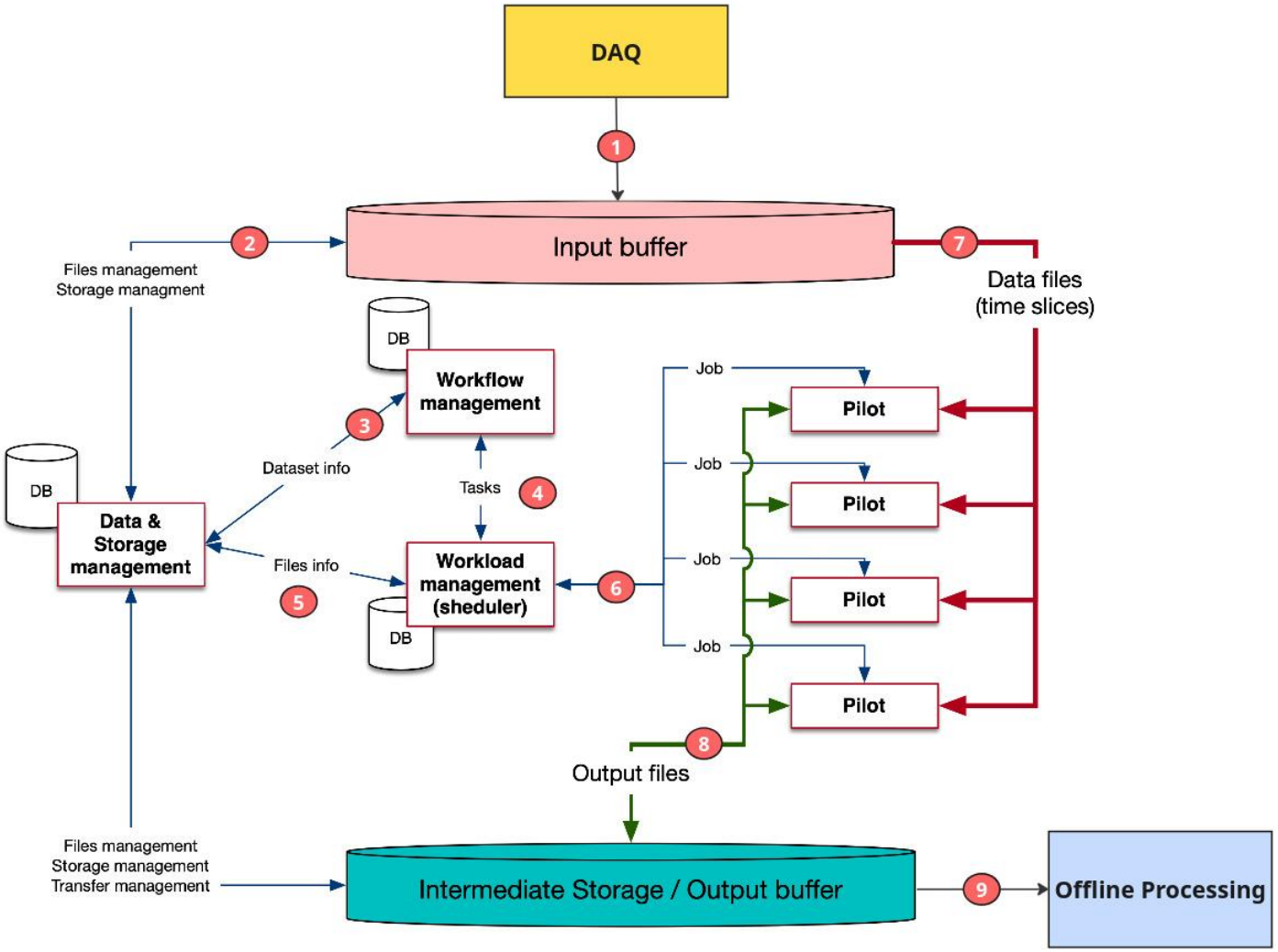
SPD Online Filter
Middleware Status Update

Nikita Greben
Joint Institute for Nuclear Research, MLIT, Dubna

23.10.2025

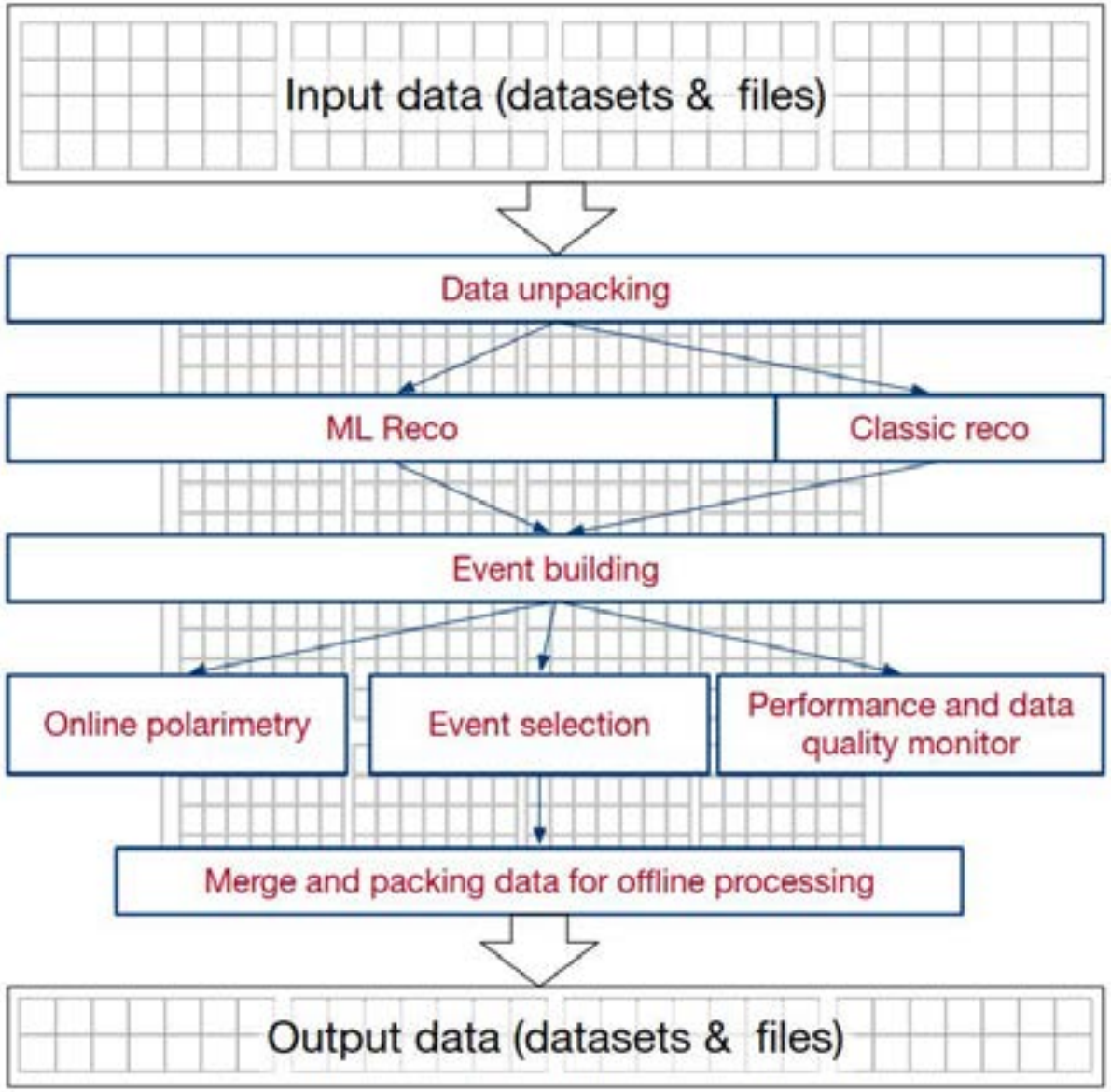
Reminder: main components

- **Data & Storage Management (Polina Korshunova)**
  - Data lifecycle support (data catalog, consistency check, cleanup, storage);
- **Workflow Management System (Leonid Romanyched, Nikita Greben)**
  - Define and execute processing chains by generating the required number of computational tasks;
- **Workload management system (Nikita Greben, Leonid Romanychev):**
  - Create the required number of processing jobs to perform the task;
  - Control job execution through pilots working on compute nodes;

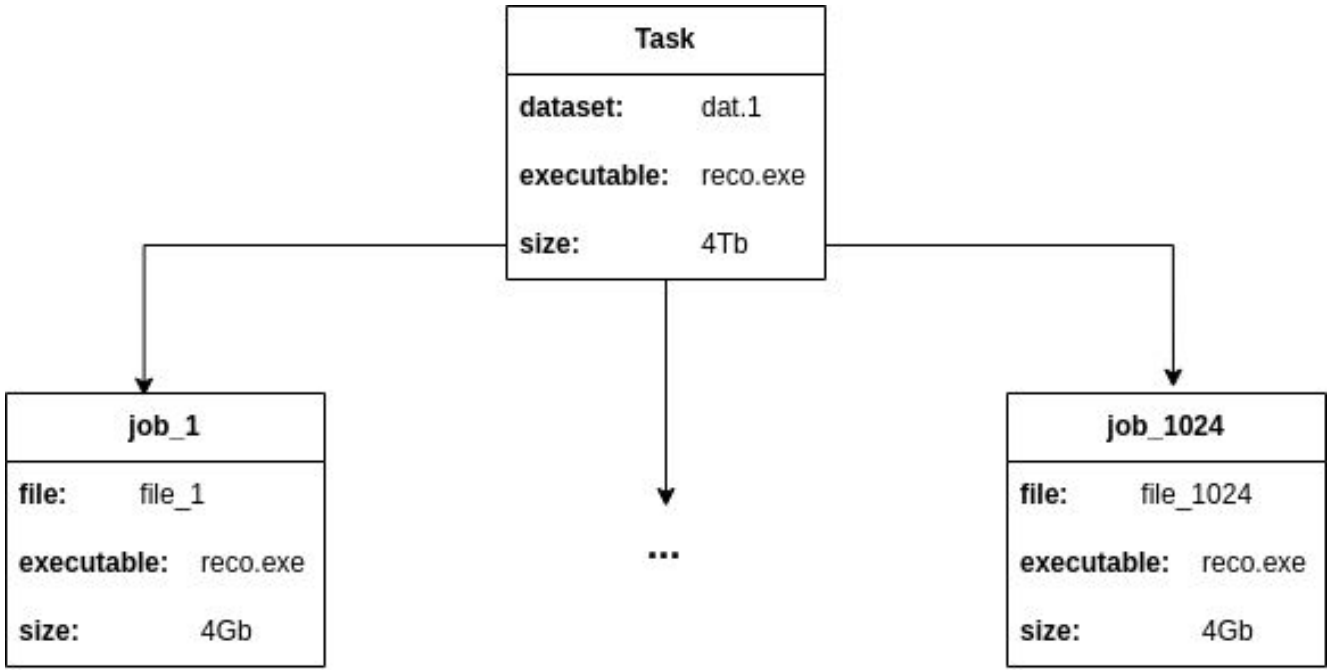


Reminder: Workflow Example

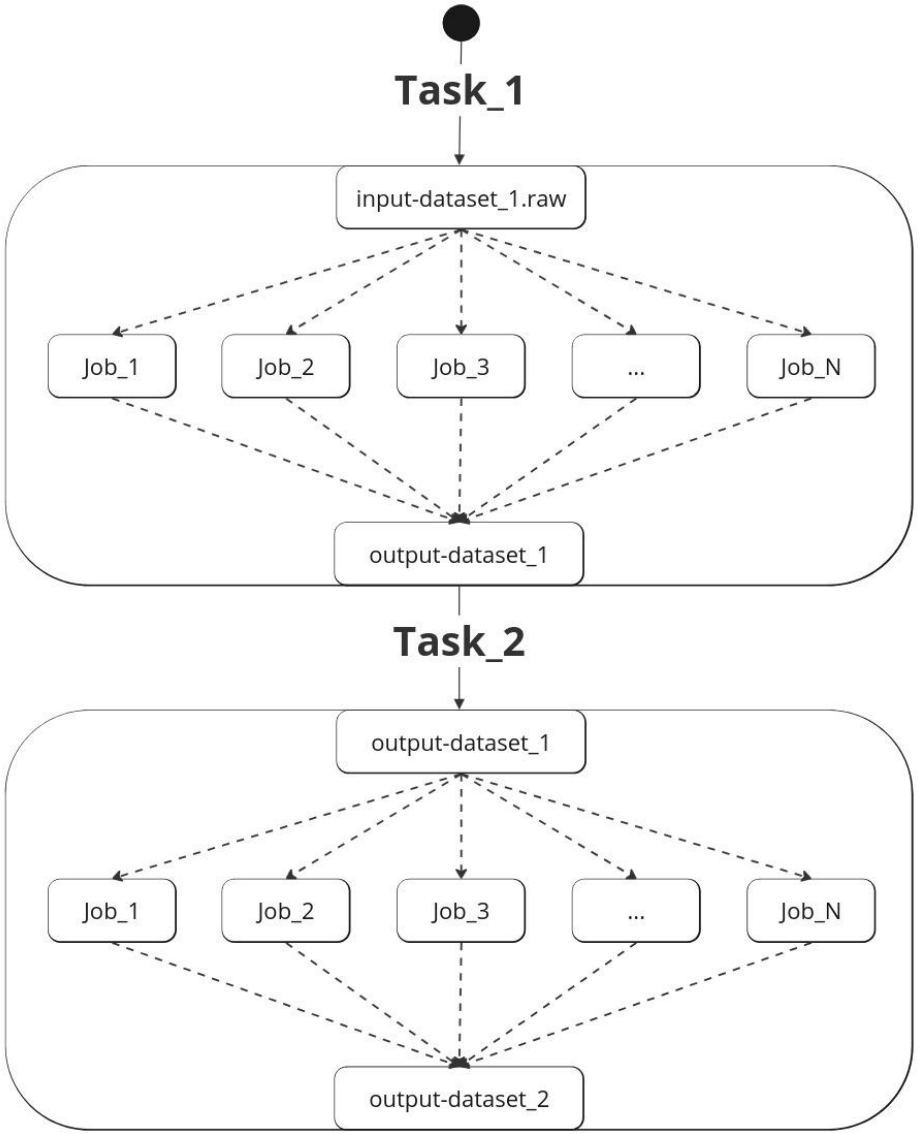
- The workflow is triggered by the presence of input datasets with matching masks.
- The output dataset of one stage is used as the input for the next stage(s).
- The final datasets are sent to the output buffer and are ready for unloading to the "Offline Processing" stage.
- **Operator** is responsible for designing and defining the workflow.



