Machine Learning Data Platform

Overview of the Machine Learning Data Platform (MLDP) with focus on the Data Platform (DP) subsystem

Christopher K. Allen
Craig McChesney

Topics

- ***MLDP Overview - "The Big Picture"***
- Survey of MLDP Elements
- Project Status and Road Map
- Summary

MLDP Overview
“The Big Picture”

- Motivation
- Concept
- Subsystems

MLDP Overview
Motivation

Full-stack support for machine learning and general data science applications for the diagnosis, modeling, control, and optimization of large particle accelerator and experimental physics facilities.

- From high-speed data acquisition to rapid ML/AI application development and deployment.
- A standardized platform for rapid implementation and deployment of ML/Data Science algorithms to different operating configurations and different facilities.
- Data science perspective of archived facility data

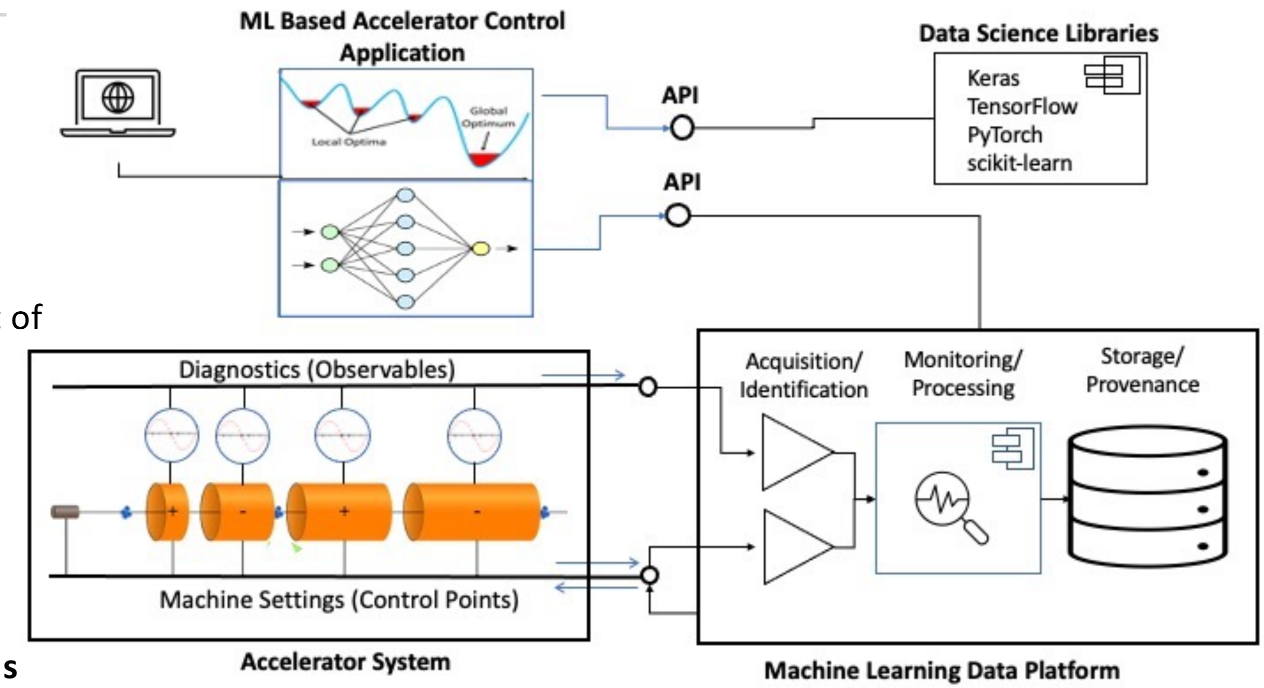
MLDP Overview

Concept

The **Machine Learning Data Platform** (MLDP) has 3 primary functions:

1. High-speed data acquisition (EPICS).
2. Archiving, processing, and management of heterogeneous, time-correlated data.
3. Data Analysis: Broad query, annotation, and processing of archive.
   (data science/ML/AI applications)

- Functions are realized by **separate subsystems** each supporting a category of use cases.



