



Christopher Gwilliams and Stephan Egli :: Paul Scherrer Institute

SciCat Project: Data Catalog System

- 1 Introduction
- 2 Data Catalog Purpose
- 3 Data Model
- 4 Architecture Goals and Rationale
- 5 Architecture Overview
- 6 How do we run it?
- 7 Development Process
- 8 Collaboration

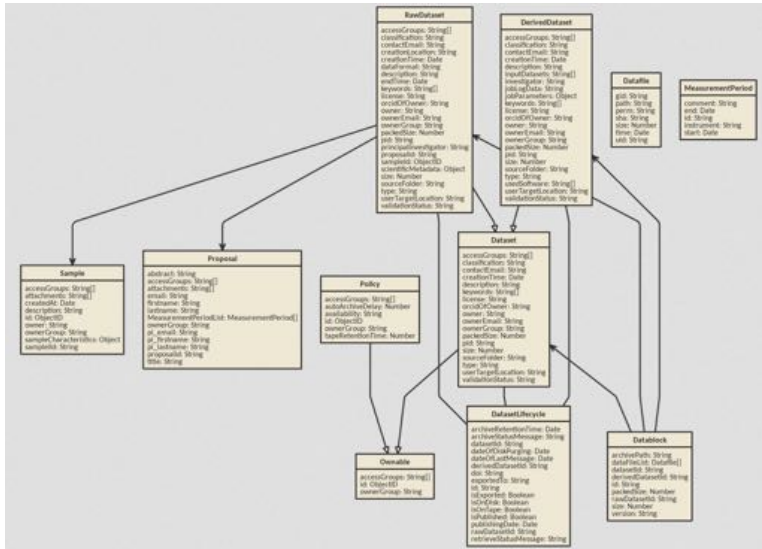
- A collaboration between PSI and ESS (and MaxIV) to create a data catalog management system
- Supporting the management of the whole data lifecycle
- Open source (<https://github.com/scicatproject>)
- Built with the requirements of scientists in the early stages
- Microservice architecture with the latest technologies

- 1 Introduction
- 2 Data Catalog Purpose
- 3 Data Model
- 4 Architecture Goals and Rationale
- 5 Architecture Overview
- 6 How do we run it?
- 7 Development Process
- 8 Collaboration

- Manage the **meta data** of raw and derived data which is taken at experiment facilities
- Meta data
 - **administrative** : data management lifecycle, ownership, filecatalog
 - **scientific**: describing the sample, beamline and experiment parameters relevant for the users data analysis
- Enables management of the lifecycle of the data from creation , data analysis and eventual deletion
 - Data can be linked to proposals and samples
 - Data can be linked to publications (DOI, PID)
 - Data can be migrated to and from longterm storage on tape
- Helps keeping track of data provenance (i.e. the steps leading to the final results)
- Allows to check scientific integrity (checksum of data)
- Allows to **find data** based on the meta data (your own data and other peoples public data)
- In the long term:
 - help to automate standardized analysis workflows
 - support the standardization of data formats

- 1 Introduction
- 2 Data Catalog Purpose
- 3 Data Model**
- 4 Architecture Goals and Rationale
- 5 Architecture Overview
- 6 How do we run it?
- 7 Development Process
- 8 Collaboration

- Meta data is linked to **Datasets**, which are **collection of files**, e.g. all files produced during a data taking run
- Each dataset gets a globally unique persistent identifier (PID)
- Each dataset is uniquely assigned to one pgroup
- Only members of the pgroup have access to the raw data and meta data belonging to the pgroup
- Only after the embargo period (typically 3 years) the data becomes public
- The pgroup membership can be defined via processes supported by the digital user office DUO (Roles: BM, PI, MP)
- The pgroups are stored centrally in the AD Identity Management system
- Data model: Define common generic (fixed) meta data and allow at the same time completely flexible and rich structured (beamline, instrument specific) scientific meta data



- Scientific meta data is up to the beamline managers to define (in collaboration with the users)
- Aim for standardization, e.g. via use of HDF5 and Nexus formats
- The catalog per se does not pose any limits here
- See example on next page. . .