Task-job relationship (reminder)

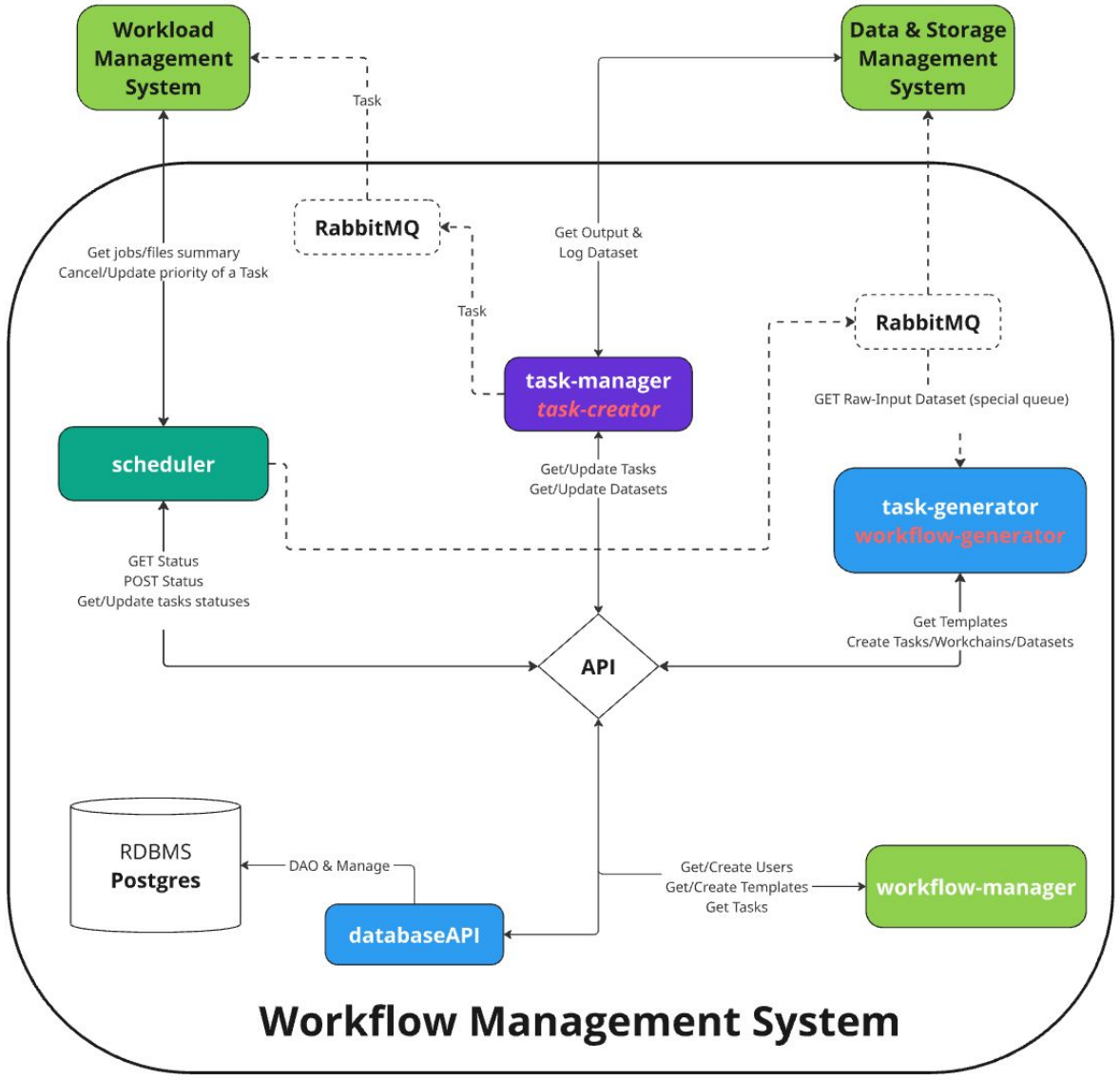


Task-job relationship



Data processing workflow example

Workflow Management System: Update



- ✅ Scheduler microservice is finished.
- ✅ The entire workflow across the three systems (**WfMS**, **WMS**, **DMS**) was executed on synthetic payload.
- ✅ CI/CD introduced.
- 😞 The person responsible is gone.

Template Definition

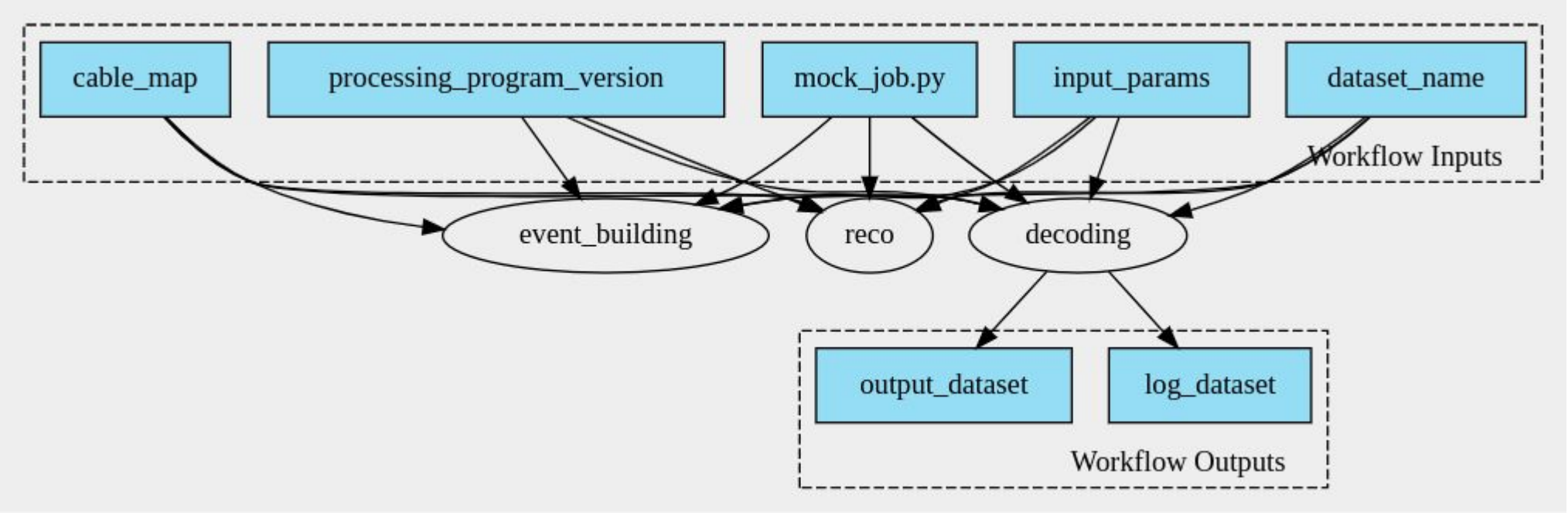
CWL description

Template name: MWGA-v5

```
class: Workflow
cwlVersion: v1.2
inputs:
  cable_map: File
  dataset_name: string
  input_params: File
  mock_job.py: string
  processing_program: string
  processing_program_version: string
label: Decoding of data
outputs:
  log_dataset:
    outputSource: decoding/log_dataset
    type: File
  output_dataset:
    outputSource: decoding/output_dataset
    type: File
steps:
  decoding:
    in:
      cable_map: cable_map
      dataset_name: dataset_name
      input_params: input_params
      processing_program: mock_job.py
      processing_program_version: processing_program_version
    out:
    - output_dataset
    - log_dataset
    run:
      baseCommand: echo
      class: CommandLineTool
```

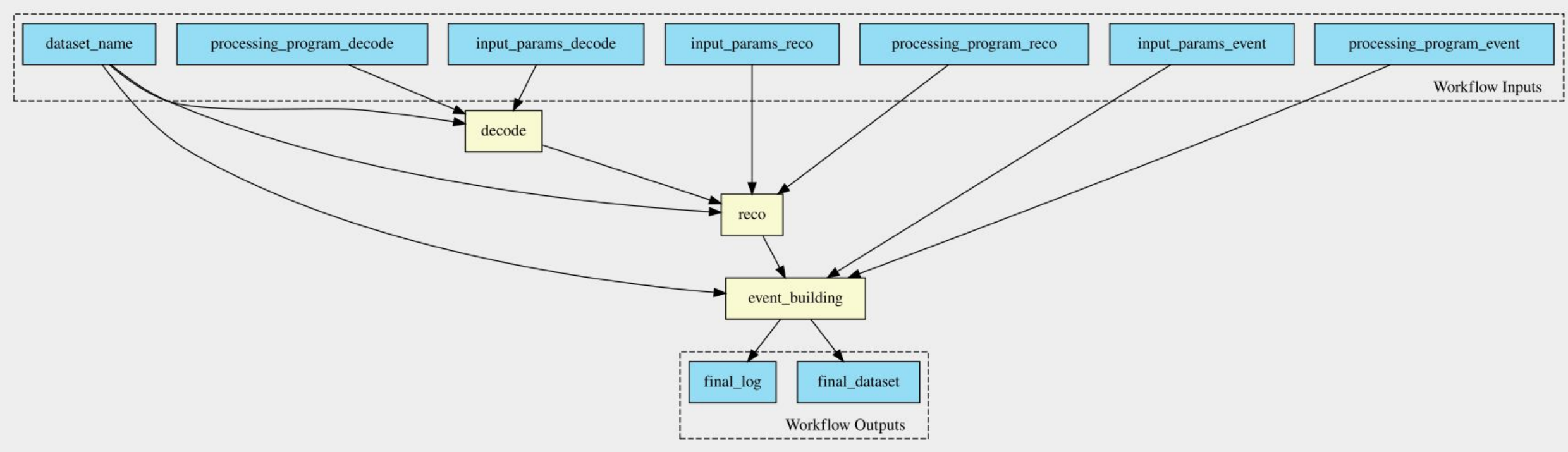
Workflow Management System serves as a sort of high level interpreter, executing the DAG

Wrong CWL (Common Workflow Language) definition

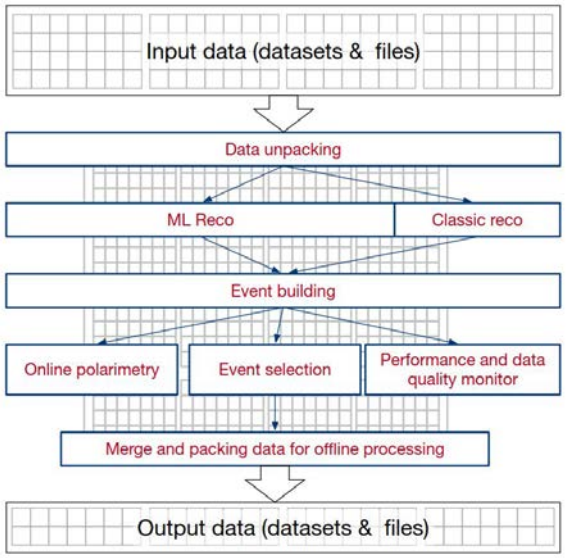


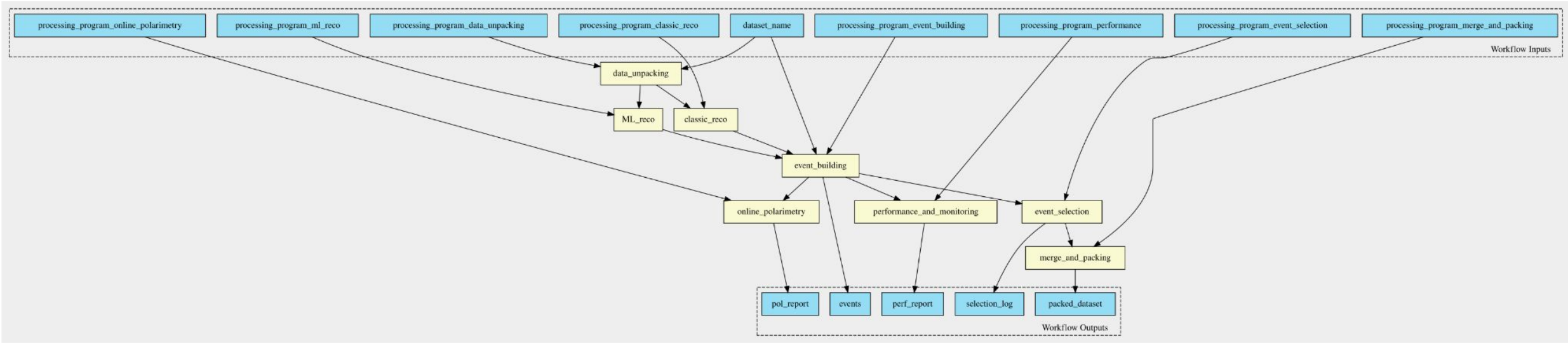
Right CWL (Common Workflow Language) definition

What was meant?