Conceptual diagram of Machine Learning Data Platform

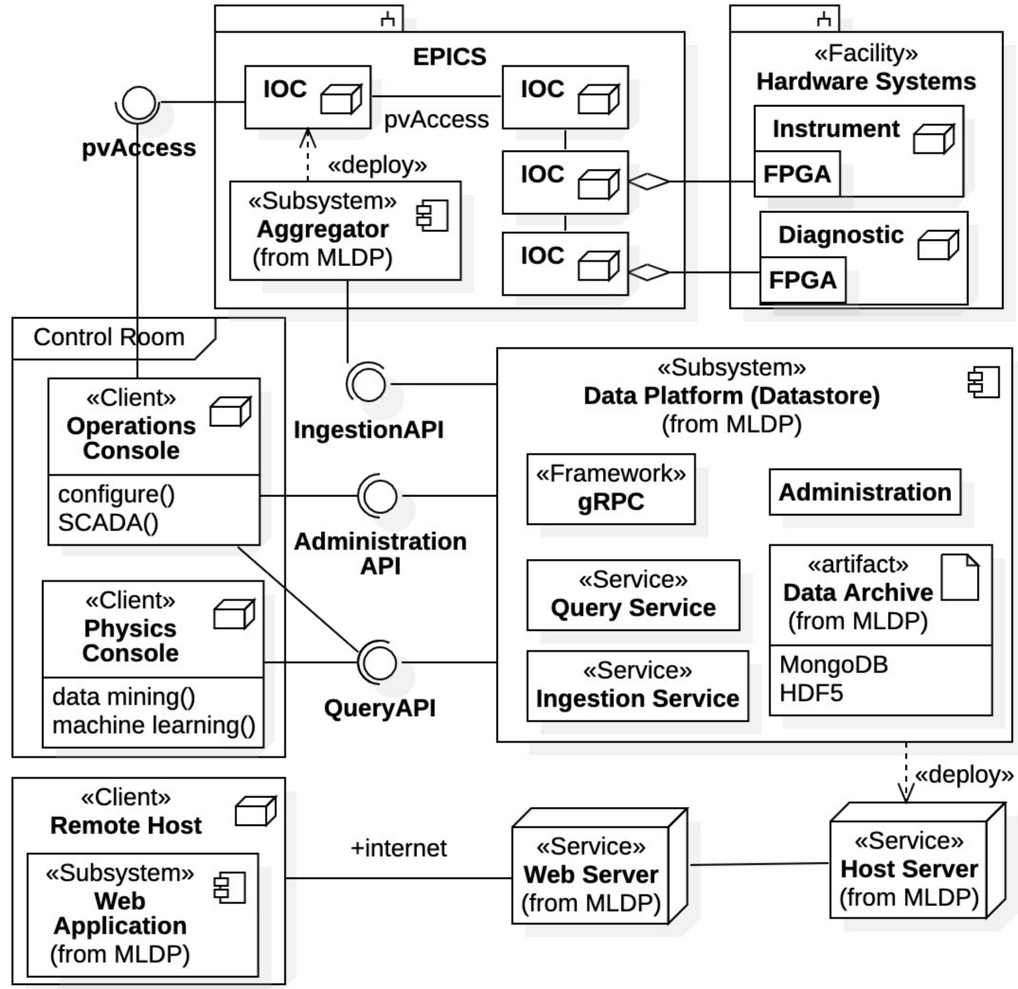
MLDP Overview

Subsystems

1. **Aggregator** (a *Data Provider*)
   - High-speed data acquisition and collection of facility hardware data.
   - Deployed with EPICS control system.

2. **Data Platform** (previously **Datastore**)
   - Standalone – deployed on separate server(s), EPICS not required.
     - Any facility may use data platform.
   - Contains the Data Archive.
   - Composed of collaborating services.
   - Independent gRPC comm. framework.
   - Well-defined APIs for communication.