Home / Dataset / 20.500.11935/00013c34-3c42-40c4-b9b9-6d01e7f4b997

Details

Delete file

Principal Investigator
 alexandra.peter@psi.ch

End Time
 04/04/2017 22:54

Creation Location
 JPS/SLS/TOMCAT

Data Format
 Tomcat-pro 2017

Scientific Metadata

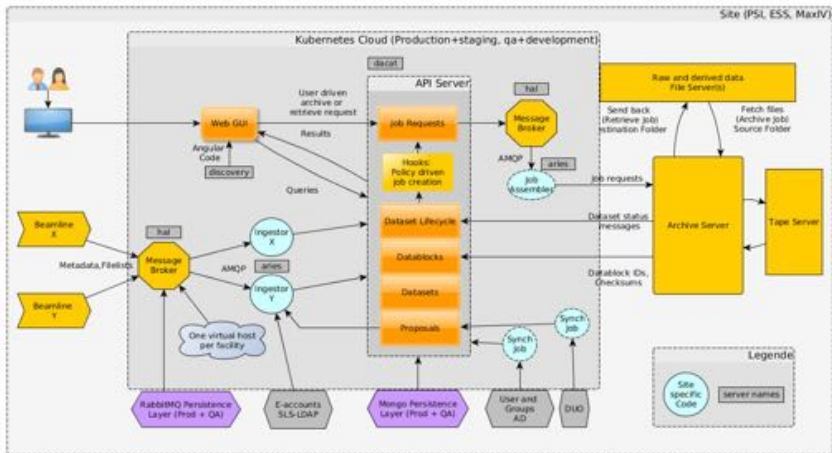
Name	Value
<ul style="list-style-type: none"> • SearchParameters <ul style="list-style-type: none"> OP-Fiber2 10um-Cu • Ring-current <ul style="list-style-type: none"> OP-Fiber1 100um-Al OP-Fiber3 10um-Fe Monostripe W-Si • Beam-energy <ul style="list-style-type: none"> FE-Fiber Fiber 50% • detectorParameters <ul style="list-style-type: none"> X-ROI Start 1 • Microscope x position <ul style="list-style-type: none"> x -0.22480 y m • Exposure time <ul style="list-style-type: none"> X-ROI End 2560 Y-ROI End 2160 • Microscope y position <ul style="list-style-type: none"> Objective 10 Microscope Cyt. Peter Millip Camera PCO Edge 5.5 	

- 1 Introduction
- 2 Data Catalog Purpose
- 3 Data Model
- 4 Architecture Goals and Rationale**
- 5 Architecture Overview
- 6 How do we run it?
- 7 Development Process
- 8 Collaboration

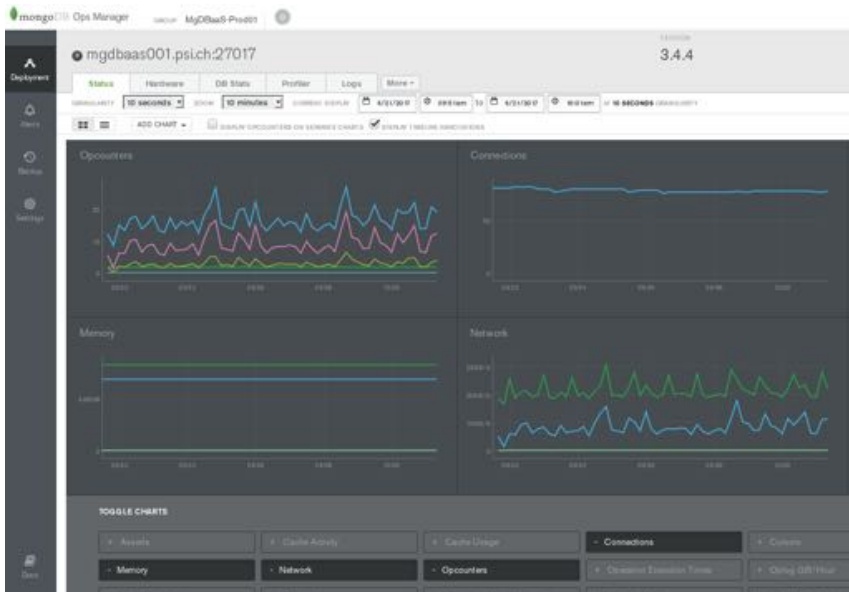
- Flexibility in terms of
 - covering the needs of the researchers , especially in terms of scientific meta data handling
 - integration into existing environments
 - ease of interfacing
 - from the beginning have customization to and deployments in other sites in mind
- Speed of changes
 - allow to add new instruments easily
 - allow to add new features in short time
- Longterm stability without being constrained in meeting new requirements
 - allow for constant evolution both in terms of features and volume. This concerns the whole DevOps processes
- Optionally: enable users to make customizations themselves