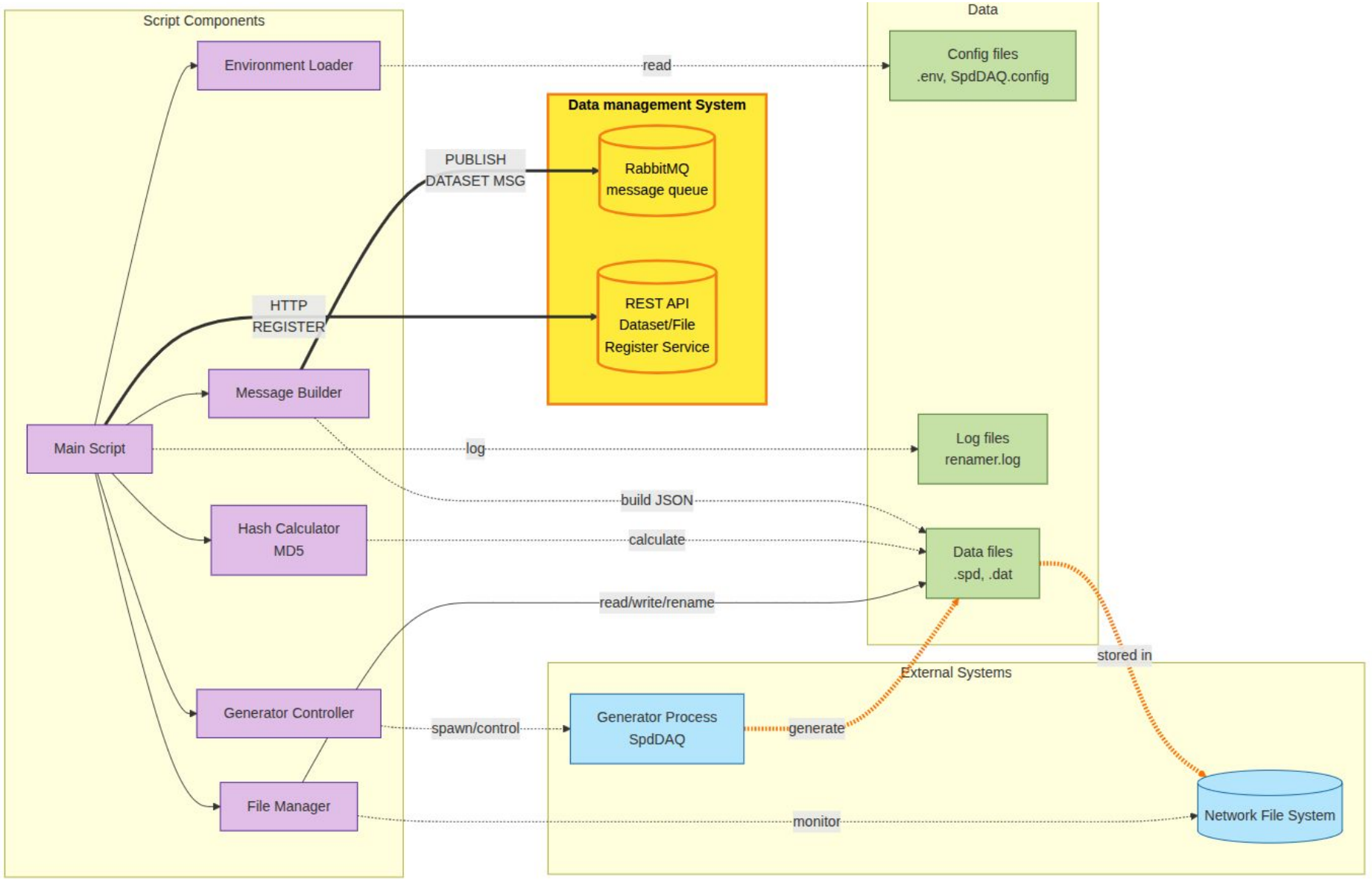


Operator's job: design and define the templates





File Management and Registration System - “DAQ Emulator”



- Only one workflow can be executed at a time.
- Needs a continuous "data stream" to test several workflows that are running simultaneously.

Running Workflow: example

Workflow Manager Templates Tasks ngreben Logout

id	wflow_id	step	template	exec	args	priority	type	mode	retries	in_ds_name	out_ds_name	log_ds_name	status
82	35	reco	MWGA-v5	mock_job.py	cable_map	1	CPU	Map	2	raw_run_07102025_035420.output.2	raw_run_07102025_035420.output.3	raw_run_07102025_035420.log.3	DEFINED
81	35	event_building	MWGA-v5	mock_job.py	cable_map	1	CPU	Map	2	raw_run_07102025_035420.output.1	raw_run_07102025_035420.output.2	raw_run_07102025_035420.log.2	RUNNING
80	35	decoding	MWGA-v5	mock_job.py	cable_map	1	CPU	Map	2	raw_run_07102025_035420	raw_run_07102025_035420.output.1	raw_run_07102025_035420.log.1	FINISHED

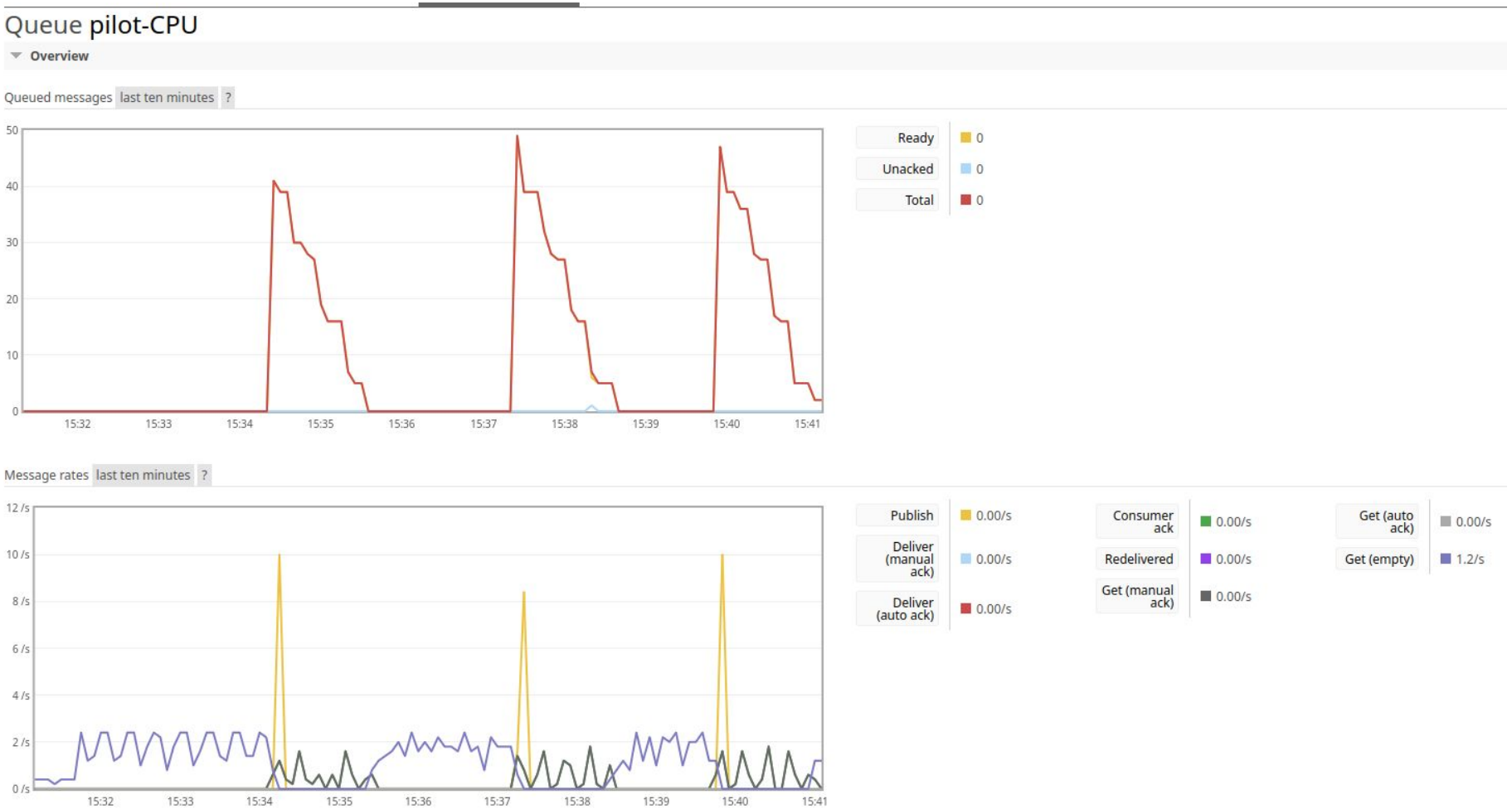
One task consist of 50 jobs.

id	wflow_id	step	template	exec	args	priority	type	mode	retries	in_ds_name	out_ds_name	log_ds_name	status
82	35	reco	MWGA-v5	mock_job.py	cable_map	1	CPU	Map	2	raw_run_07102025_035420.output.2	raw_run_07102025_035420.output.3	raw_run_07102025_035420.log.3	FINISHED
81	35	event_building	MWGA-v5	mock_job.py	cable_map	1	CPU	Map	2	raw_run_07102025_035420.output.1	raw_run_07102025_035420.output.2	raw_run_07102025_035420.log.2	FINISHED
80	35	decoding	MWGA-v5	mock_job.py	cable_map	1	CPU	Map	2	raw_run_07102025_035420	raw_run_07102025_035420.output.1	raw_run_07102025_035420.log.1	FINISHED

Finished workflow.

Workflow processing test

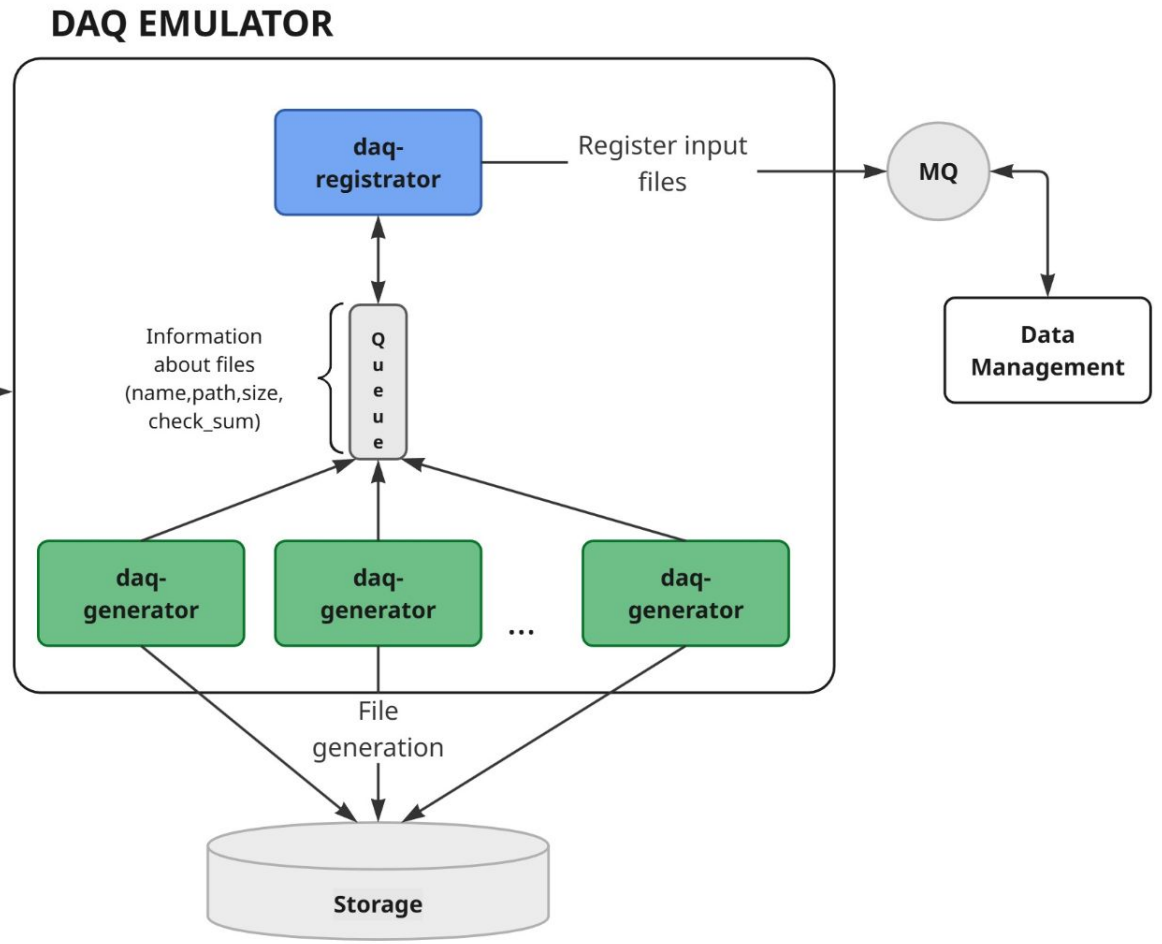
1. Three steps in the test workflow.
2. 50 jobs per task, 50 input files of ~50Mb.
3. The full data life cycle was completed.



