MESHCHERYAKOV LABORATORY of INFORMATION TECHNOLOGIES

SPD NICA

SPD Online Filter
Middleware Development Status

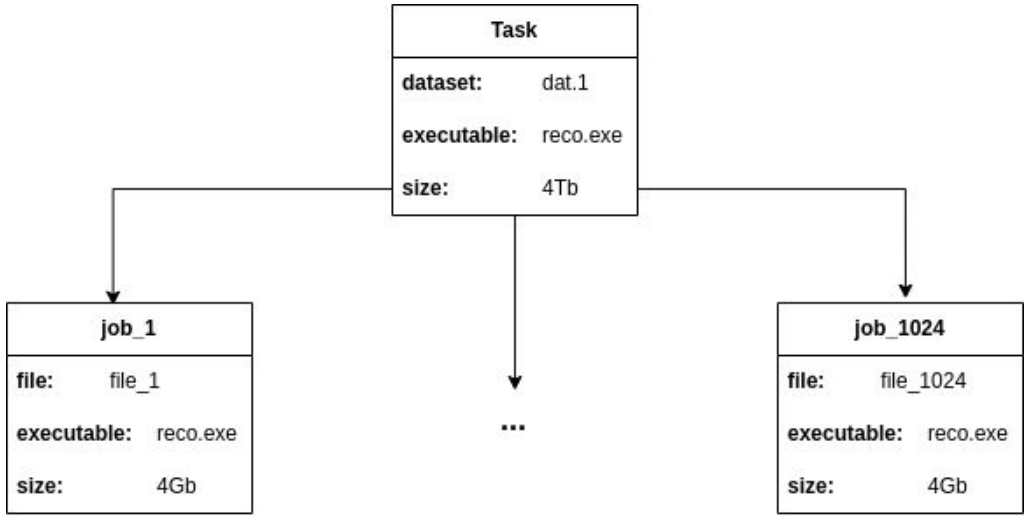
Nikita Greben[a]
Artem Plotnikov[b]
Polina Korshunova[b]

[a] Joint Institute for Nuclear Research, Dubna
[b] Moscow Engineering Physics Institute, Moscow

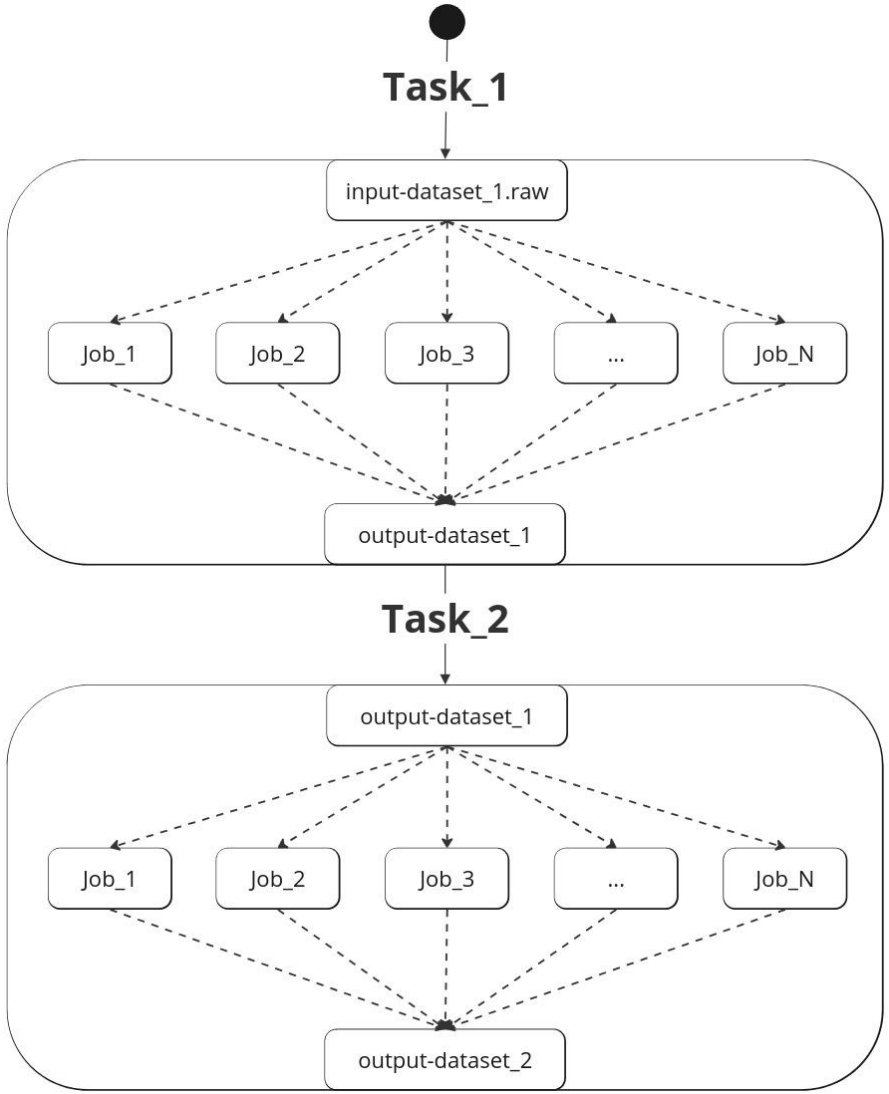
VIII SPD Collaboration Meeting. 7.11.2024

High-throughput computing

- **HTC** is defined as a type of computing that simultaneously executes numerous simple and computationally independent jobs to perform a data processing task.
- Since each data element can be processed simultaneously, this can be applied to data aggregated by a data acquisition system (DAQ).
- To ensure efficient utilization of computational resources, data processing should be multi-stage:
  - One stage of processing → **task**
  - Processing a block of data (file) → **job**



Task-job relationship

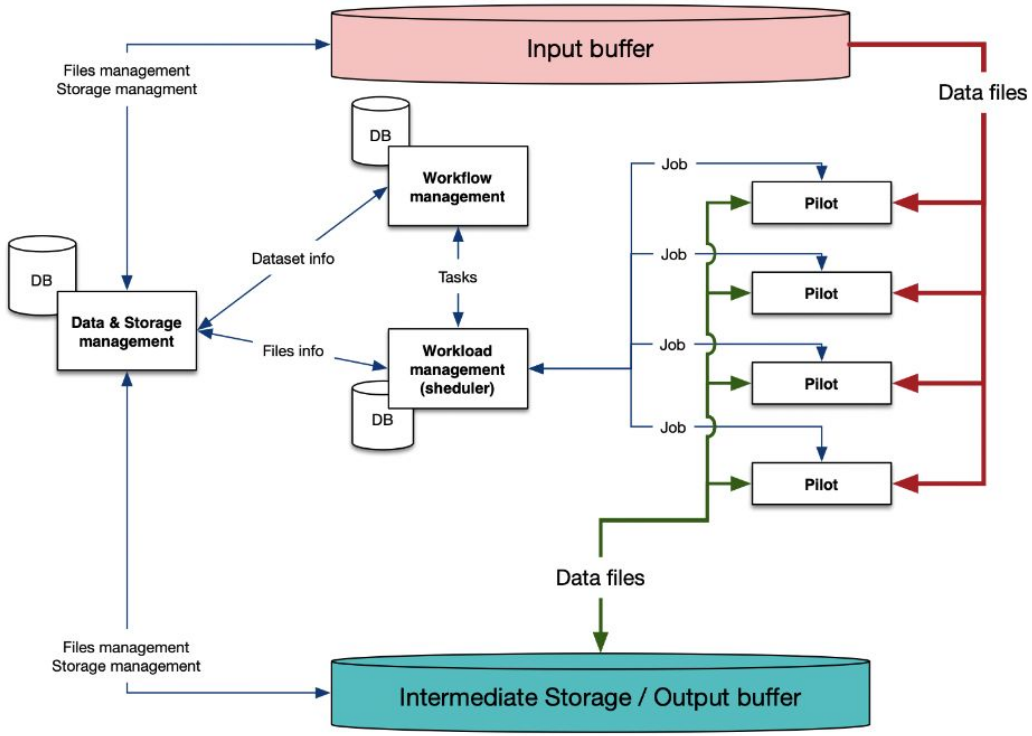


Data processing workflow example

Middleware software

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

- **Data management system (one master student)**
  - Data lifecycle support (data catalog, consistency check, cleanup, storage);
- **Workflow Management System (one master student)**
  - Define and execute processing chains by generating the required number of computational tasks;
- **Workload management system (2 PhD students, including me):**
  - Create the required number of processing jobs to perform the task;
  - Control job execution through pilots working on compute nodes;
- Coordinated by **Danila Oleynik**