- 1 Introduction
- 2 Data Catalog Purpose
- 3 Data Model
- 4 Architecture Goals and Rationale
- 5 Architecture Overview**
- 6 How do we run it?
- 7 Development Process
- 8 Collaboration

Architecture Graph and Data Flows



- MongoDB backend
- NoSQL - no fixed structure and common data format (kind of)
- Map/Reduce queries and the option to use Javascript as the query language
- Powerful indexing and support for file storage
- Fault tolerant and drivers for most languages



- Loopback (API creation framework based on NodeJS and Express)
- APIs auto generated from JSON configuration to configure data model
 - Can create SDKs for many target languages
- Plugins for authentication and backend agnostic
- Logic can be added in Javascript files
- Auto documenting
- Swagger based API creation (open format)
- User accounts (through Active Directory) and System accounts (with roles for different administrative activities within the system i.e. Archiving)
- Many authorisation methods available through NPM packages

StrongLoop API Explorer Token Not Set [Set Access Token](#)

Datafile [Show/Hide](#) | [List Operations](#) | [Expand Operations](#)

Dataset [Show/Hide](#) | [List Operations](#) | [Expand Operations](#)

- GET** /Datasets Find all instances of the model matched by filter from the data source
- PUT** /Datasets Update an existing model instance or insert a new one into the data source
- POST** /Datasets Create a new instance of the model and persist it into the data source

Response Class (Status 200)

Model: **Model Schema**

```

"string"
{
  "RepositoryOfRecord": "string",
  "ReliabilityFlag": true,
  "Version": "string",
  "TypeOfScienceMetadata": "string",
  "ArchiveLocation": "string",
  "DisposalTime": "2016-09-24",
  "Exported": true,
  "ExportedTo": "string",
  "Published": true,
}

```

Response Content Type

Parameters

Parameter	Value	Description	Parameter Type	Data Type
data	<input type="text"/>	Model instance data	body	Model: Model Schema

Parameter content type:

```

{
  "DOI": "string",
  "InitialFolderLocation": "string",
  "Size": 0,
  "Checksum": "string",
  "CreationTime": "2016-09-24",
  "CreationLocation": "string",
  "ParentLocation": "string"
}

```

- RabbitMQ queuing system
- Data published to queues using many protocols
- Data source and format does not matter



Message broker RabbitMQ Management - Ingesting from Beamlines

RabbitMQ Management

Paul-Scherrer-Institut (PSI) (CH) | https://hal-ga.psi.ch/#/

Cluster: rabbit@melanie-ga-rabbitmq-3454546293-hsmth (change) User: user (change) Log out
RabbitMQ 3.6.9 Erlang 17.4

Overview Connections Channels Exchanges Queues Admin Virtual host: All

Overview

▼ Totals

Queued messages (chart: last minute) [?]

Ready	0
Unacked	0
Total	0

Message rates (chart: last minute) [?]

Publish	24/s	Deliver (auto ack)	22/s
Publisher confirm	24/s	Consumer ack	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s
		Get (manual ack)	0.00/s
		Get (auto ack)	0.00/s

- Node-RED
- Javascript visual programming flow framework for the Internet of Things
- Data received from RabbitMQ and formatted according to the data model
- Flows publish formatted data to API server

The screenshot displays the Node-RED web interface. On the left, there are two panels: 'subflows' and 'input'. The 'subflows' panel contains nodes for 'Create Dataset Entry', 'Create Datablocks', 'Create Archive Job', and 'Create Dataset Lifecycle data'. The 'input' panel contains nodes for 'inject', 'catch', 'status', 'link', 'mqtt', 'http', 'websocket', 'tcp', and 'udp'. The main workspace shows a flow named 'Flow 1' with the following nodes in sequence:

- 'Timeout connection to message broker' (orange node, status: connected)
- 'Turn message into JSON object' (yellow node)
- 'Store foldercontents and prepare xml conversion' (orange node)
- 'Create Dataset Entry' (orange node)
- 'Create Datablocks' (orange node)
- 'Create Dataset Lifecycle data' (orange node)
- 'catch all' (red node)
- 'Log exceptions' (green node)