DAQ Emulator Service

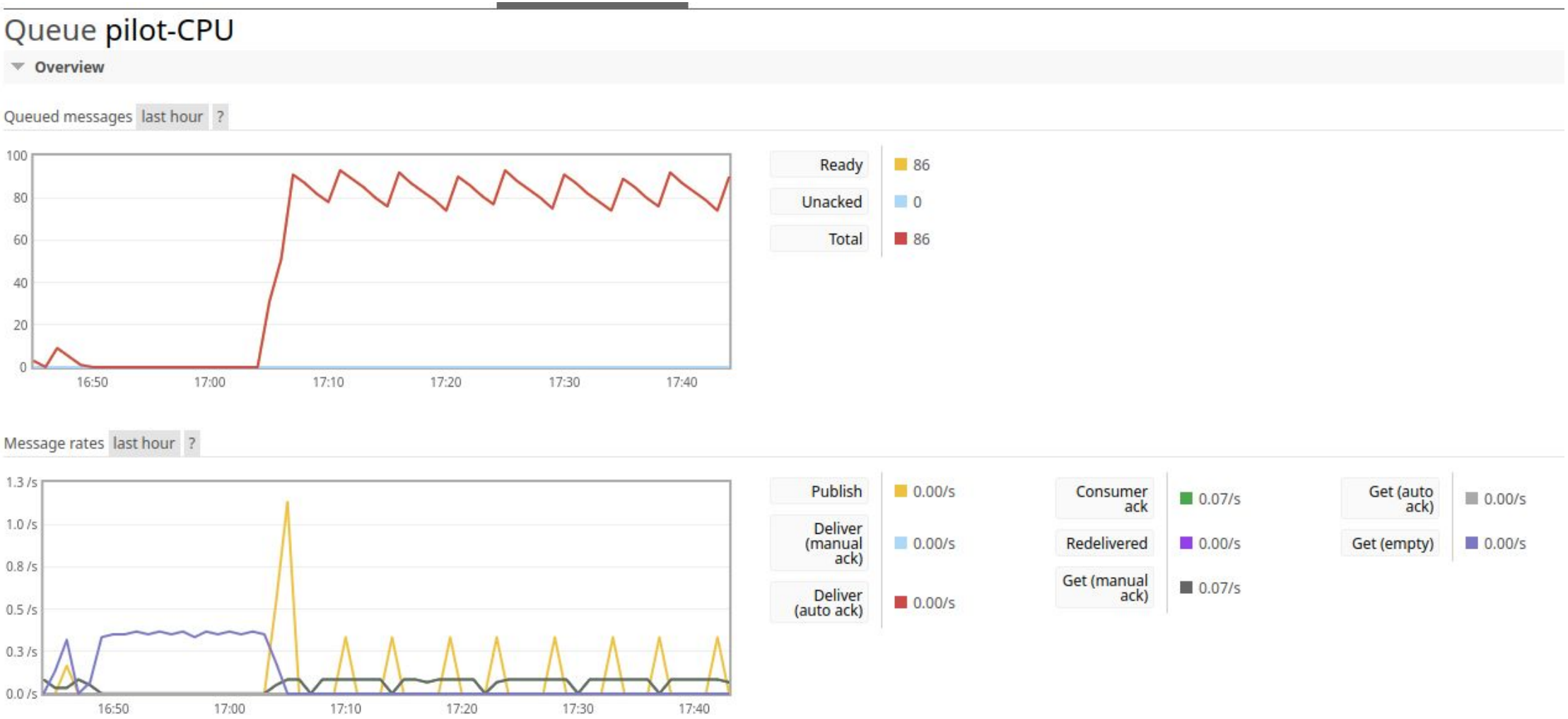
config.ini

```
[General]
number_of_generators = 5
number_of_runs = 10
sleep = 7200

[Files]
file_size = 50
number_of_files = 100

[Datasets]
name_template = "_run_"
dataset_size = 10
```



- Will allow us to perform complex loading tests.
- Several workflows at a time, several tasks/datasets.
- Continuous generation and registration of primary datasets.

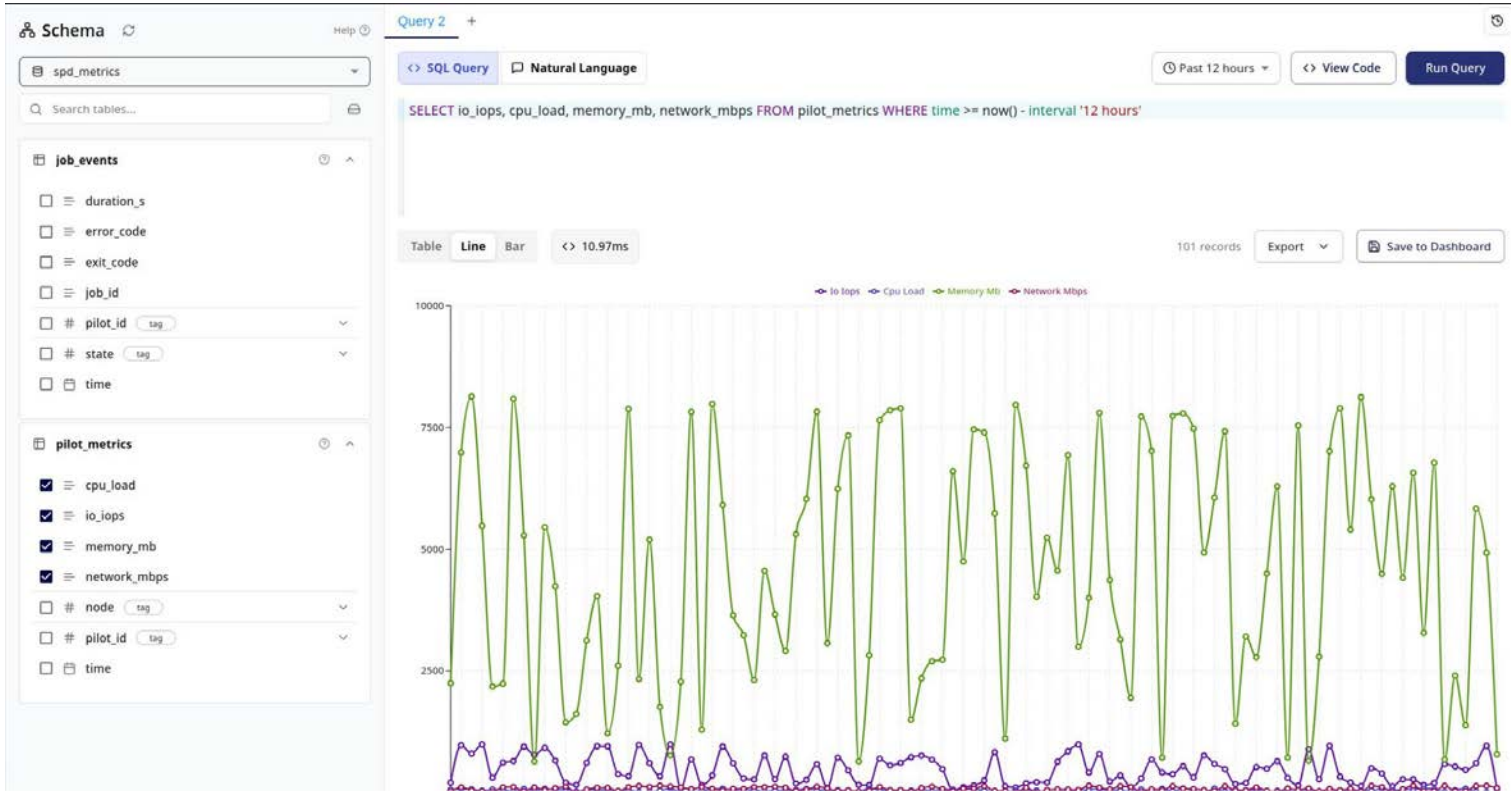
First results of DAQ Emulator Service



An example of the simultaneous execution of **ten workflows** on **ten primary datasets**. Each bump represents a new batch of jobs that are ready to be executed and have been published by the scheduler.

Conclusion

- **Data & Storage Management**
  - Robust and the most stable system.
  - Ongoing work on the uploading data from the output storage to Rucio.
- **Workflow Management System**
  - Requires more precise definition (design) of the templates.
  - Major refactoring cycle and possible redesign.
  - Lagging behind compared to other systems.
- **Workload Management System**
  - Mostly ready to perform complex load tests.
- **Nearest work**
  - Definition of requirements for monitoring system.
  - Integration with DAQ Emulator services and performing primary load tests.

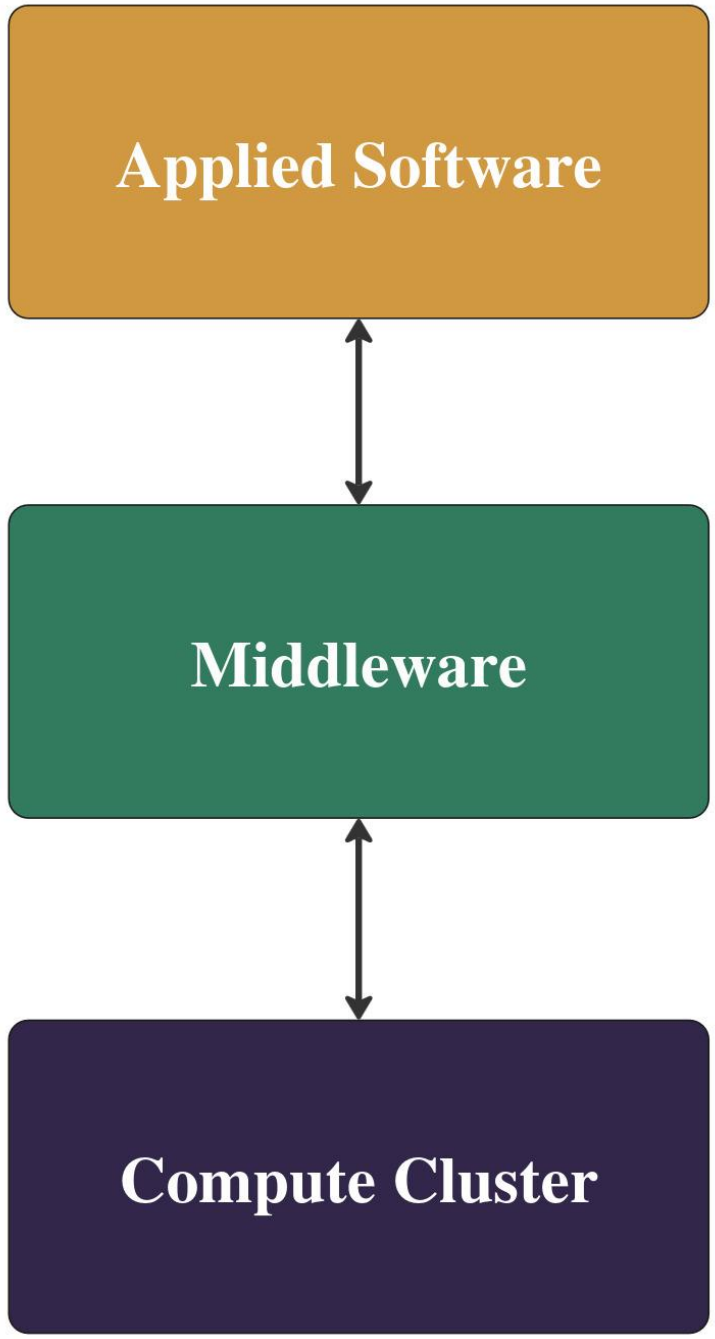


An example of imported **preliminary metrics** on the InfluxDB Explorer dashboard.