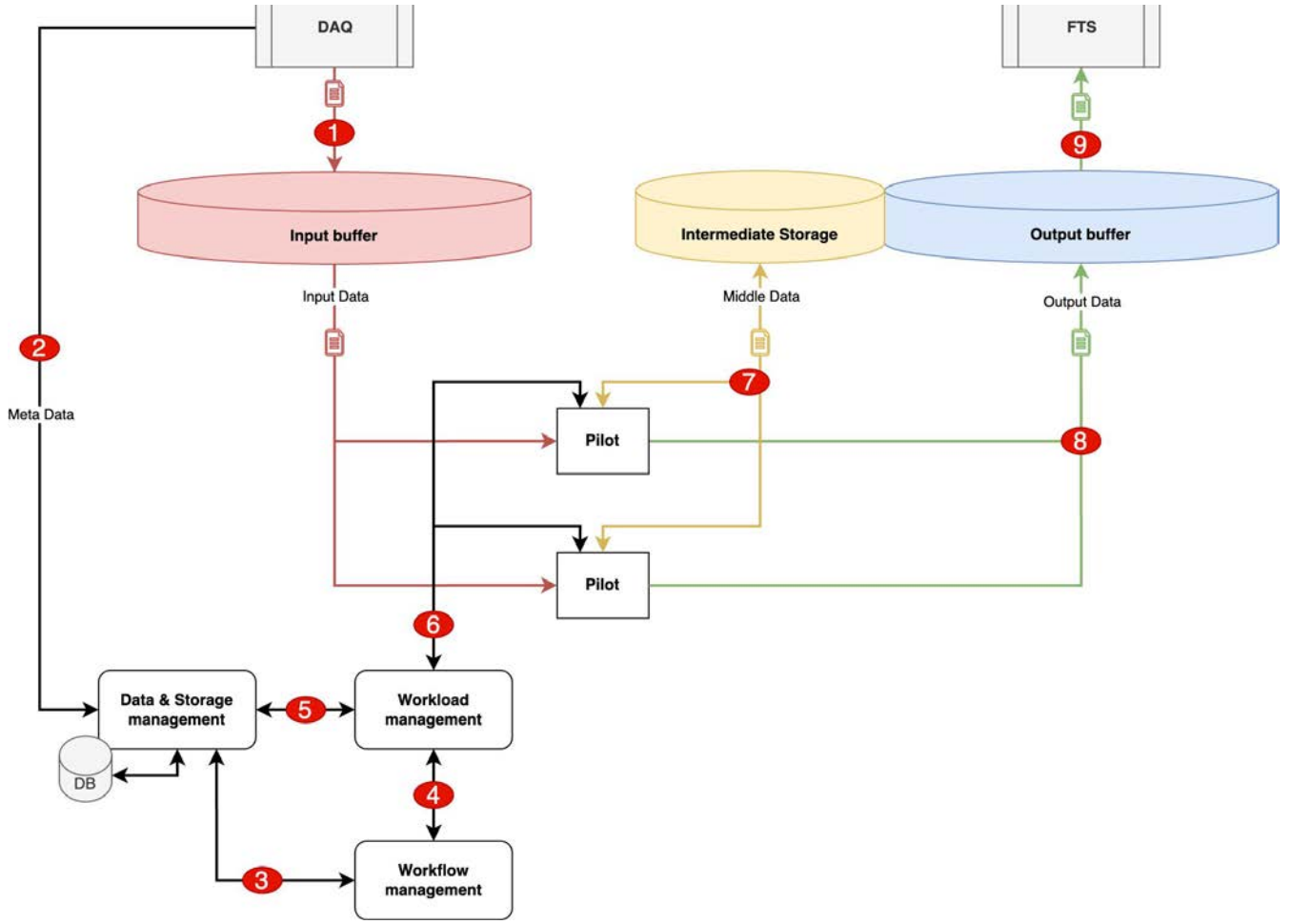


Architecture of SPD Online Filter

Dataflow and data processing concept

Main data streams:

- SPD DAQs, after dividing sensor signals into time blocks, send data to the SPD Online Filter input buffer as files of a consistent size.
- The workflow management system creates and deletes intermediate and final data sets
- The workload management system “populates” the data sets with information about the resulting files
- At each stage of data processing, pilots will read and write files to storage and create secondary data



Tereschenko, D., Ponomarev, E., Oleynik, D. *et al.* SPD On-Line Filter: Workflow and Data Management Systems. *Phys. Part. Nuclei* **55**, 603–605 (2024).

Workload management system requirements - reminder

The key requirement - systems must meet the **high-throughput paradigm**.

- **Task registration**: formalized task description, including job options and required metadata registration;
- **Jobs definition**: generation of required number of jobs to perform task by controlled loading of available computing resources;
- **Jobs execution management**: continuous job state monitoring by communication with pilot, job retries in case of failures, job execution termination;
- **Consistency control**: control of the consistency of information in relation to the tasks, files and jobs;
- **Scheduling:** implementing a scheduling principle for task/job distribution;

Forming jobs based on dataset contents, one file per one job

Data and Workflow Management system requirements

Data management

- ❏ Abstraction from the DAQ data format;
- ❏ The ability to logically group 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 a data processing perspective;
- ❏ Control the consistency of information in the catalog with respect to input and output storage.

