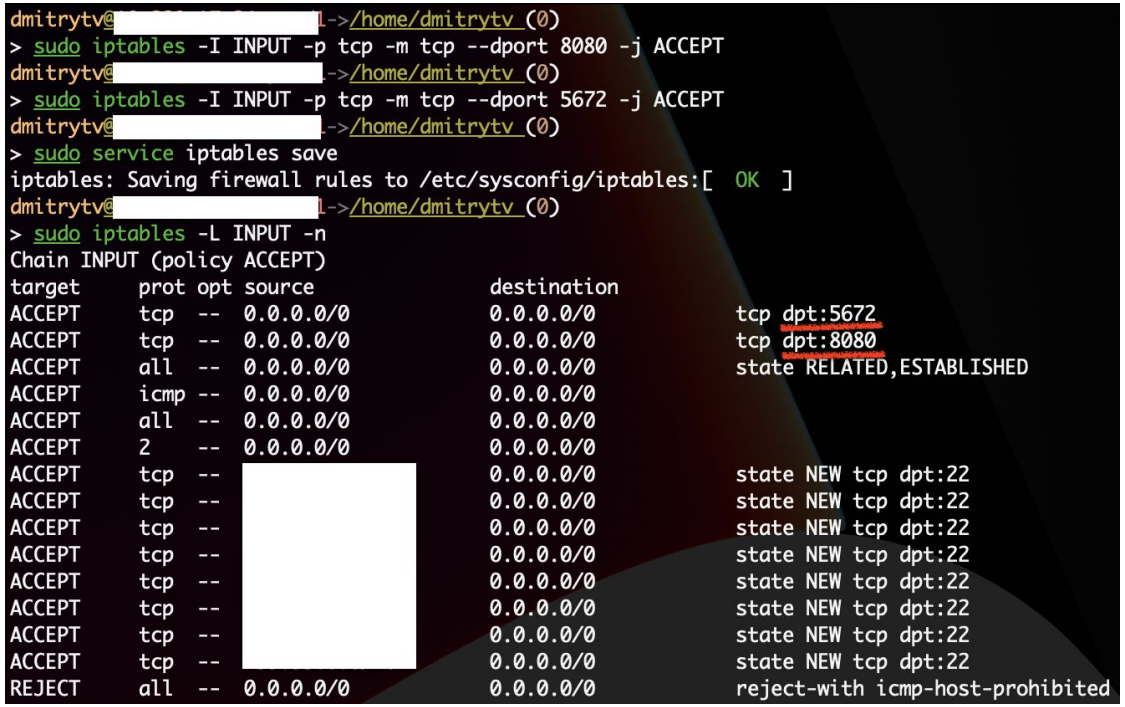
Демонстрация работы механизма DLQ

Доп. Прототипирование. Подготовка окружения для сборки и запуска

- Определены .env файлы переменных окружения для задания настроек
- Разработаны Docker образы сервисов на основе Dockerfile-ов. Составлен Docker Compose файл для всей инфраструктуры.
- Настроены сетевые доступы на стенде



Развёрнутые прототипы и инфраструктура в виде Docker контейнеров



Настройка Firewall на стенде МЛИТ ОИЯИ