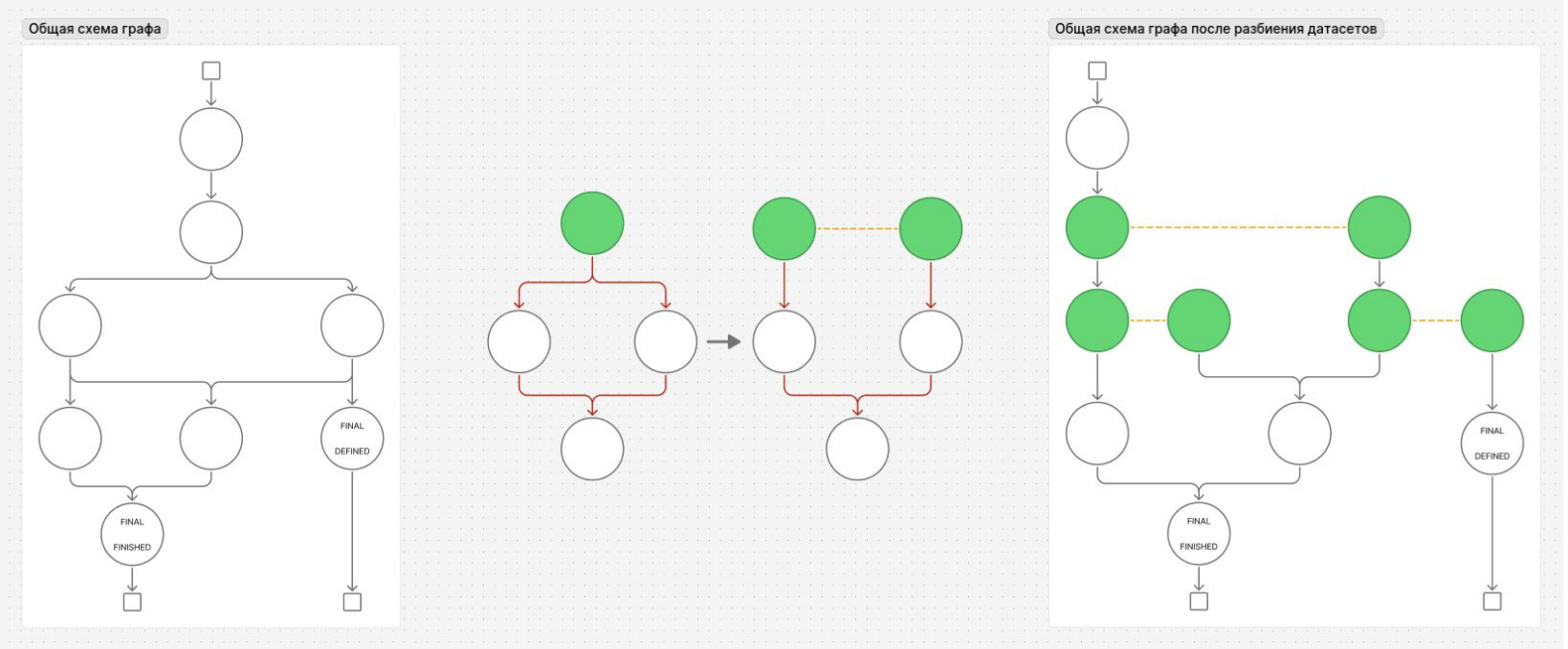
Next steps/milestones

- ✅ **Workflow processing has been achieved - entire data lifecycle tested!**
- 🔵 **DAQ Emulator service**
- 🔵 **Collecting requirements for monitoring system**
- ❓ **Middleware deployment and release management**
  - ❑ Deploy pilots on the testbed - test job execution on a large scale.
  - ❑ Focus on shipping SPD Online Filter as standalone software.
  - ❑ Work on the deployment on the **upcoming testbed** (256 CPU Cores, 1TB RAM, 120TB HDD).
- ❓ **Middleware and applied software integration**
  - ❑ Requires prototyped applied software and simulated data.
  - ❑ Non-functional requirements for applied software.
  - ❑ Move to the execution of the jobs on the pilot with a "real" payload.

Backup slides


Applied Software
Middleware
Compute Cluster

Design of the templates - important. How they will look like?



“One dataset per task” policy

name character varying (255)	meta_data json	status_code character varying (20)
raw_run_07102025_035420.log.3	{"task_id": 82}	CLOSED
raw_run_07102025_035420.output.3	{"task_id": 82}	CLOSED
raw_run_07102025_035420.log.2	{"task_id": 81}	CLOSED
raw_run_07102025_035420.output.2	{"task_id": 81}	TO_DELETE
raw_run_07102025_035420.log.1	{"task_id": 80}	CLOSED
raw_run_07102025_035420.output.1	{"task_id": 80}	TO_DELETE

Intermediate datasets marked to be deleted

Thoughts about monitoring

1. Technical monitoring - for operations:

- incoming / outgoing RPS (requests per second)
- number of successful / failed requests
- count of error / warn logs
- messages sent via broker and percent failures
- number of lost messages
- resource usage: CPU, memory, disk

Technical metrics: come from services, brokers and DB health (not DB content)

2. Business monitoring - about data processing workflows:

- number of tasks in the queue
- detailed info for each task and its subtasks (jobs)
- ratio of successful / failed tasks
- other process metrics as needed

Business metrics: require database queries (on update or periodic)

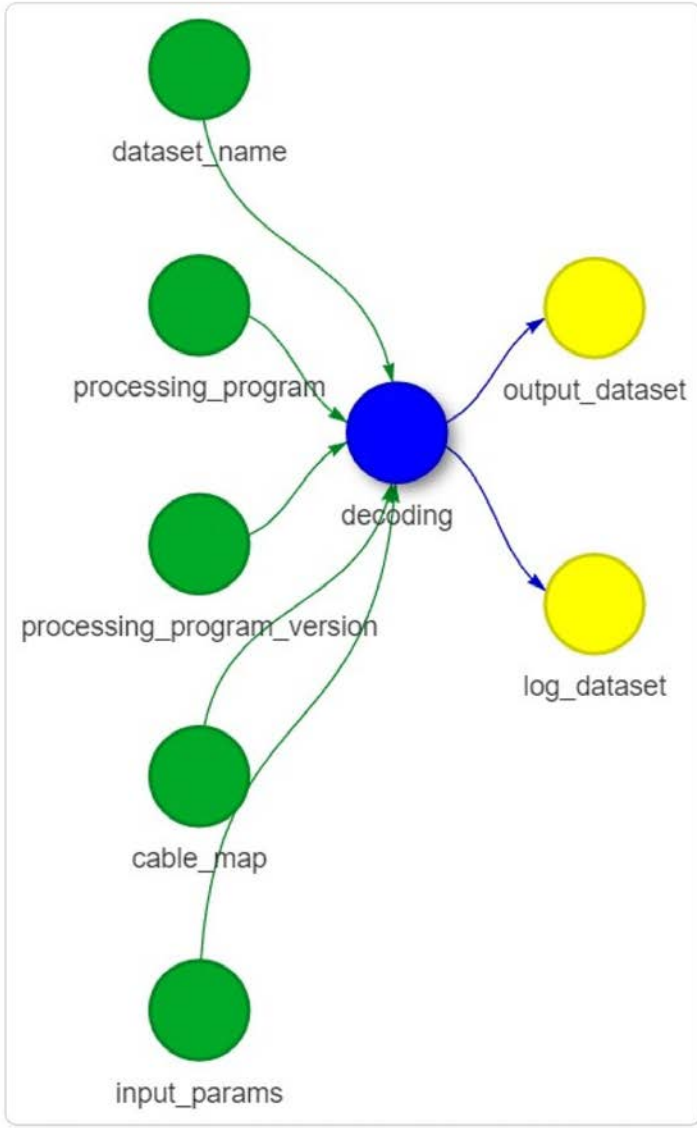
Workflow Management System - Core logic

The main objectives of Workflow Management System:

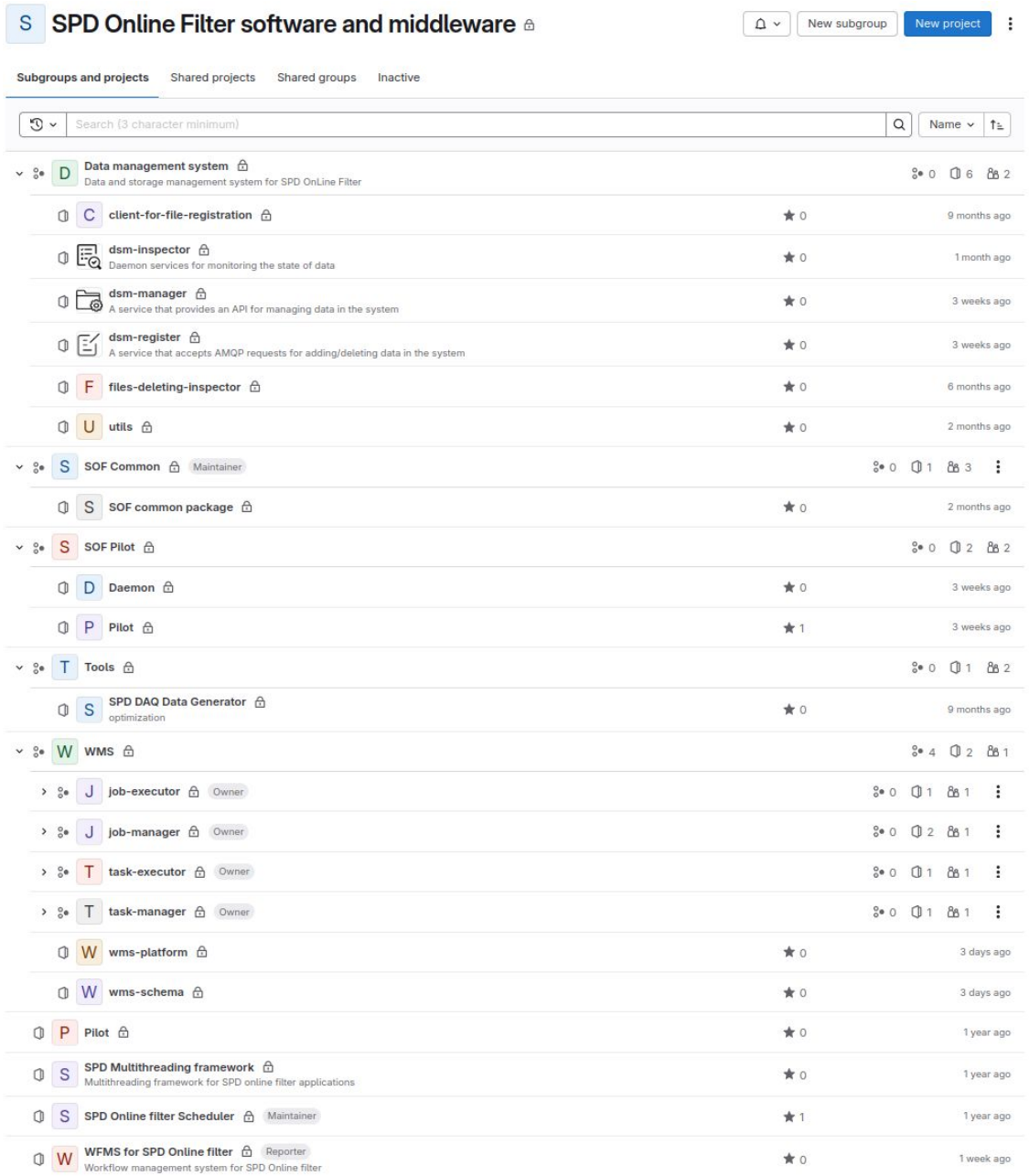
1. Retrieves input datasets from Data Management System;
2. Maps these datasets with the appropriate CWL template;
3. Generates the workchain from this template;
4. Generates tasks and sends them to the Workload Management System for further execution;
5. Oversees datasets: decision making for creation, closure, deletion;
6. Manages the concurrent execution of workchains and tasks.

Пример CWL-шаблона

```
cwlVersion: v1.2
label: Decoding of data
class: Workflow

inputs:
  dataset_name: string
  processing_program: string
  processing_program_version: string
  cable_map: File
  input_params: File
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: dataset_name
      processing_program: processing_program
      processing_program_version: processing_program_version
      cable_map: cable_map
      input_params: input_params

    out: [output_dataset, log_dataset]

outputs:
  output_dataset:
    type: File
    outputSource: decoding/output_dataset
  log_dataset:
    type: File
    outputSource: decoding/log_dataset
```



SPD Online Filter middleware code base



- ➢ Around ~25 000 lines of code for the entire SPD Online Filter Middleware;
- ➢ Full deployment requires ~16 Docker containers: one container per microservice;
- ➢ Configured CI/CD pipeline, currently only for the Workload Management System;
- ❗ May need to be reorganized to deploy as a standalone project on the testbed
  - Hardware for the prototyping of a compute cluster
- ❗ Deploying Pilot Agents to Compute Nodes.