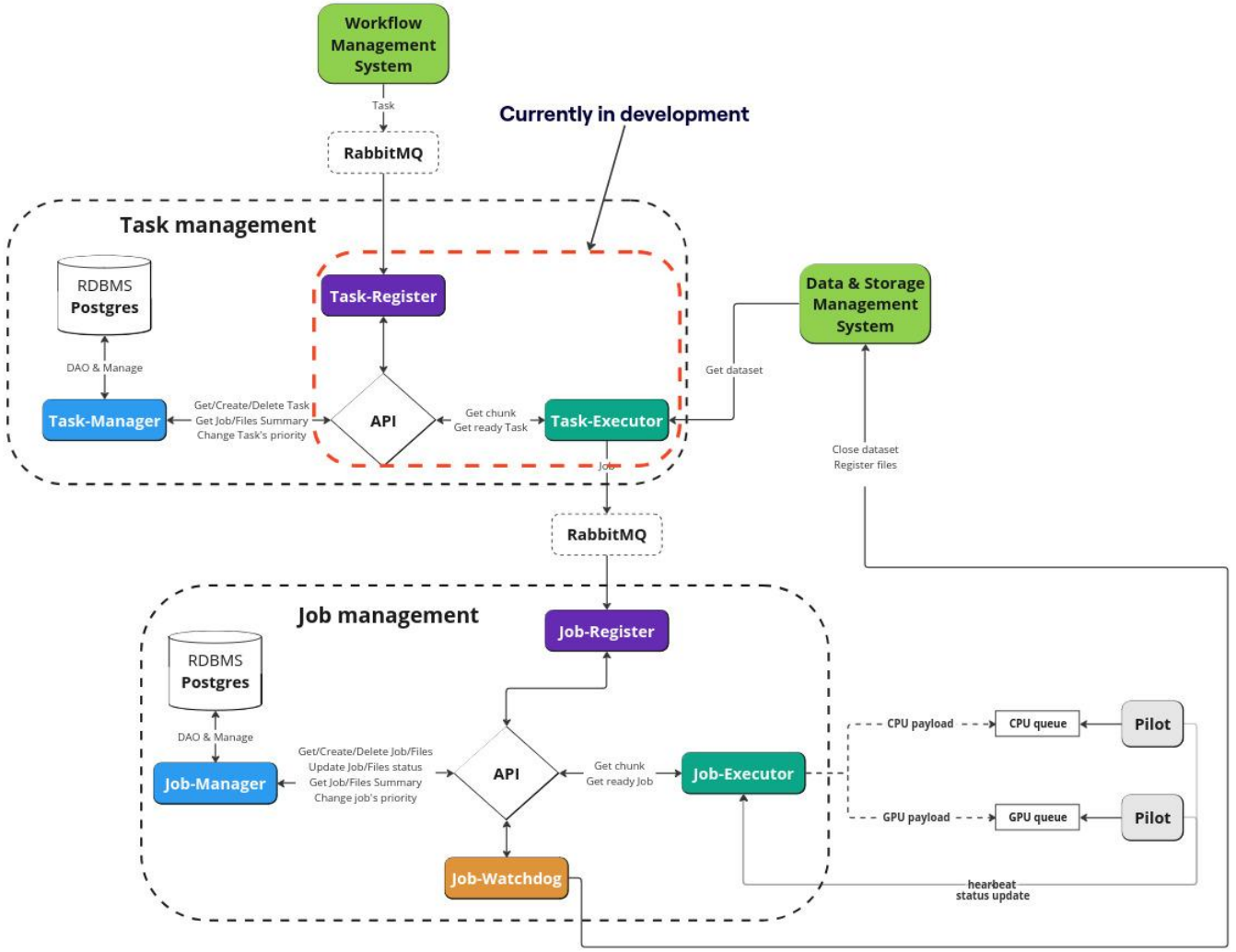
Workflow management

- ❏ Define workflows which represents multi-stage processing;
- ❏ Organizing data processing sequences (chains);
- ❏ Formation of a request for data processing according to a certain sequence;
- ❏ Processing request execution.

Architecture and functionality of Workload Management System

Reminder

- **task-manager** – implements both external and internal REST APIs. Responsible for registering tasks for processing, cancelling tasks, reporting on current output files and tasks in the system.
- **task-executor** – responsible for forming jobs in the system by dataset contents.
- **job-manager** – accountable for storing jobs and files metadata, as well as providing a REST API for the executed jobs.
- **job-executor** – responsible for distribution of jobs to pilot applications, updating the status of jobs
- **pilot** – responsible for running jobs on compute nodes, organizing their execution, and communicating various information about their progress and status.



Pilot Agent Reminder

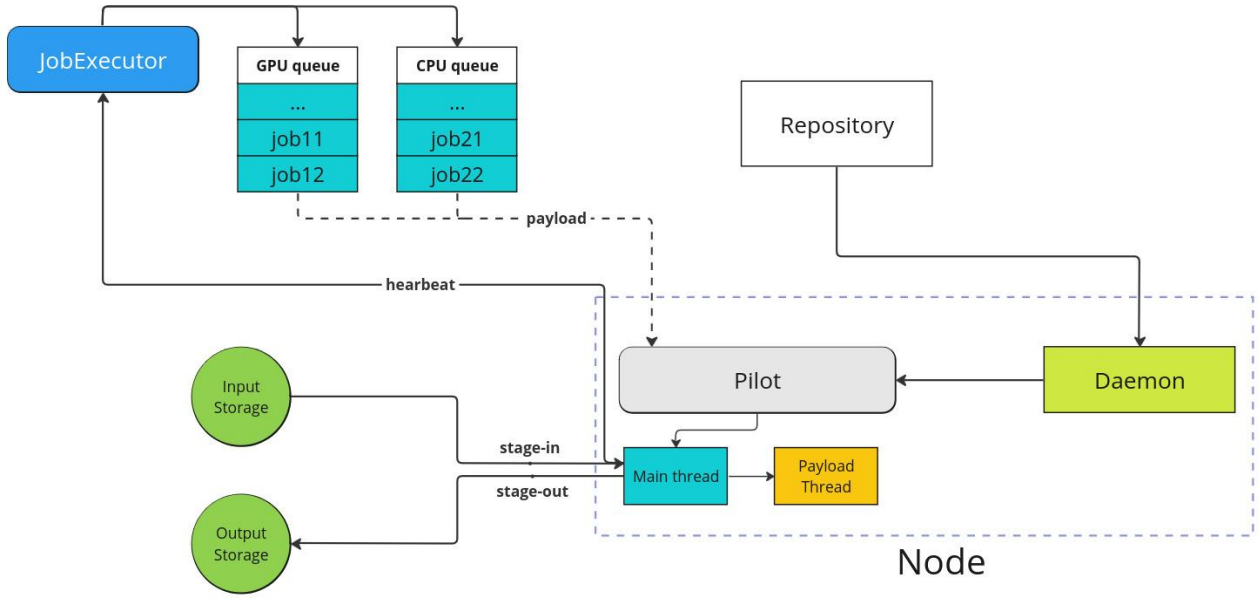
- The agent application is deployed on a compute node and consists of the following two components: a UNIX daemon and the pilot itself.
- The UNIX daemon's objective is to run the next pilot by downloading an up-to-date version from the repository.
- Pilot itself is a multi-threaded Python application responsible for
  - Receiving and validating jobs from the message broker;
  - Downloading input files for the payload stage and uploading the result files to the output storage;
  - Launching a subprocess to execute a payload (decoding DAQ format, track recognition algorithm, etc.)
  - Keeping the upstream system informed of the current status of the payload and the pilot itself via heartbeat/status updates during each phase of pilot execution;

Two types of nodes:

- Multi-CPU
- Multi-CPU + GPU

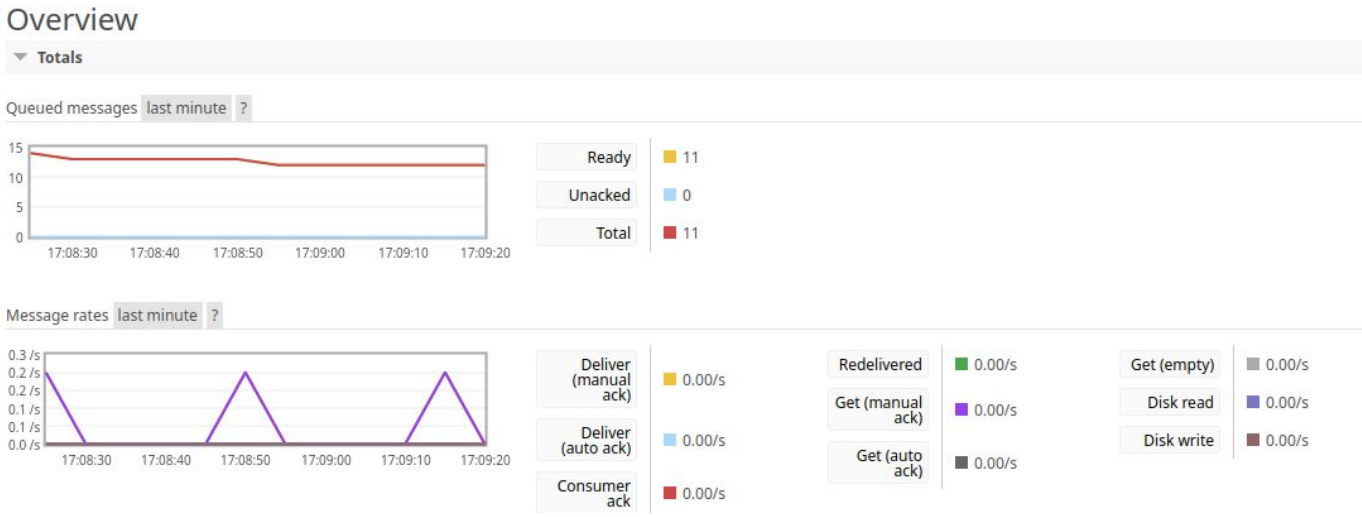
Two communication channels:

- HTTP (aiohttp)
- AMQP (message broker - RabbitMQ)

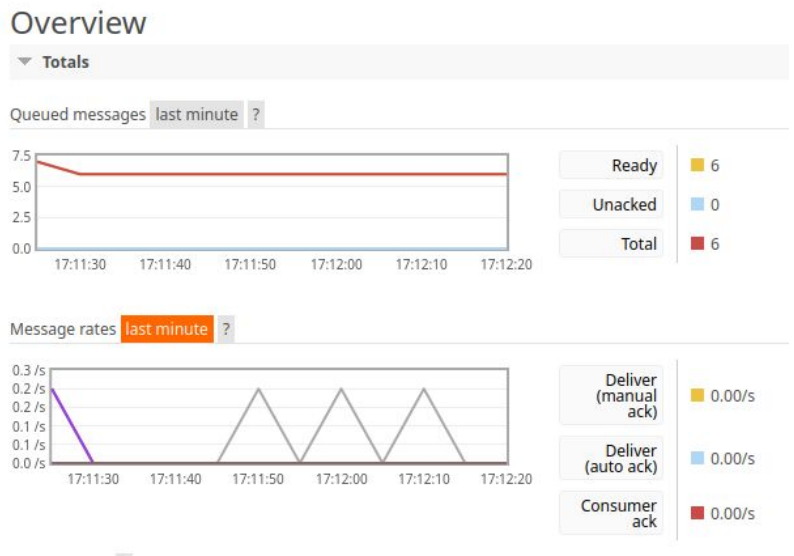


Workload Management System - Pilot

- ✓ A detailed job status model has been described;
- ✓ Error codes introduced;
- ✓ Pilot ran through all stages of the current job execution;
- ✓ Pilot at this stage runs a script that does a basic MD5 hash compute;
- ✓ **UNIX Daemon** is implemented and currently running;
- ✓ Two pilots are currently running on two different virtual machines;
- ✓ **No more pilot emulator!**
- ✖ Major cycle of tests and refactoring is required;
- ✖ Debugging during execution of the entire task (all jobs associated with a task);



UNIX Daemon's running Pilot



Aftermath of executing the entire queue

Workload Management System Current Status

Design of services:

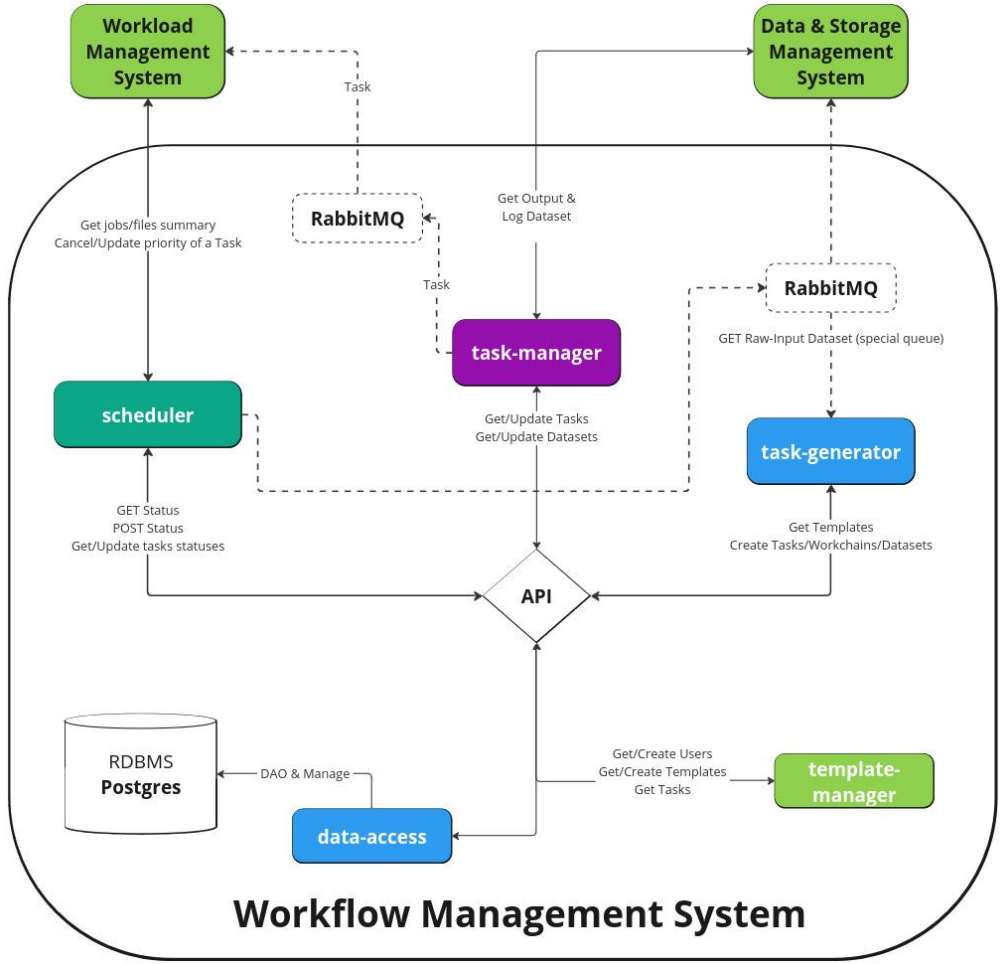
- ✓ Designed and implemented a list of required REST API methods and their signatures;
- ✓ Implemented a mechanism for declaring the data model in the database based on ORM and migration scripts;
- ✓ Configured CD tools (build and deployment) on the JINR LIT infrastructure;
- ✓ Designed inter-service interaction scenarios – defined API contracts;
- ✓ Designed Pilot internal architecture;
- ✓ Workload Management System - Pilot Interaction Models in Finite State Machine;
- ✖ A layer to handle reply-messages (designated queue) after registration from DSM should be added;

Prototype of services:

- ✓ Job management subsystem is the most advanced: most interactions implemented and being tested;
- ✓ Pilot and Pilot Daemon is currently working;
- ✓ Pilot handles all stages of job execution on the given workload;
- ✖ Task processing;

Workflow Management System

- **task-manager** – a service that requests the last dataset created in the previous step of the workflow chain, populates it, and sends the next task to the WMS.
- **task-generator** – responsible for starting the workflows based on the available templates**.**
- **template-manager** – service for interaction with the data processing operator/user**.**
- **data access** – a service that encapsulates direct database access, provides a RESTful API's through endpoints**.**
- **scheduler** – a services responsible for making decision on when to close datasets, cancel or change a priority of a task.