On the right, the 'debug' console shows the following log output:

```

6/22/2017, 9:24:36 AM node: 7438x200 800x60
context: msg.payload - Object
+ { id: "20.500.11935/1567629b-8c2b-49c.", isOnDisk: true, isOnTape: false, archiveStatusMessage: "Not yet scheduled for archivin.", retrieveStatusMessage: "Never retrieved" ... }

6/22/2017, 9:24:36 AM node: 7438x200 800x60
context: msg.payload - Object
+ { id: "20.500.11935/7177f12a-1252-4a9.", isOnDisk: true, isOnTape: false, archiveStatusMessage: "Not yet scheduled for archivin.", retrieveStatusMessage: "Never retrieved" ... }

6/22/2017, 9:24:36 AM node: 7438x200 800x60
context: msg.payload - Object
+ { id: "20.500.11935/755e163e-8990-43e.", isOnDisk: true, isOnTape: false, archiveStatusMessage: "Not yet scheduled for archivin.", retrieveStatusMessage: "Never retrieved" ... }

6/22/2017, 9:24:36 AM node: 7438x200 800x60
context: msg.payload - Object
+ { id: "20.500.11935/1b34c7c5-9fb2-4a6.", isOnDisk: true, isOnTape: false, archiveStatusMessage: "Not yet scheduled for archivin.", retrieveStatusMessage: "Never retrieved" ... }

```

- Built using Angular and NgRx
- Created with Typescript
- Component based architecture to reuse throughout the application
- Responsive, Standards compliant and all other hip, web development words

Search...

Search

Filter Results

Results: 2052

Archive

View

Retrieve

Beamline

/PSI/SLS/TOMCAT

Group

p16738

Date Range

Clear

Source Folder	Size (GB)	Creation Time	Group	Proposal ID	Archive Status	Retrieve Status
/sls/X02DA/Data10/e16738	8.65	25/07/2017 01:33	p16738	unknown	datasetOnDisk	datasetRetrie
/sls/X02DA/Data10/e16738	8.65	23/07/2017 00:53	p16738	20.500.11935/	datasetOnDisk	datasetRetrie
/sls/X02DA/Data10/e16738	8.65	21/07/2017 12:35	p16738	20.500.11935/	datasetOnArc	datasetRetrie
/sls/X02DA/Data10/e16738	8.65	20/07/2017 20:04	p16738	20.500.11935/	datasetCreate	
/sls/X02DA/Data10/e16738	8.65	21/07/2017 21:37	p16738	20.500.11935/	datasetCreate	
/sls/X02DA/Data10/e16738	8.65	23/07/2017 19:50	p16738	20.500.11935/	datasetCreate	
/sls/X02DA/Data10/e16738	8.65	24/07/2017 03:28	p16738	20.500.11935/	datasetCreate	
/sls/X02DA/Data10/e16738	8.65	21/07/2017 05:23	p16738	20.500.11935/	datasetCreate	
/sls/X02DA/Data10/e16738	8.65	25/07/2017 04:27	p16738	unknown	datasetCreate	
/sls/X02DA/Data10/e16738	8.65	24/07/2017 08:36	p16738	20.500.11935/	datasetCreate	
/sls/X02DA/Data10/e16738	8.65	22/07/2017 03:07	p16738	20.500.11935/	datasetCreate	
/sls/X02DA/Data10/e16738	8.65	22/07/2017 07:52	p16738	20.500.11935/	datasetCreate	

- Kubernetes cluster (5 nodes)
- Each microservice is built into a docker file and saved in a registry
- Kubernetes and Helm package manager deploys the containers and handles:
 - Routing
 - Scaling
 - Server failures
 - Updates

The screenshot shows the Kubernetes Management Dashboard interface. At the top, there is a search bar and a '+ CREATE' button. The main navigation bar shows 'Workloads > Deployments'. On the left, a sidebar lists various Kubernetes resources, with 'Deployments' highlighted under the 'Workloads' section. The main content area displays a table of Deployments with the following columns: Name, Namespace, Labels, Pods, Age, and Images. Each row includes a green checkmark icon on the left and a three-dot menu icon on the right.

Name	Namespace	Labels	Pods	Age	Images
tomcat-ingestor	production	name: tomcat...	1 / 1	8 hours	nodored/node-r...
tomcat-ingestor	qa	name: tomcat...	1 / 1	8 hours	nodored/node-r...
dacat-api	production	name: dacat...	1 / 1	8 hours	registry.psi.ch:5...
dacat-api	qa	name: dacat...	1 / 1	8 hours	registry.psi.ch:5...
melanie-product...	production	app: melanie-... chart: rabbit... heritage: Tiller release: mela...	1 / 1	8 hours	bitnami/rabbitm...
melanie-qa-rabb...	qa	app: melanie-... chart: rabbit... heritage: Tiller release: mela...	1 / 1	8 hours	bitnami/rabbitm...
kubernetes-das...	kube-system	k8s-app: kube...	1 / 1	9 hours	gcr.io/google_e...
nginx-ingress-co...	kube-system	k8s-app: ngin...	1 / 1	6 days	gcr.io/google_e...
default-http-bac...	kube-system	k8s-app: defa...	1 / 1	a month	gcr.io/google_e...
tiller-deploy	kube-system	app: helm name: tiller	1 / 1	a month	gcr.io/kubemete...