DAQ data generator

1. Using **SPD DAQ Data Generator**, we've generated **50 files**, each ~**2Gb**;
2. Input dataset has been registered with these files;
3. Task has been processed (or 50 jobs);
4. The payload for **Pilot** is simple: compute the MD5/BLAKE3 hash, as there is no actual computation involved at this stage.;
5. Takes about ~**7 min** to generate a file, using JINR Cloud VM: 12x 1-core Intel Xeon E5-2650
6. Registration of the entire dataset: ~**10 sec**

```
# Configuration file for SPD DAQ data generator
# 2023/03/01

#Data file name format: run-<run number>-<chunk
number>-<builder id>.spd
DataFileNameFormat = run-%06u-%05u-%02u.spd

#RND generator seed:
RandomSeed = 12345

#The size limit of the output data file in bytes:
DataFileSizeLimit  = 2147483648

#debug mode for debuging front-end card. If it is 1
then generator will
#produce all data words (headers and trailers) even
if there are no hits,
#otherwise all empty data blocks are removing
DebugMode          = 0

#Source ID(s) of the clock modulue(s) for
measurement start of frame time:
FrameClockID       = 1000,1001

#Source ID(s) of the TDC module(s) for measurement
of the bunch crossing time:
BunchCrossingID    = 1004

#Slice length in ns (must be less than smallest TDC
over-roll time (4.5 ms for RS)):
SliceLength        = 10000

#Number of slices in a frame:
FrameLength        = 100000
```

Future plans

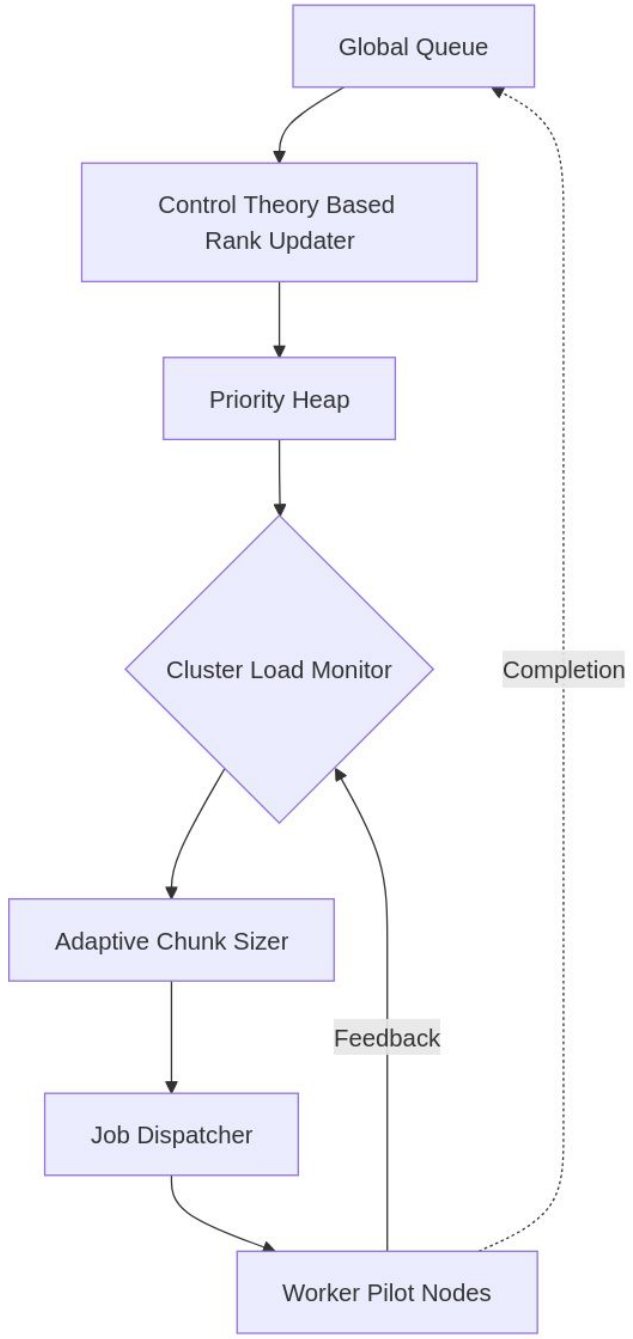
job-executor (Scheduler)

1. Expected to process tasks from a global queue;
2. Each dataset has a rank (priority) that determines its processing order;
3. Tasks are processed in priority order, with dynamic updates to maintain system responsiveness;
4. **Priority-based job scheduling mechanism** is expected, with rank update scheme involving **Control Theory** (option to be explored later);
5. Not applicable at this stage of the development process.

$$\mathrm{r}_{i+1} = \underbrace{\alpha \ln(x_i + 1)}_{\text{Aging}} - \underbrace{\beta 2^{y_i}}_{\text{Retry Penalty}} + \underbrace{\gamma r_i}_{\text{History}} + \underbrace{\delta(1 - L)}_{\text{Load}}$$

$$\mathbf{r}_{i+1} = \Gamma \mathbf{r}_i + \alpha \ln(\mathbf{x}_i + \mathbf{1}) - \beta \cdot 2^{\mathbf{y}_i} + \delta(1 - L)\mathbf{1}$$

$$\Gamma = \mathrm{diag}(\gamma_1, ..., \gamma_N) \text{ (job-specific history weights)}$$
$$\mathbf{x}_i = [x_i^{(1)}, ..., x_i^{(N)}]^\top \text{ (job ages)}$$
$$\mathbf{y}_i = [y_i^{(1)}, ..., y_i^{(N)}]^\top \text{ (retry counts)}$$



Task-executor (Scheduler)

Continuous-Time Domain

The original aging term is the following:

$$\alpha \ln(x(t) + 1)$$

With lead-lag compensation, should be

$$\alpha_{\text{adj}}(t) = \mathcal{L}^{-1}\left[\frac{1 + \tau_1 s}{1 + \tau_2 s} \cdot \mathcal{L}(\alpha \ln(x(t) + 1))\right]$$

Heaviside step function $H(t-t_k)$ introduces instantaneous jumps at retry times t_k

$$y(t) = \sum_{k=1}^{N_{\text{retry}}} H(t - t_k)$$

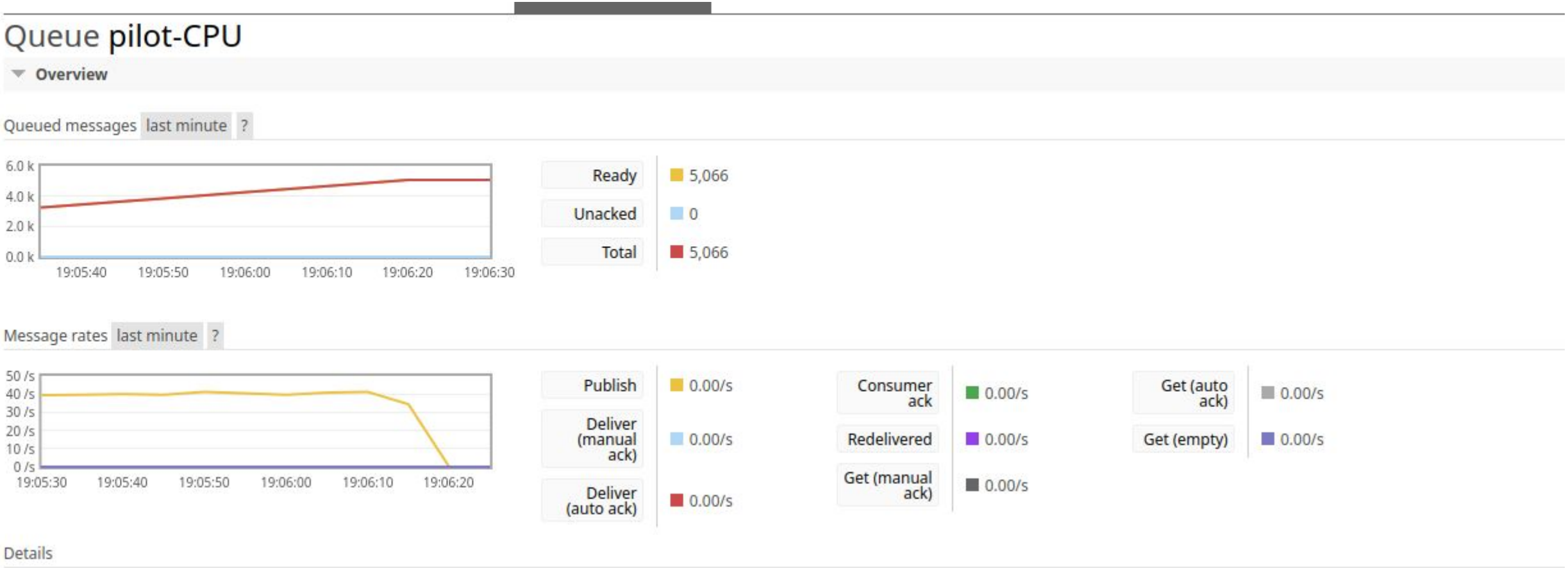
And retry penalty depends on past events (retry history), making the system state depend on its history, so we have a **delay differential equation**, which models the “physics” of retry-driven rank adjustments of our jobs

$$\frac{dr}{dt} = (\gamma - 1)r(t) + \alpha_{\text{adj}}(t) - \beta 2^{y(t)} + \delta(1 - L)$$

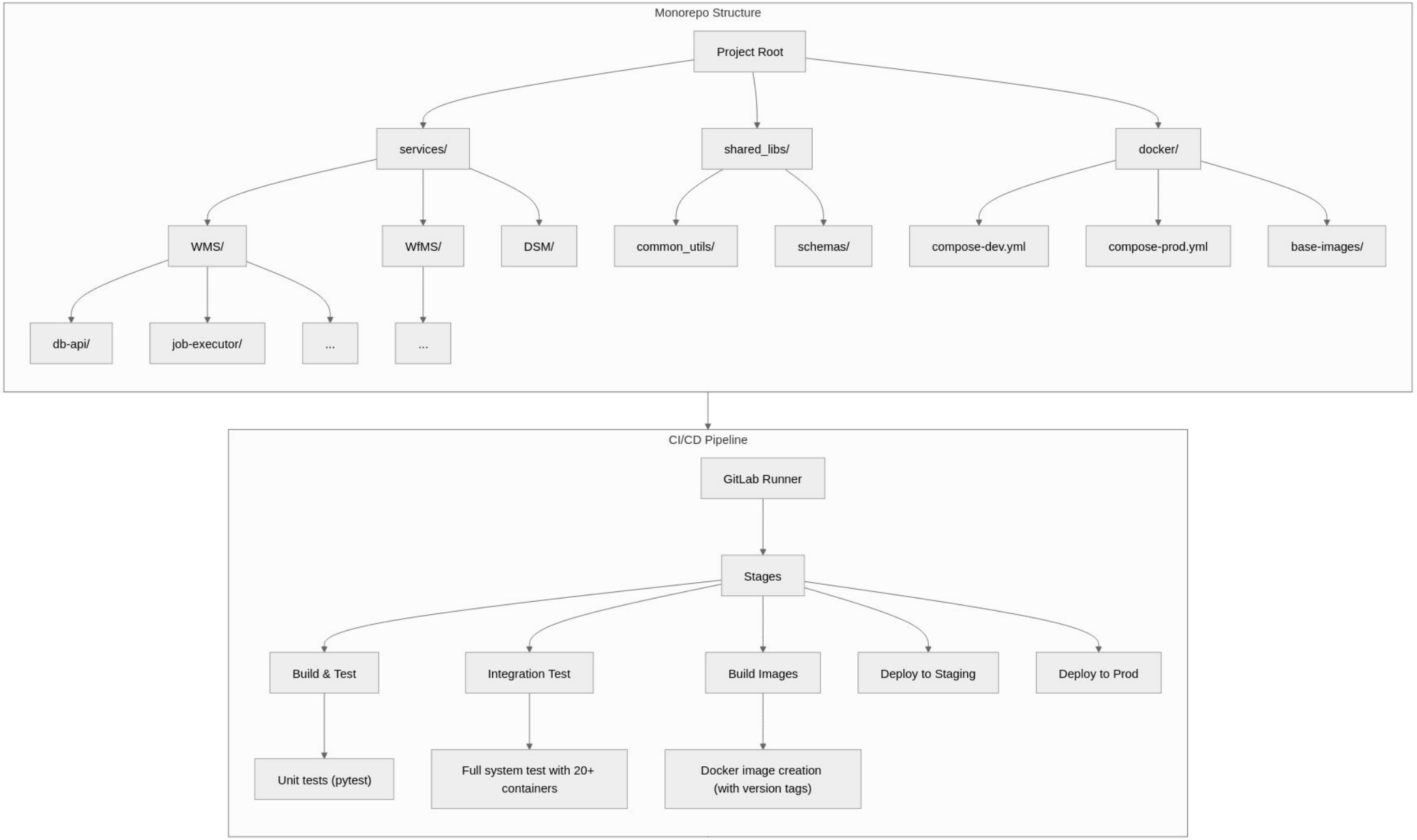
	Discrete Simulation	Continuous Solution
Retries	Exact event times	Requires Dirac delta
Implementation	Matches real code	Theoretical analysis
Stability	Bounded by design	Must prove convergence?
Visualization	Step changes	Smooth curves (Runge-Kutta Solver?)