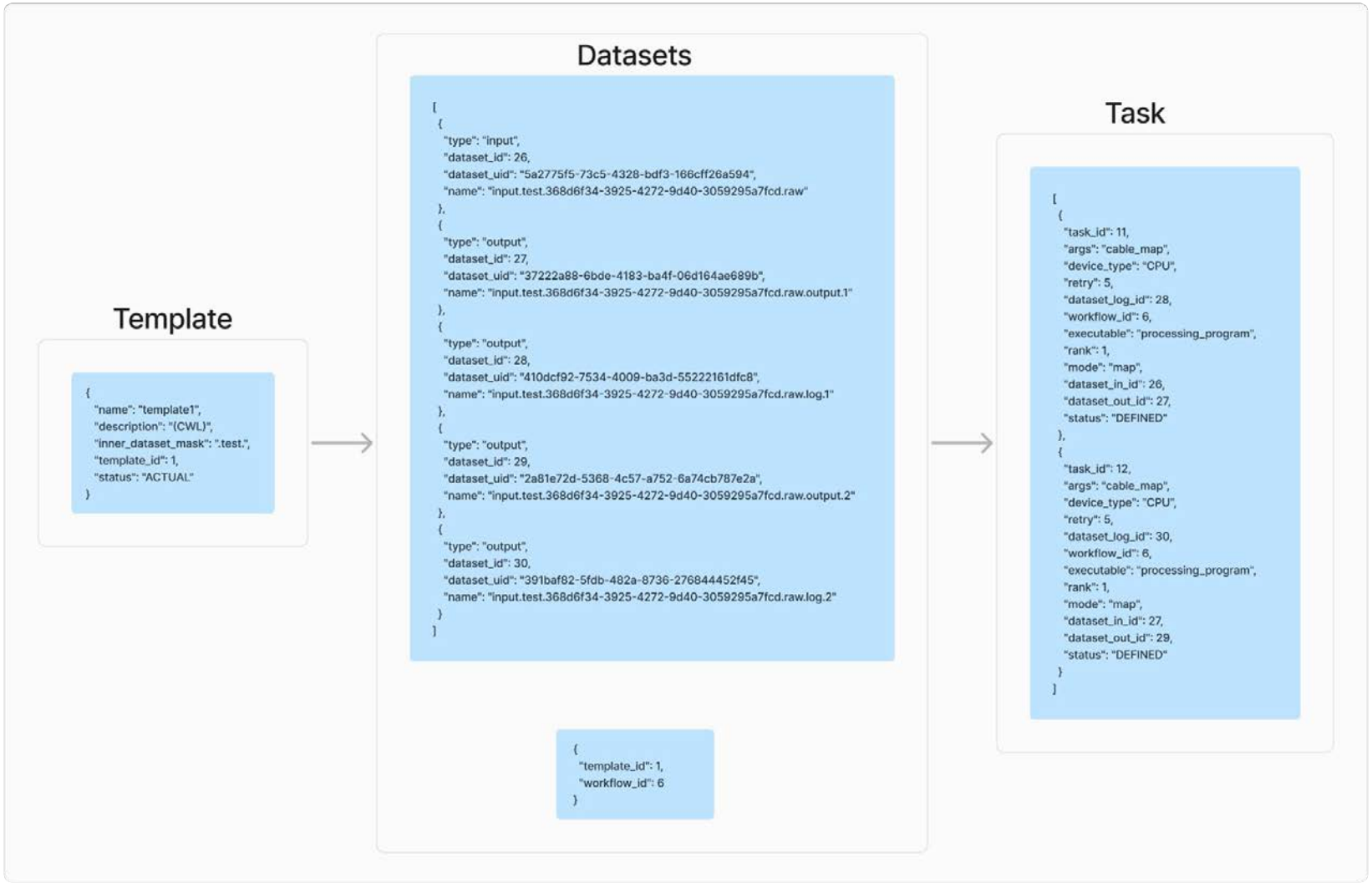


Workflow Management System High-Level Architecture

Task generation service

1. Getting registered datasets from **DMS** from RabbitMQ;
2. Matching datasets by name mask to the desired template;
3. Registration of input dataset in the system;
4. Creating a workflow from a template;
5. Creating output dataset and log dataset in the system;
6. Creating a task;

From template to tasks



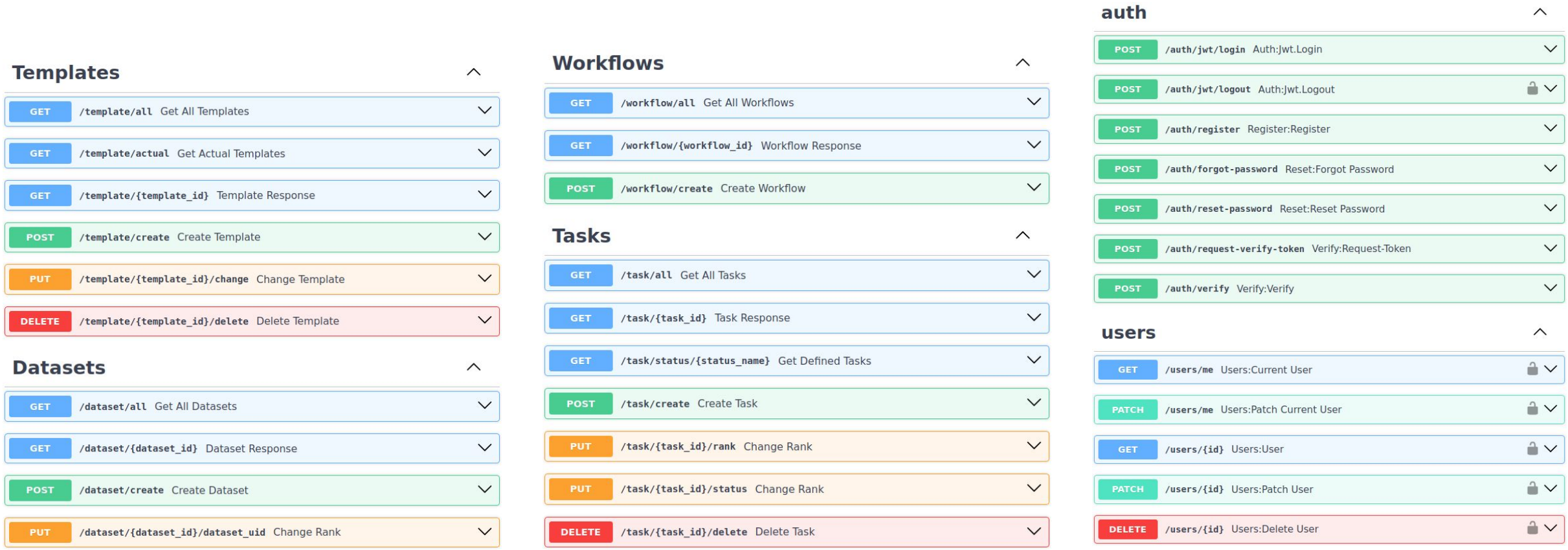
Task management service

- Iterate on tasks in “DEFINED” status;
- Querring DMS about the status of the input dataset (“CLOSED”);
- Creating output datasets and log datasets in DMS;
- Sending the task to RabbitMQ for further processing in WMS;
- Change task status to "IN_PROGRESS".

Sending tasks for processing

Data access service

- Data access service is implemented and provides all necessary endpoints at this stage;
- Test coverage is required;