- Each beamline makes a cURL request to a predefined endpoint (language agnostic)
- There is one separate (node-red) container for each beamline, to maximize independence
- There is one "Virtual host" for the message broker for each facility

- 1 Data captured at beamline and cURL command publishes data (in any format) to RabbitMQ
- 2 Node-RED flow consumes data from the queue and formats according to RawDataset model
- 3 API server receives call and saves to MongoDB
- 4 Frontend website queries for datasets and presents to the logged in user with access
- 5 User chooses to archive dataset and command is sent to the API server
- 6 API server submits job to the archive system. Archive system updates the DatasetLifecycle

- 1 Introduction
- 2 Data Catalog Purpose
- 3 Data Model
- 4 Architecture Goals and Rationale
- 5 Architecture Overview
- 6 How do we run it?**
- 7 Development Process
- 8 Collaboration

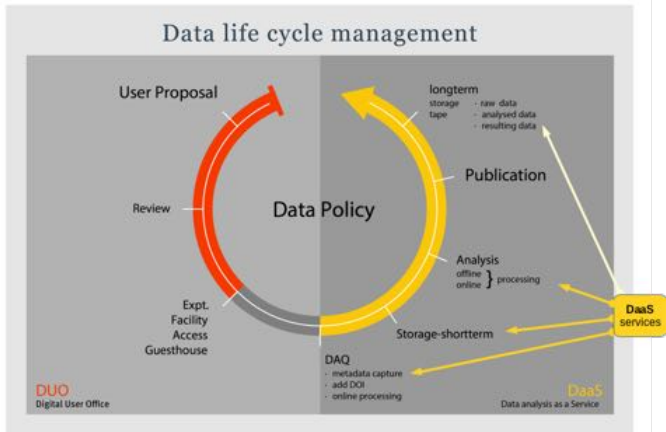
- We use a Kubernetes Cluster as the basis for the deployment
- Do you have a Kubernetes cluster?
 - YES! Clone it and run it!
 - NO! Then grab a spare computer with a load of computing power
 - localdeploy is a prebuilt tool to create a Kubernetes cluster on your machine and deploy the basic services

- 1 Introduction
- 2 Data Catalog Purpose
- 3 Data Model
- 4 Architecture Goals and Rationale
- 5 Architecture Overview
- 6 How do we run it?
- 7 Development Process**
- 8 Collaboration

- We apply the "lazyness principle": avoid own coding wherever possible (e.g. use configuration driven development)
- Development and QA version for testing, Production version rolled out to beamline scientists
- Agile development with ESS and Max IV
- Regular meetings with scientists and requirements gathering. Issues created in repo.

- 1 Introduction
- 2 Data Catalog Purpose
- 3 Data Model
- 4 Architecture Goals and Rationale
- 5 Architecture Overview
- 6 How do we run it?
- 7 Development Process
- 8 Collaboration**

- SciCat Project - <https://github.com/scicatproject>
- Dedicated development started only about 6 months ago and we would like to see if our development can provide mutual benefits
- We want your feedback and experiences, as well as to identify areas with similar goals
- Standard Github procedure - collaborate with merge requests
- PSI is very open to suggestions for improvements and welcomes concrete contributions, ideally in form of reusable components
- We prefer to actually have a common code base, not a "one-time fork-and-forget" situation



Data policy defines (long term) goals concerning data storage, life cycle management, data access and ownership

- Implementation of data policy needs a central data catalog

/home/amor/data/2015/000/amor2015e000008.hdf	2015-04-28 10:37:27	113820	<input type="checkbox"/> sign <input type="checkbox"/> archive <input type="checkbox"/> sign	<input type="checkbox"/> encrypt <input type="checkbox"/> 2 tapes <input type="checkbox"/> encrypt
--	---------------------	--------	--	--

Dataset 579da05e0024ef0bsb885425

[Basic data](#)
[Sample/Probe](#)
[Beam/Source](#)
[Detector](#)
[Scan](#)
[Images](#)

Attach Files