First “load testing”

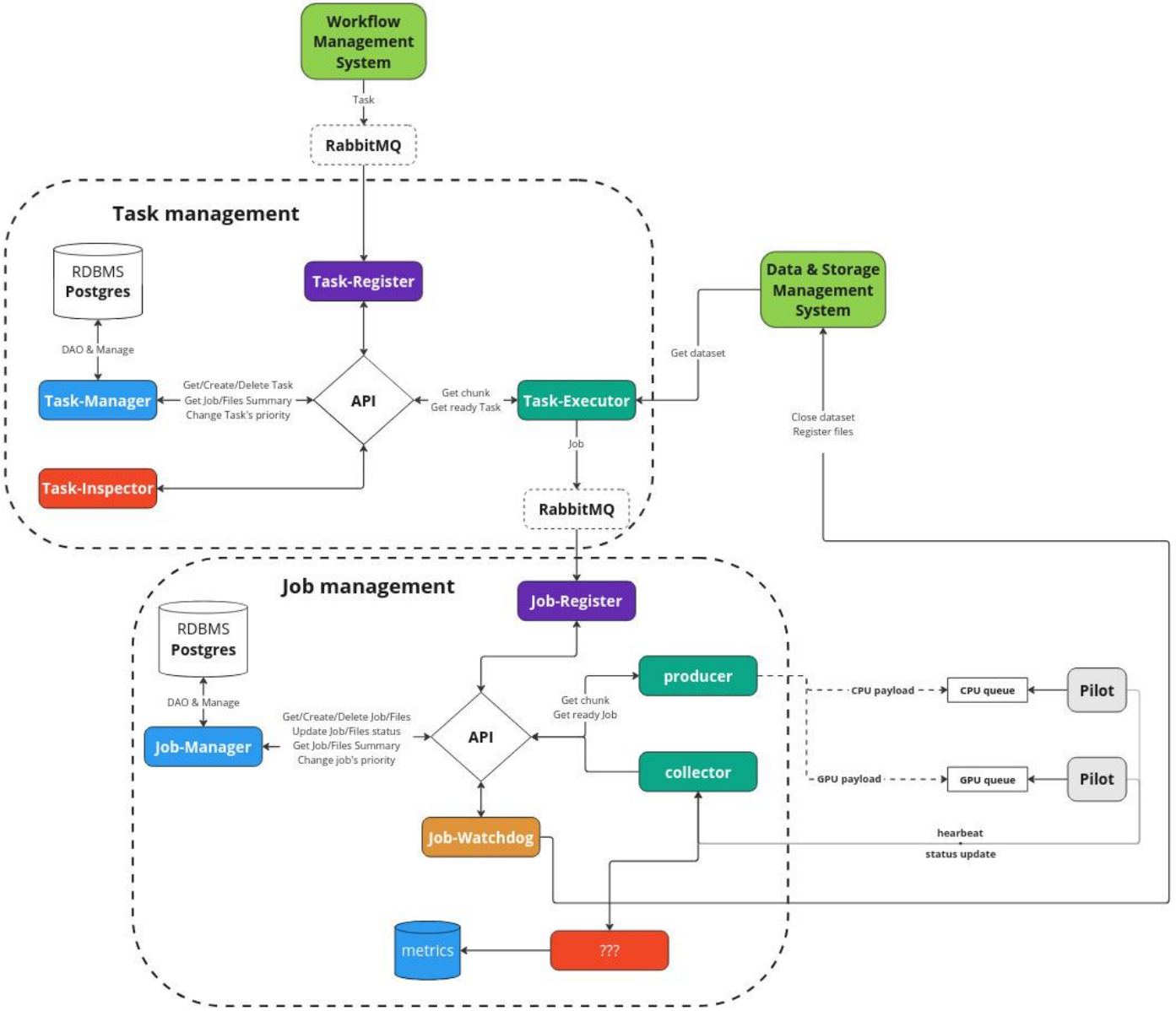
1. Workload Management System generates ~5000 jobs in less than a minute
2. Must be tested on meaningful data and payload, the system may not need to be over engineered more



Gitlab project structure for CI/CD



Workload Management System

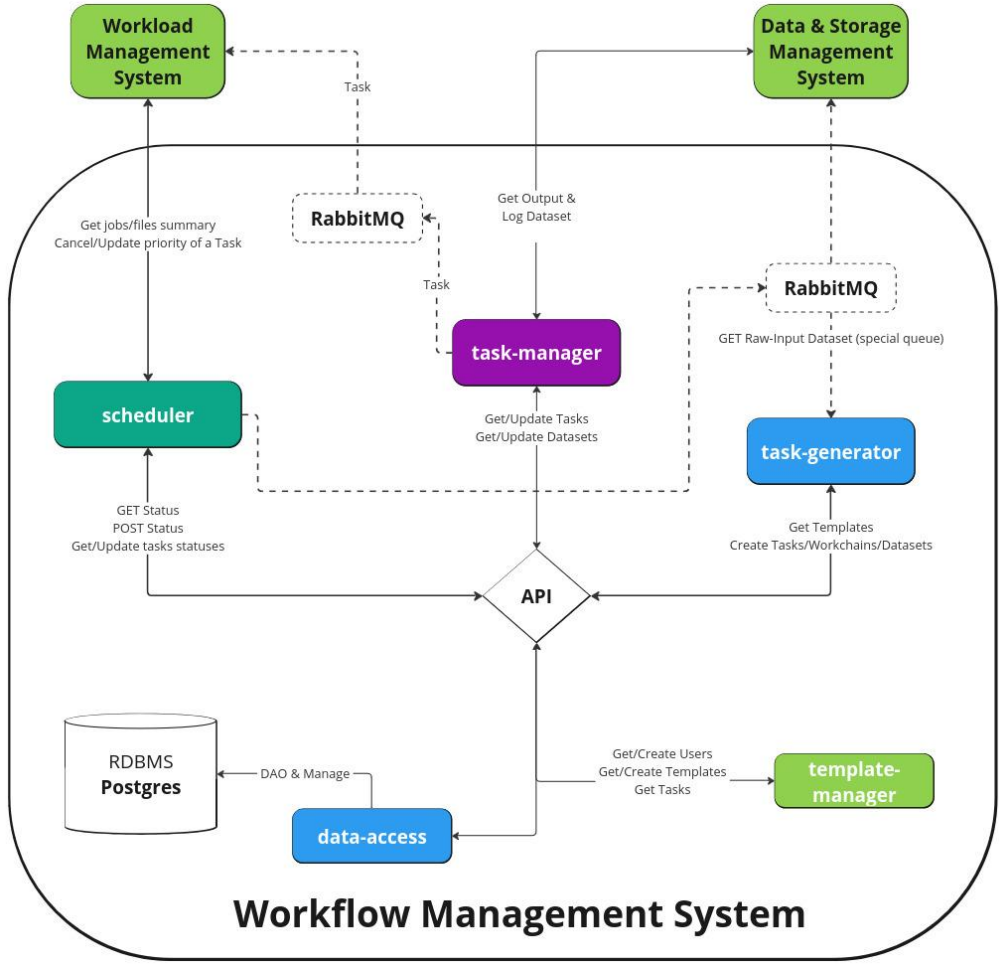


Next steps:

1. Implement metric-collection service;
2. Monitoring service, traces collection.

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

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

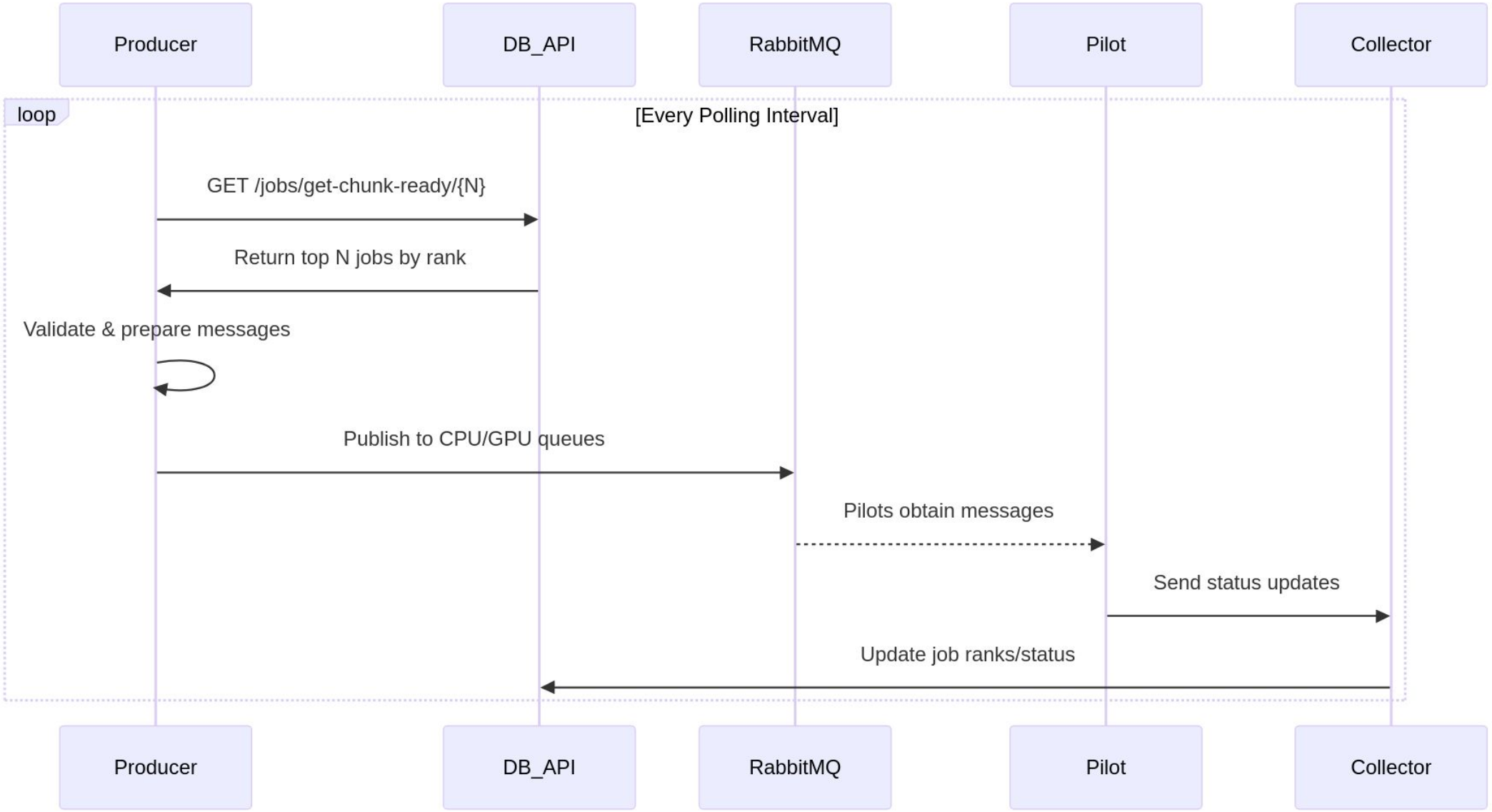
Workload management system requirements - reminder

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

Workload Management System - Pilot Agent



Data consistency

Curl

```
curl -X 'GET' \
  'http://10.220.16.177:8080/api/v1/file/0fb1e1af-5c02-4d17-87a9-64debf5e6a17' \
  -H 'accept: application/json'
```

Request URL

```
http://10.220.16.177:8080/api/v1/file/0fb1e1af-5c02-4d17-87a9-64debf5e6a17
```

Server response

Code	Details
200	Response body

```
{
  "name": "input.test.a976a020-3de5-44e2-91ee-319e426eda2f.raw",
  "path": "input_40",
  "storageId": "b3307ad4-f2b3-4f3a-a390-4f4e2762c620",
  "size": 50,
  "checkSum": "c1349c048472b4cebd57669e1558b72a",
  "statusCode": "CREATED",
  "id": "0fb1e1af-5c02-4d17-87a9-64debf5e6a17"
}
```

Curl

```
curl -X 'GET' \
  'http://10.220.16.177:8080/api/v1/dataset/?file_id=0fb1e1af-5c02-4d17-87a9-64debf5e6a17' \
  -H 'accept: application/json'
```

Request URL

```
http://10.220.16.177:8080/api/v1/dataset/?file_id=0fb1e1af-5c02-4d17-87a9-64debf5e6a17
```

Server response

Code	Details
200	Response body

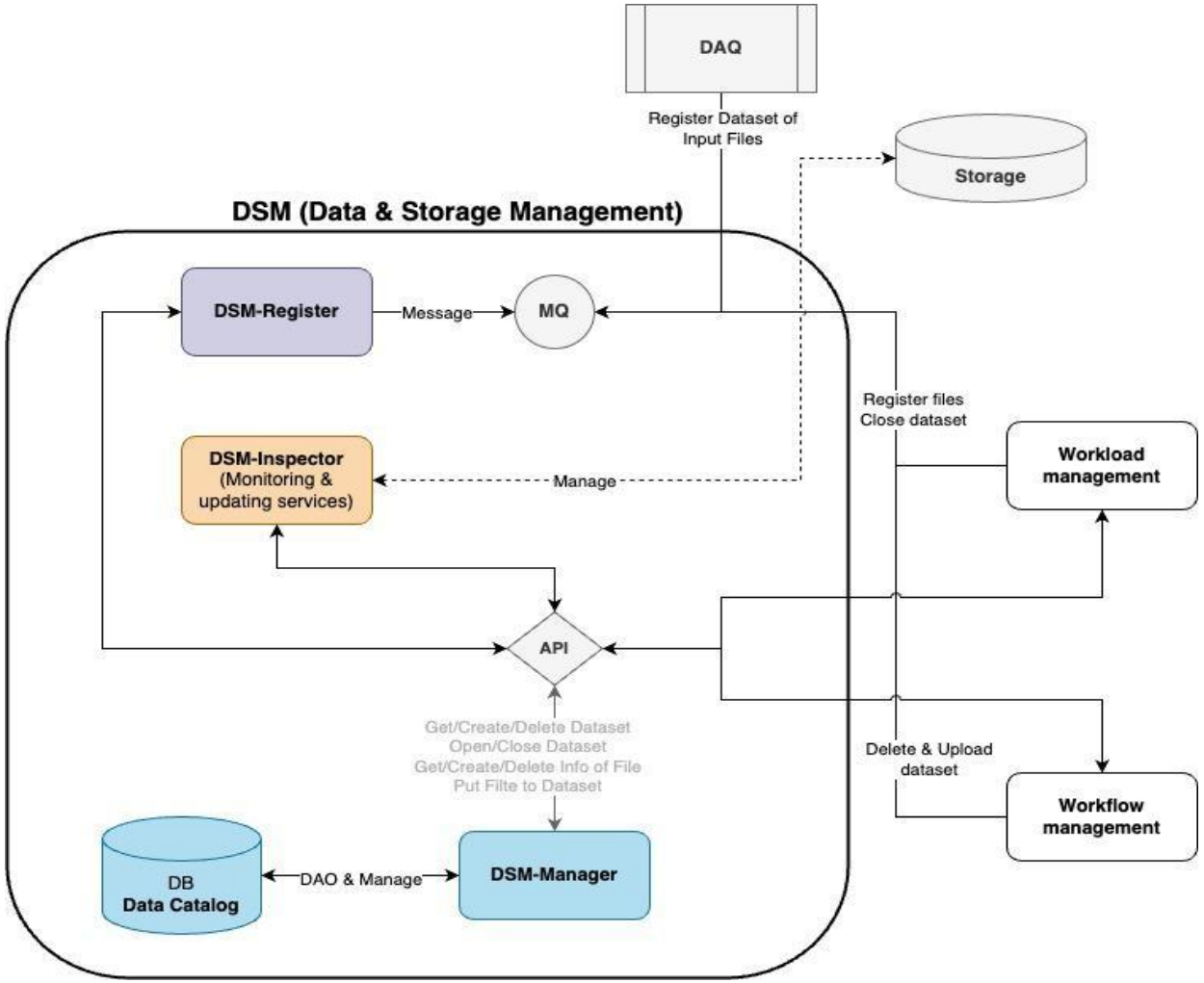
```
[
  {
    "name": "input.test.b255570d-33bf-4dc3-9e6e-718df9a1a8ef.raw",
    "metaData": {
      "run_number": 49,
      "files": 10
    },
    "statusCode": "CLOSED",
    "id": "f61828be-64b5-44e8-9d18-a1a22068094d"
  }
]
```

	id [PK] uuid	name character varying (255)	path character varying (255)	storage_id uuid	size integer
1	0fb1e1af-5c02-4d17-87a9-64debf5e6a17	input.test.a976a020-3de5-44e2-91ee-319e426eda2f.raw	input_40	b3307ad4-f2b3-4f3a-a390-4f4e2762c620	50