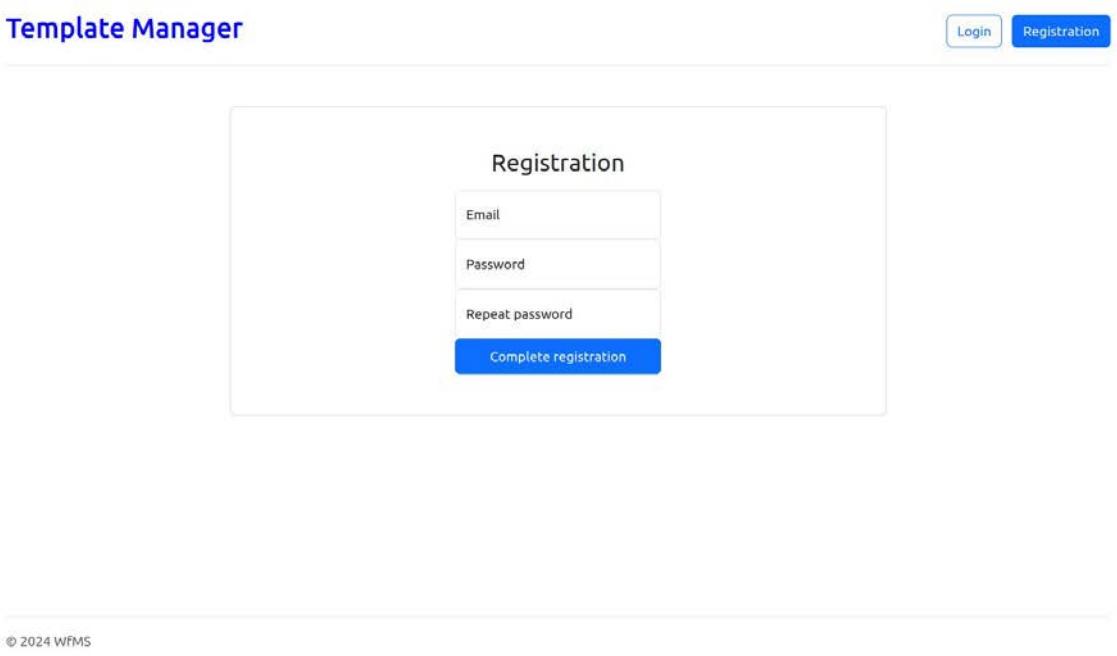


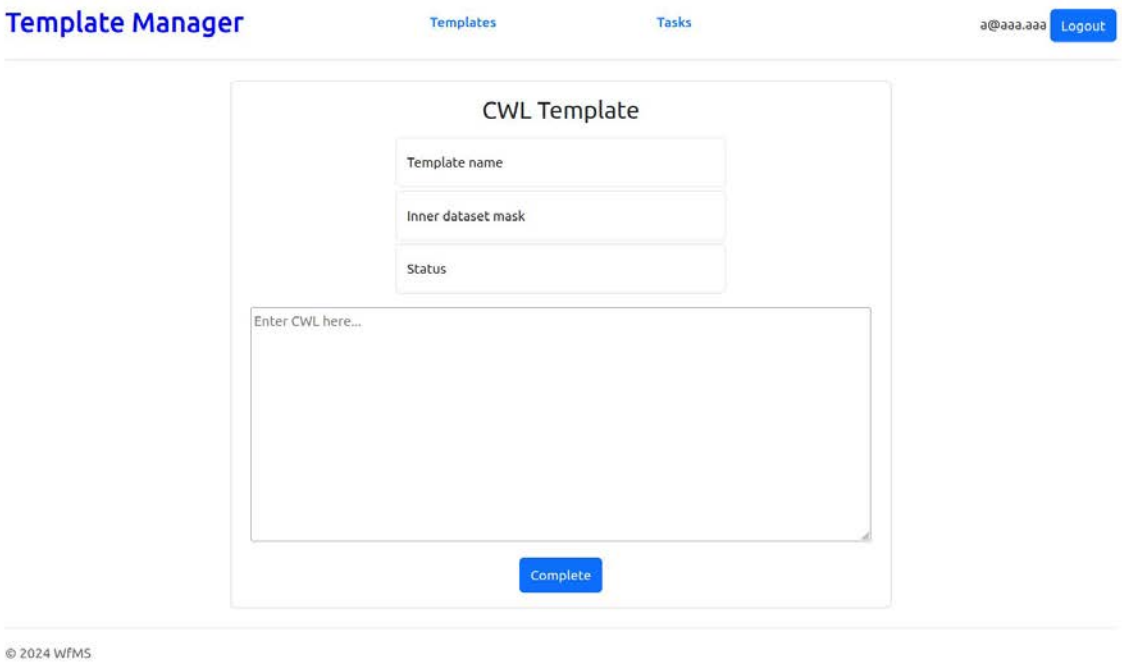
Description of the current implemented API's

Service for interaction with user

- Registration and authorization of users with different rights;
- CWL template/tasks output;
- Creation of CWL templates by superuser;
- CWL template status changes by superuser;
- Store template in the database;

- FastAPI Users
- JWT-token





Examples of Templates and Tasks

- Viewing templates and tasks is available to all users who have completed the authorization process;
- Template creation is only available to superusers;

Template Manager Templates Tasks a@aaa.aaa Logout

Create template

template_id	name	inner_dataset_mask	description	status
1	template1	.test.	{"steps": {"decoding": {"run": {"class": "CommandLineTool", "baseCommand": "echo", "inputs": {"dataset_name": {"type": "string"}, "processing_program": {"type": "string"}, "processing_program_version": {"type": "string"}, "cable_map": {"type": "File"}, "input_params": {"type": "File"}}, "outputs": {"output_dataset": {"type": "File"}, "log_dataset": {"type": "File"}}}, "in": {"dataset_name": ".test.", "processing_program": "processing_program", "processing_program_version": "processing_program_version", "cable_map": "cable_map", "input_params": "input_params"}, "out": "[output_dataset, log_dataset]"}, "reconstruction": {"run": {"class": "CommandLineTool", "baseCommand": "echo", "inputs": {"dataset_name": {"type": "string"}, "processing_program": {"type": "string"}, "processing_program_version": {"type": "string"}, "cable_map": {"type": "File"}, "input_params": {"type": "File"}}, "outputs": {"output_dataset": {"type": "File"}, "log_dataset": {"type": "File"}}}, "in": {"dataset_name": ".test.", "processing_program": "processing_program", "processing_program_version": "processing_program_version", "cable_map": "cable_map", "input_params": "input_params"}, "out": "[output_dataset, log_dataset]"}}}	ACTUAL
2	template2	.test.	{"steps": {"decoding": {"run": {"class": "CommandLineTool", "baseCommand": "echo", "inputs": {"dataset_name": {"type": "string"}, "processing_program": {"type": "string"}, "processing_program_version": {"type": "string"}, "cable_map": {"type": "File"}, "input_params": {"type": "File"}}, "outputs": {"output_dataset": {"type": "File"}, "log_dataset": {"type": "File"}}}, "in": {"dataset_name": ".test.", "processing_program": "processing_program", "processing_program_version": "processing_program_version", "cable_map": "cable_map", "input_params": "input_params"}, "out": "[output_dataset, log_dataset]"}}}	ARCHIVED

Created template

Template Manager Templates Tasks a@aaa.aaa Logout

task_id	wflow_id	exec	args	rank	device	mode	retry	datas_in_id	datas_out_id	datas_log_id	status
11	6	processing_program	cable_map	1	CPU	map	5	26	27	28	IN_PROGRESS
12	6	processing_program	cable_map	1	CPU	map	5	27	29	30	IN_PROGRESS
13	7	processing_program	cable_map	1	CPU	map	5	31	32	33	IN_PROGRESS
14	7	processing_program	cable_map	1	CPU	map	5	32	34	35	IN_PROGRESS
15	8	processing_program	cable_map	1	CPU	map	5	36	37	38	IN_PROGRESS
16	8	processing_program	cable_map	1	CPU	map	5	37	39	40	IN_PROGRESS
17	9	processing_program	cable_map	1	CPU	map	5	41	42	43	IN_PROGRESS
18	9	processing_program	cable_map	1	CPU	map	5	42	44	45	IN_PROGRESS
19	10	processing_program	cable_map	1	CPU	map	5	46	47	48	IN_PROGRESS
20	10	processing_program	cable_map	1	CPU	map	5	47	49	50	IN_PROGRESS
21	11	processing_program	cable_map	1	CPU	map	5	51	52	53	IN_PROGRESS
22	11	processing_program	cable_map	1	CPU	map	5	52	54	55	IN_PROGRESS
23	12	processing_program	cable_map	1	CPU	map	5	56	57	58	IN_PROGRESS
24	12	processing_program	cable_map	1	CPU	map	5	57	59	60	IN_PROGRESS

WfMS task description

Workflow Management System Current Status

Current results:

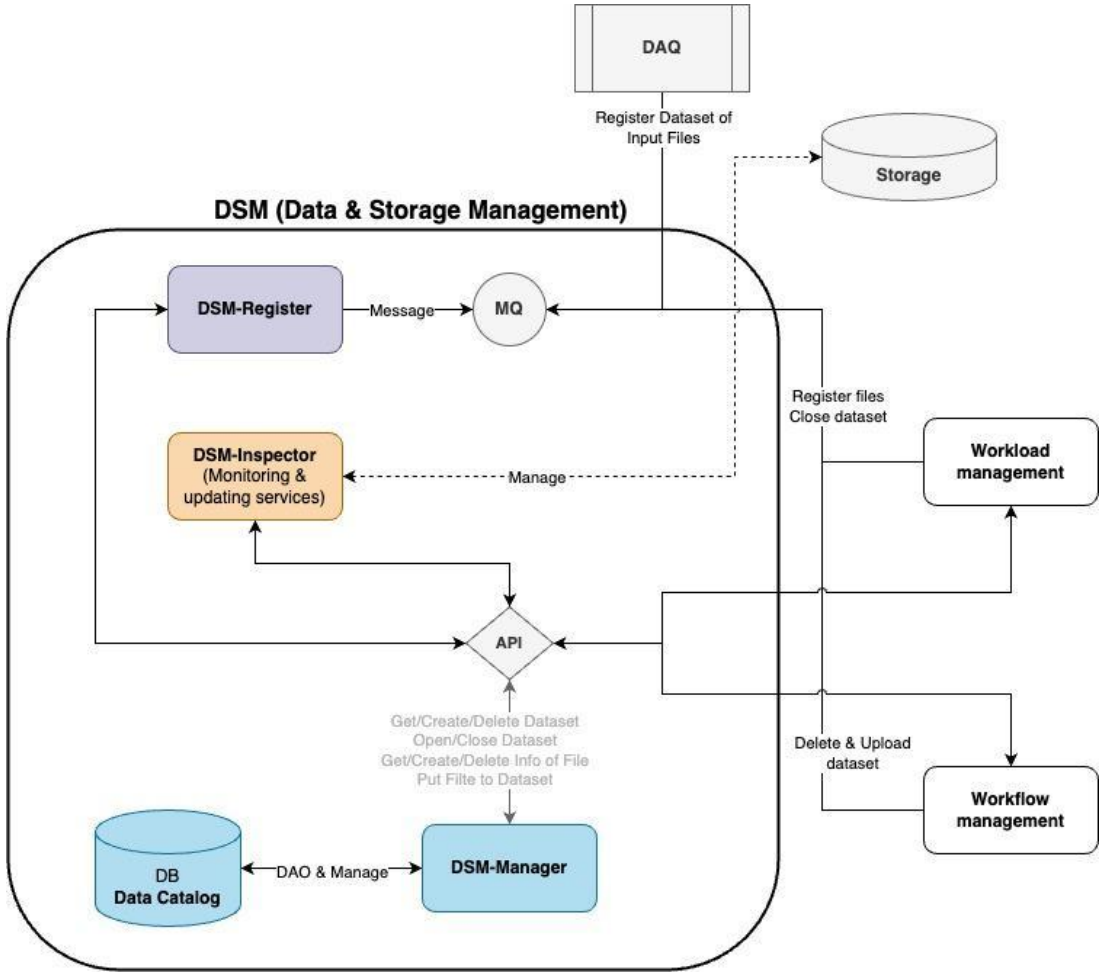
- ✓ Designed a list of required REST API's and implemented data access service;
- ✓ Implemented a service for user interaction, allowing for templates and tasks management;
- ✓ Implemented a task generation service: maps a dataset by mask to the desired template, creates a workchain, and generates tasks;
- ✓ Multi-container application orchestrated via docker-compose;

Further plans:

- ➢ To go fully asynchronous;
- ➢ Integrate with SPD IAM;
- ➢ Add support for loading a template from a file;
- ➢ Run integration tests;
- ➢ Implement a service for interaction with **WMS;**

Architecture and functionality of Data Management System

- **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 information about the system's data structure
  - 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 System

Tereschenko, D., Ponomarev, E., Oleynik, D. *et al.* SPD On-Line Filter: Workflow and Data Management Systems. *Phys. Part. Nuclei* **55**, 603–605 (2024).

RabbitMQ configured queues

Exchange: dsm.register

Overview

Bindings

This exchange

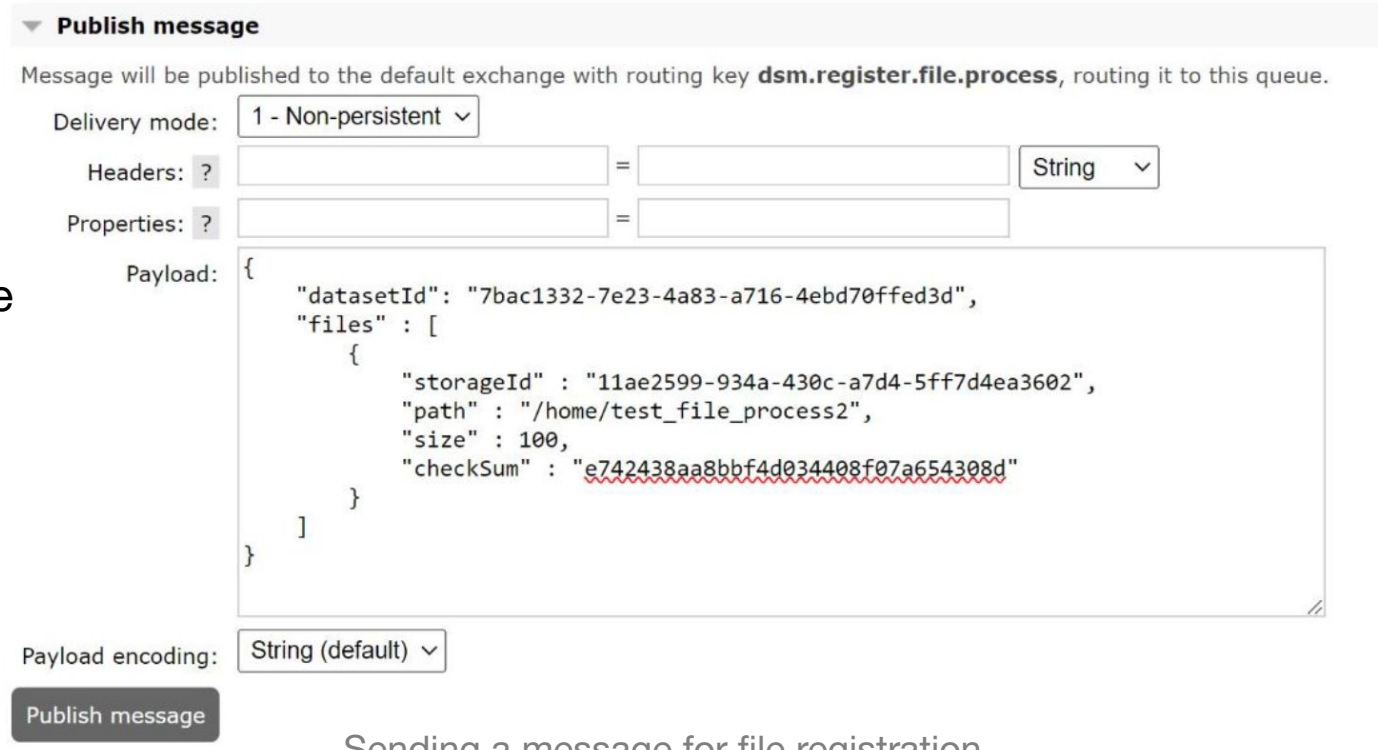
⇓

To	Routing key	Arguments	
dsm.register.dataset.close	dataset.close		Unbind
dsm.register.dataset.delete	dataset.delete		Unbind
dsm.register.dataset.input	dataset.input		Unbind
dsm.register.dataset.upload	dataset.upload		Unbind
dsm.register.file.input	file.input		Unbind
dsm.register.file.process	file.process		Unbind
dsm.register.file.process.reply	file.process.reply		Unbind

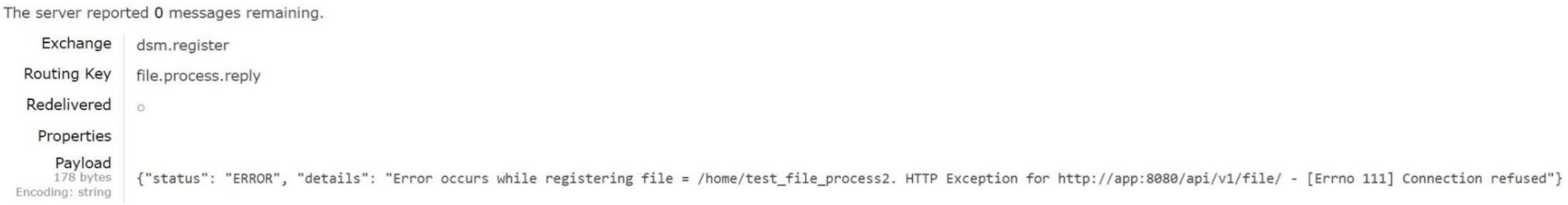
Exchange	Routing Key	Appointment
dsm.register (direct)	file.input	Receiving information about incoming files to the input buffer
	file.process	Receiving information about new files, received during processing
	dataset.close	Accepting a request to close a dataset
	dataset.upload	Accepting an application to upload files in a dataset to an external storage
	dataset.delete	Accepting a request to delete files in a dataset on the internal storage