3. **Web Application** provides remote access to Data Archive using an internet web browser.
   - Attaches to the Data Platform subsystem.



Machine Learning Data Platform subsystems and deployment

Topics

- Overview of the MLDP - “The Big Picture”
- ***Survey of MLDP Elements***
- Project Status and Road Map
- Summary

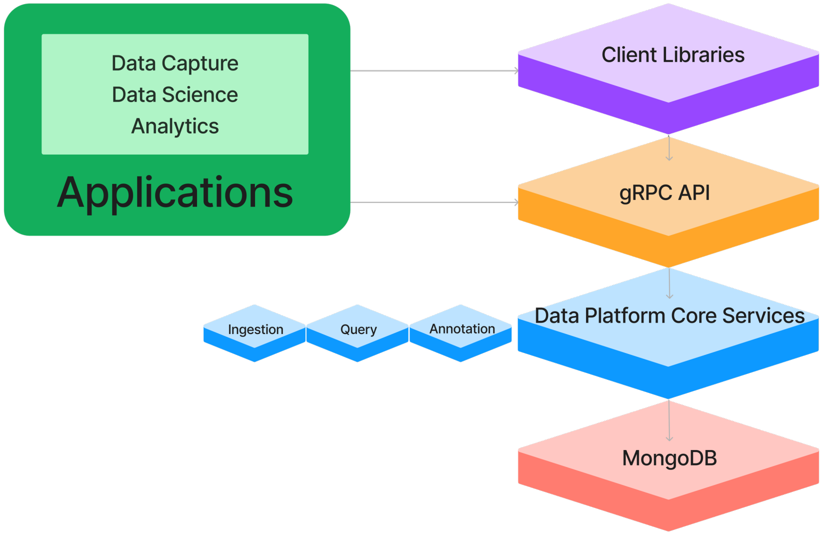
Survey of MLDP Elements

- ***Tech Stack***
- Aggregator / Data Provider
- Data Archive
- gRPC API
- Data Platform / Core Services
- Client Libraries
- Web Application

MLDP Elements

Tech Stack

- The Data Platform Core Services are implemented as Java server applications.
- The MongoDB document-oriented database management system is used by the services for persistence.
- The Data Platform API is built upon the gRPC open-source high-performance communication framework.
- Java and Python Client Libraries are provided for higher-level interaction hiding the gRPC API details.
- Applications communicate either directly via the gRPC API or using the Client Libraries.



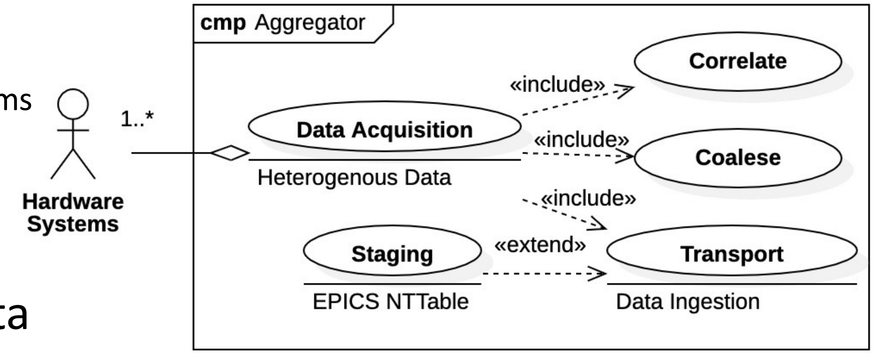
Survey of MLDP Elements

- Tech Stack
- ***Aggregator / Data Provider***
- Data Archive
- gRPC API
- Data Platform / Core Services
- Client Libraries
- Web Application