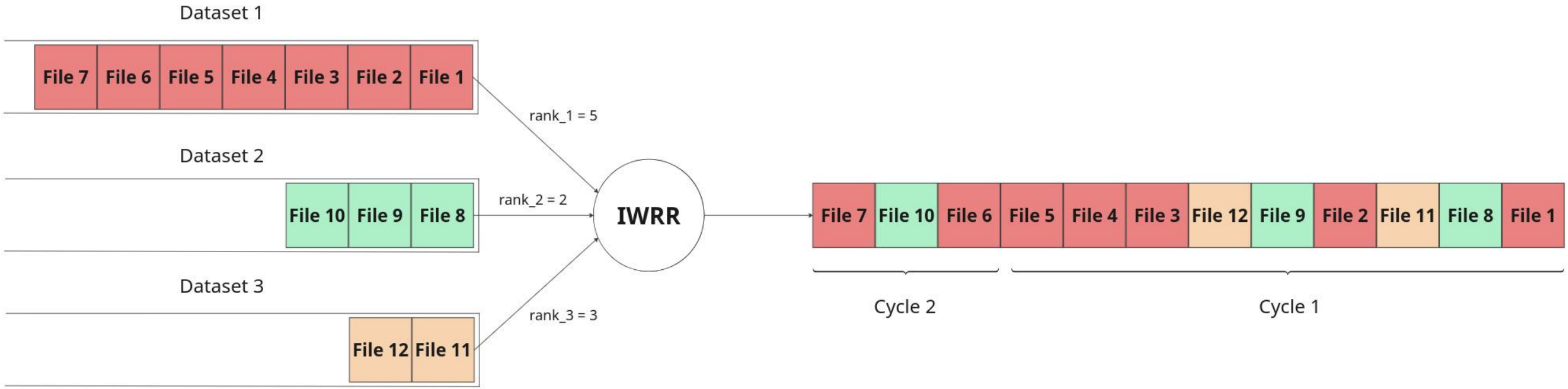
	id [PK] uuid	name character varying (255)	path character varying (255)	storage_id uuid	size integer
1	0fb1e1af-5c02-4d17-87a9-64debf5e6a17	input.test.a976a020-3de5-44e2-91ee-319e426eda2f.raw	input_40	b3307ad4-f2b3-4f3a-a390-4f4e2762c620	100

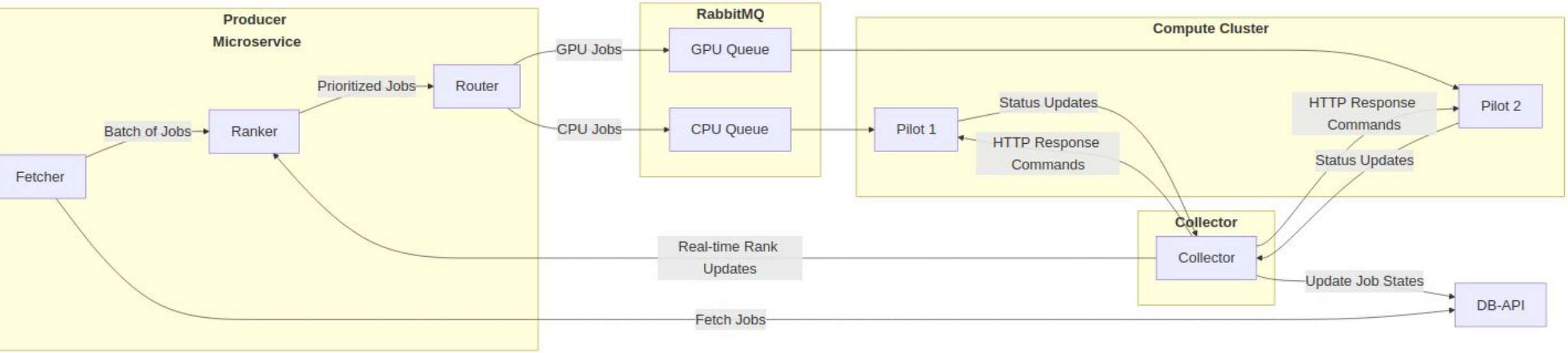
Data & Storage Management

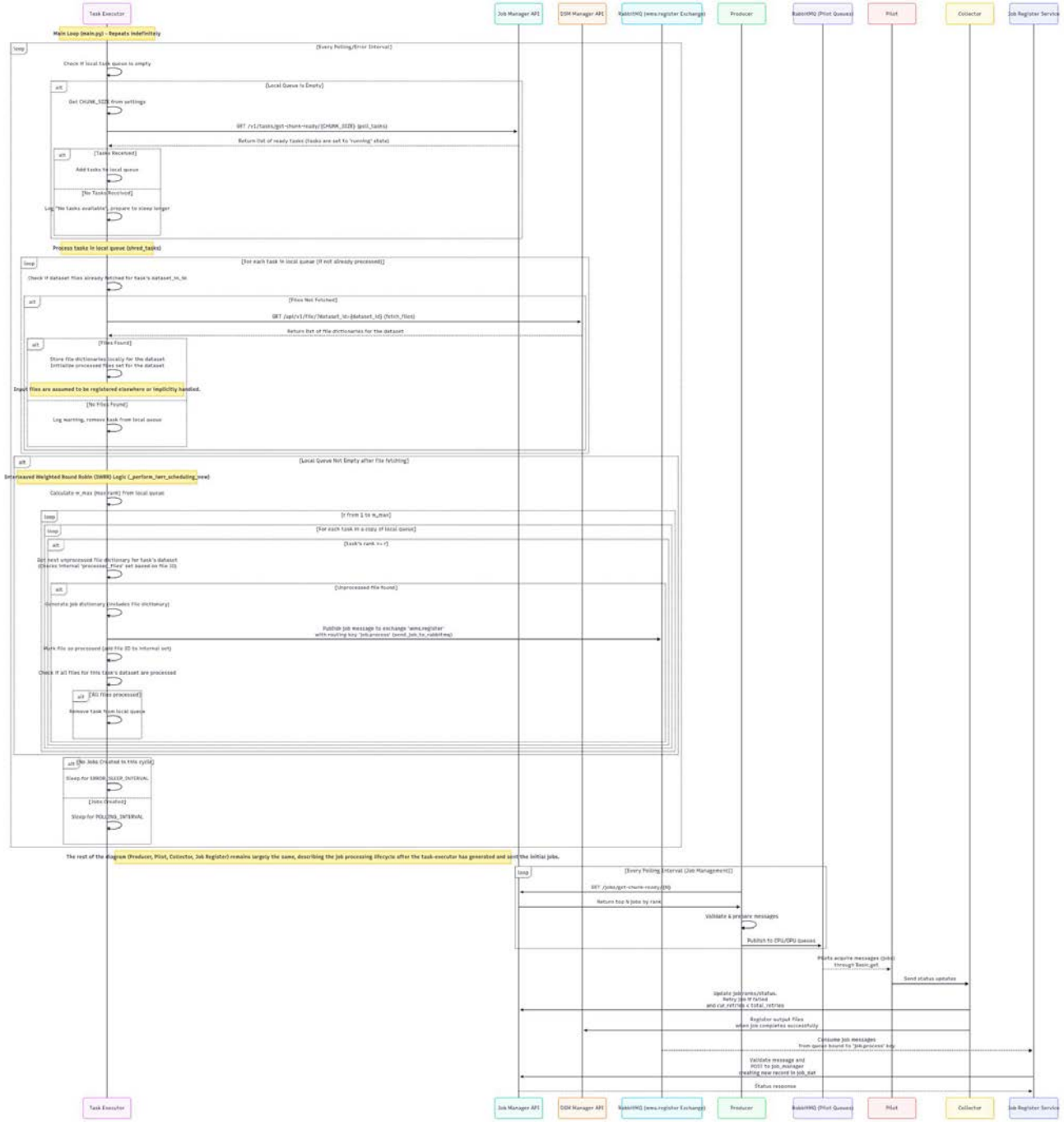
1. **DSM-Register (Data Registration)**:
   a. Create a new consumer for the queue dsm.register.dataset.delete
   b. Write a correspondent message handler
2. **DSM-Manager (REST API of data catalog):**
   a. Getting the list of files/datasets by status
   b. Searching for a file by name
3. **DSM-Inspector (Daemon tasks)**:
   a. Storage monitoring service for dark files
   b. Checking file integrity
   c. Deleting files and datasets



Architecture of Data Management System


Dataset 1
File 7
File 6
File 5
File 4
File 3
File 2
File 1
rank_1 = 5
Dataset 2
File 10
File 9
File 8
rank_2 = 2
IWRR
Dataset 3
File 12
File 11
rank_3 = 3
File 7
File 10
File 6
File 5
File 4
File 3
File 12
File 9
File 2
File 11
File 8
File 1
Cycle 2
Cycle 1


Producer
Microservice
Fetcher
Batch of Jobs
Ranker
Prioritized Jobs
Router
GPU Jobs
CPU Jobs
RabbitMQ
GPU Queue
CPU Queue
Compute Cluster
Pilot 1
Status Updates
HTTP Response
Commands
Pilot 2
HTTP Response
Commands
Status Updates
Collector
Collector
Real-time Rank
Updates
Update Job States
DB-API
Fetch Jobs


Task Executor
Job Manager API
DIM Manager API
RabbitMQ (www.register Exchange)
Producer
RabbitMQ (Pilot Queues)
Pilot
Collector
Job Register Service
Main Loop (main.py) - Repeats indefinitely
[Every Polling/Error Interval]
Check if local task queue is empty
alt
[Local Queue Is Empty]
Get CHUNK_SIZE from settings
GET /v1/tasks/get-chunk-ready/{CHUNK_SIZE} (poll_tasks)
Return list of ready tasks (tasks are set to 'running' state)
alt
[Tasks Received]
Add tasks to local queue
[No Tasks Received]
Log "No tasks available", prepare to sleep longer
Process tasks in local queue (shred_tasks)
loop
[For each task in local queue (if not already processed)]
Check if dataset files already fetched for task's dataset_in_id
alt
[Files Not Fetched]
GET /api/v1/file/?dataset_id={dataset_id} (fetch_files)
Return list of file dictionaries for the dataset
alt
[Files Found]
Store file dictionaries locally for the dataset
Initialize processed files set for the dataset
Input files are assumed to be registered elsewhere or implicitly handled.
[No Files Found]
Log warning, remove task from local queue
[Local Queue Not Empty after File Fetching]
Interleaved Weighted Round Robin (IWRR) Logic (_perform_iwrr_scheduling_new)
Calculate w_max (max rank) from local queue
loop
[r from 1 to w_max]
loop
[For each task in a copy of local queue]
[Task's rank >= r]
alt
Get next unprocessed file dictionary for task's dataset
(Checks internal 'processed_files' set based on file ID)
alt
[Unprocessed file found]
Generate job dictionary (includes file dictionary)
Publish job message to exchange 'www.register'
with routing key 'job.process' (send_job_to_rabbitmq)
Mark file as processed (add file ID to internal set)
Check if all files for this task's dataset are processed
alt
[All files processed]
Remove task from local queue
alt
[No Jobs Created in this cycle]
Sleep for ERROR_SLEEP_INTERVAL
[Jobs Created]
Sleep for POLLING_INTERVAL
The rest of the diagram (Producer, Pilot, Collector, Job Register) remains largely the same, describing the job processing lifecycle after the task-executor has generated and sent the initial jobs.
loop
[Every Polling_Interval (Job Management)]
GET /jobs/get-chunk-ready/{N}
Return top N jobs by rank
Validate & prepare messages
Publish to CPU/GPU queues
Pilots acquire messages (jobs)
through Basic.get
Send status updates
Update job.progress,status,
Retry job if failed
and cur_retries < total_retries
Register output files
when job completes successfully
Consume job messages
from queue bound to 'job.process' key
Validate message and
POST to job_manager
creating new record in job_dat
Status response