Example: occurrence of an error during file registration

- Let's temporarily suspend the **dsm-manager** service and send a message to the **dsm.register.file.process** queue.
- An error should occur when connecting to the service - corresponding error message should be sent to the **dsm.register.file.process.reply** queue



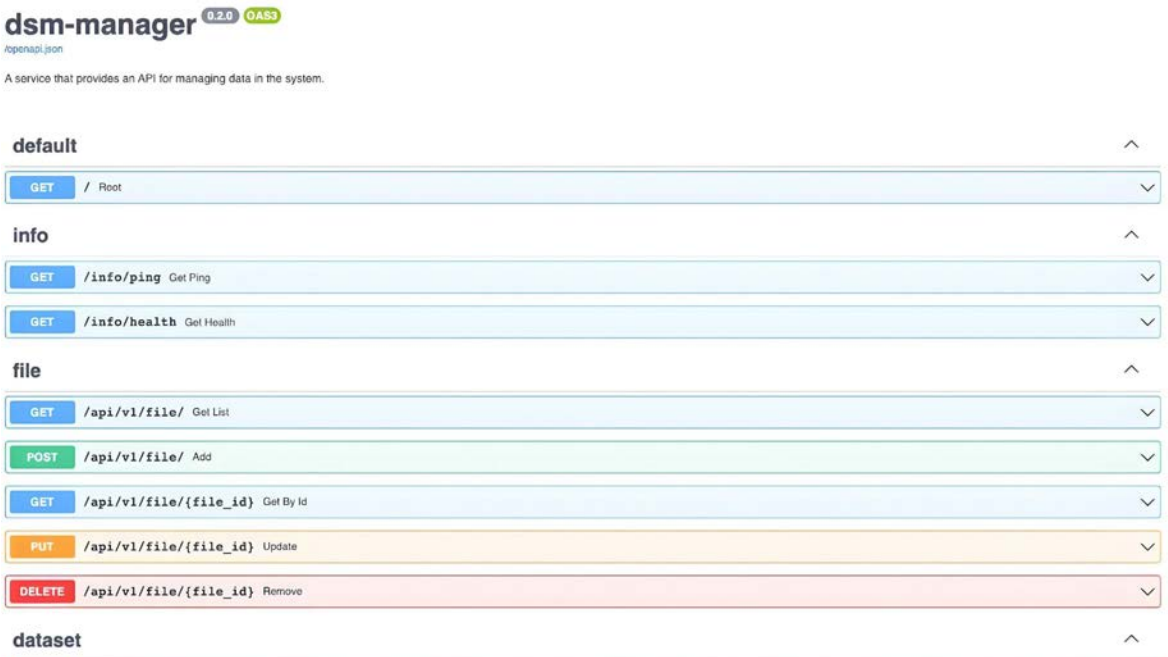
Sending a message for file registration



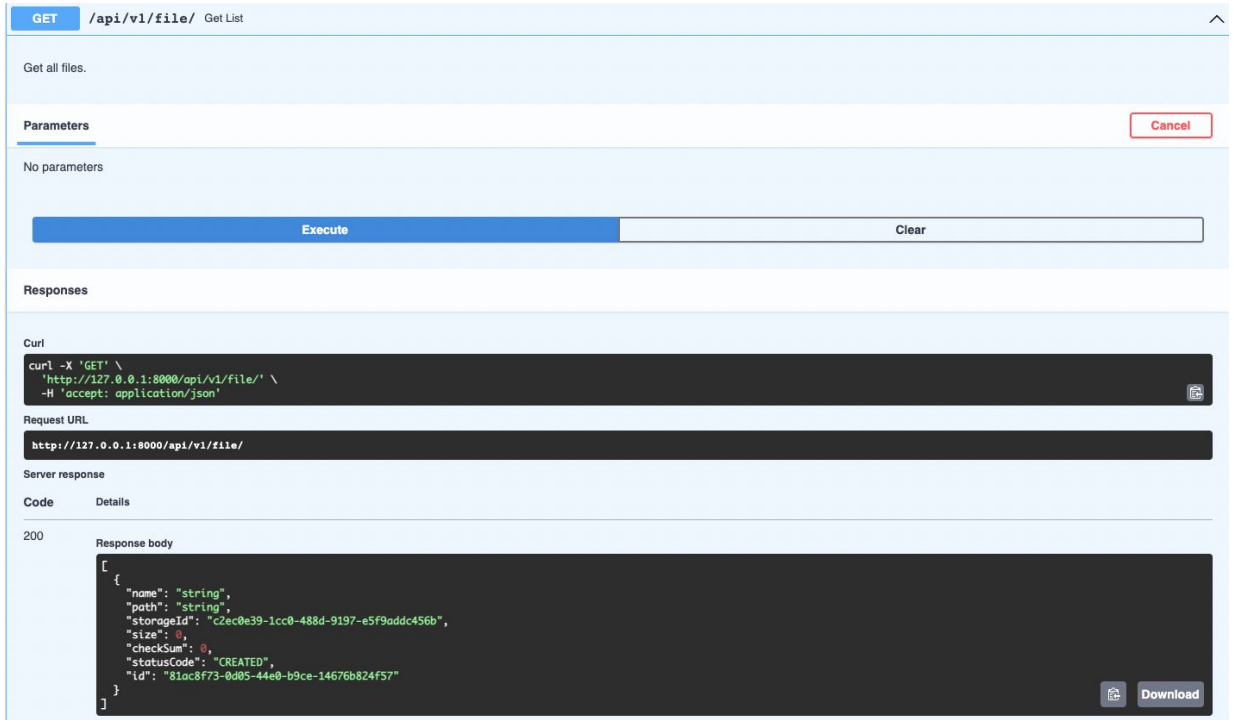
Response in reply-queue

DSM-Manager

- A basic set of CRUD operations on data in the form of REST API is developed.
- The whole application construction is based on one of SOLID principles - DIP (dependency inversion principle) and is implemented using **Dependency Injector** tool.



Swagger UI with API description of the service



Example of calling the service to get the list of files

Data Management System Status

Current results:

- ✓ **dsm-manager** is fully functional for this stage of prototyping;
- ✓ **dsm-register** is mostly implemented;

Further plans:

dsm-inspector

Implement background services for

- ➢ Deleting files on storages;
- ➢ Control file uploads;
- ➢ Control storage utilization;

dsm-register

Realize processing of messages from queues

- ➢ dsm.register.dataset.closed;
- ➢ dsm.register.dataset.upload;
- ➢ dsm.register.dataset.delete;

Next major steps

- **Task and workflow processing**
  - Execute the entire workflow set up on the level of **Workflow Management System**;
  - The entire workflow - a chain of dependent tasks.
- **Middleware and applied software integration**
  - Requires prototyped applied software and simulated data;
  - Non-functional requirements for applied software.
- **Logging**
  - Currently, each microservice logs are mapped to the host via a shared file system between Docker and the host.
  - Ideally – **ELK** (*Elastic-Logstash-Kibana*) stack to build a log analysis platform.
- **Configuration**
  - Consider to centralize some of the shared configurations across multiple services (*Consul, Etcd*).
- **Documentation**
  - Given the increasing complexity of the internal logic of the software, it is necessary to document each step of the development.
- **Metrics and monitoring**
  - For example, service query-per-second, API responsiveness, service latency etc. (*InfluxDB, Prometheus, Graphana*)

Thank you for your attention!