MLDP Elements

Aggregator / Data Provider

Clients here are hardware systems



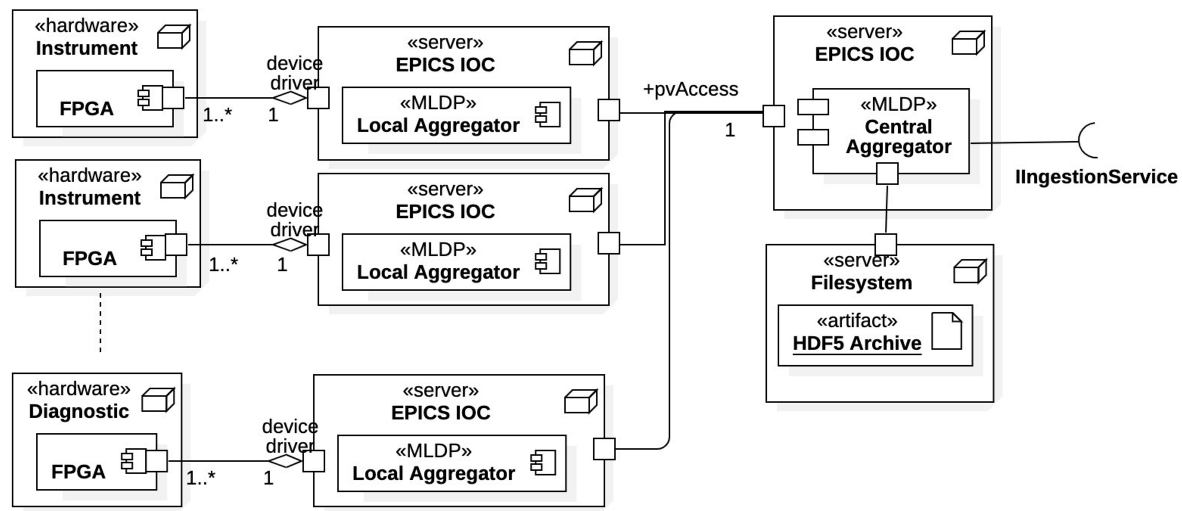
Aggregator use cases

Aggregator performs synchronous, high-speed data acquisition and collection within EPICS control system.

Distributed system

- Local Aggregators – proximal to hardware
  - Collect and align local hetero data.
  - May have multiple data sources.
  - Transport to Central Aggregator.

- Central Aggregator
  - Coalesce all aggregated data.
  - Stage as NTTable "snapshots".
  - Transport to Data Platform via API.



Aggregator system architecture

Survey of MLDP Elements

- Tech Stack
- Aggregator / Data Provider
- ***Data Archive***
- gRPC API
- Data Platform / Core Services
- Client Libraries
- Web Application

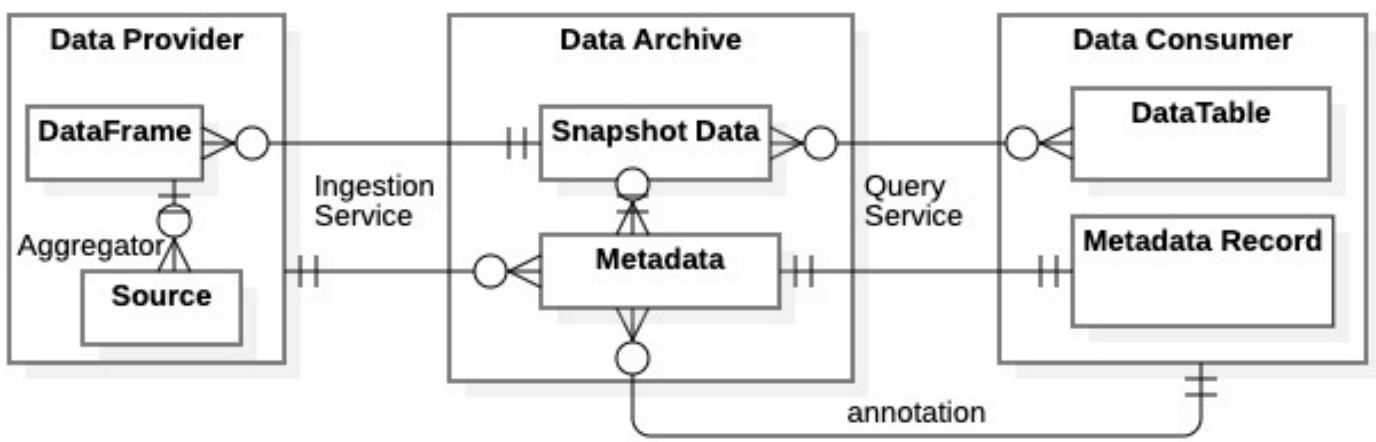
MLDP Elements

Data Archive

NOTE: The MLDP data archive is maintained completely by the Data Platform subsystem.

Data Archive contains

- PV Time-Series Data – Heterogenous, correlated, time-series device data.
  - Flows in as Data Frames (NTTables) .
  - Flows out as dynamic Data Tables.
- Metadata – Characteristics/associations/properties within snapshot data.
  - Created by Ingestion Service and augmented by Query Service.
  - Flows out as Metadata Records.
- Annotations – User additions to data archive (notes, relations, calculations).
  - Created by Data Consumers as "value-added" data.
  - Flows out as Annotation Records.



MLDP data flow diagram

Survey of MLDP Elements

- Tech Stack
- Aggregator / Data Provider
- Data Archive
- ***gRPC API***
- Data Platform / Core Services
- Client Libraries
- Web Application

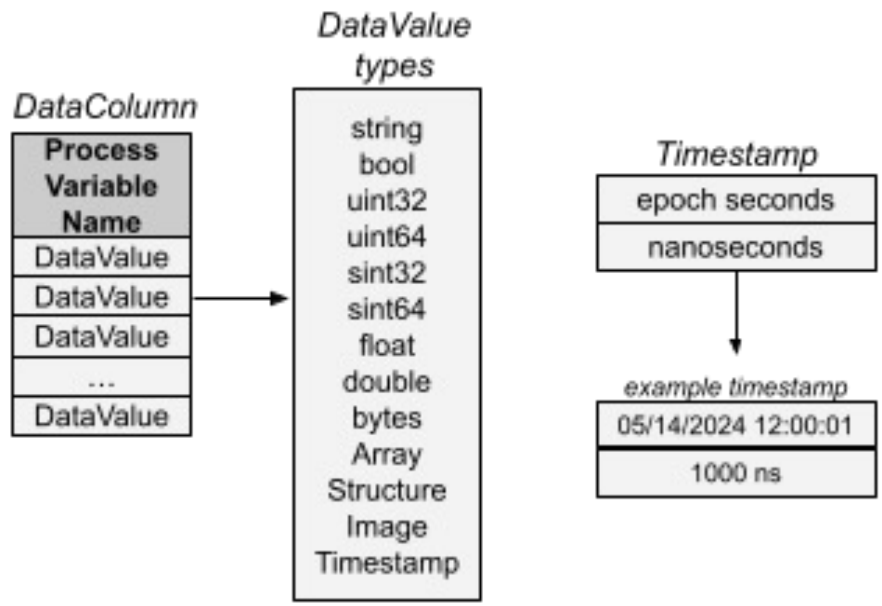
MLDP Elements
gRPC API

- The gRPC communication framework was originally developed by Google for use in connecting microservices.
- It uses HTTP/2 for transport, and Protocol Buffers as both the interface definition language and message interchange format.
- Supports simple unary single request / response APIs as well as unidirectional and bidirectional streaming.
- We chose gRPC for the Data Platform API because it can meet our performance requirements for data ingestion, provides bindings for virtually any programming language, and supports a variety of application styles.

MLDP Elements

gRPC API: Heterogeneous Data

- The API defines type “DataValue” to represent a range of heterogeneous data types for use in the ingestion and query interfaces ranging from simple scalar data types to complex types including multi-dimensional arrays, structures, and images.
- Each DataValue can optionally include “ValueStatus” information captured from the control system.
- Times are represented using components for epoch second and nanoseconds (a third component could be added for finer resolution).



Survey of MLDP Elements

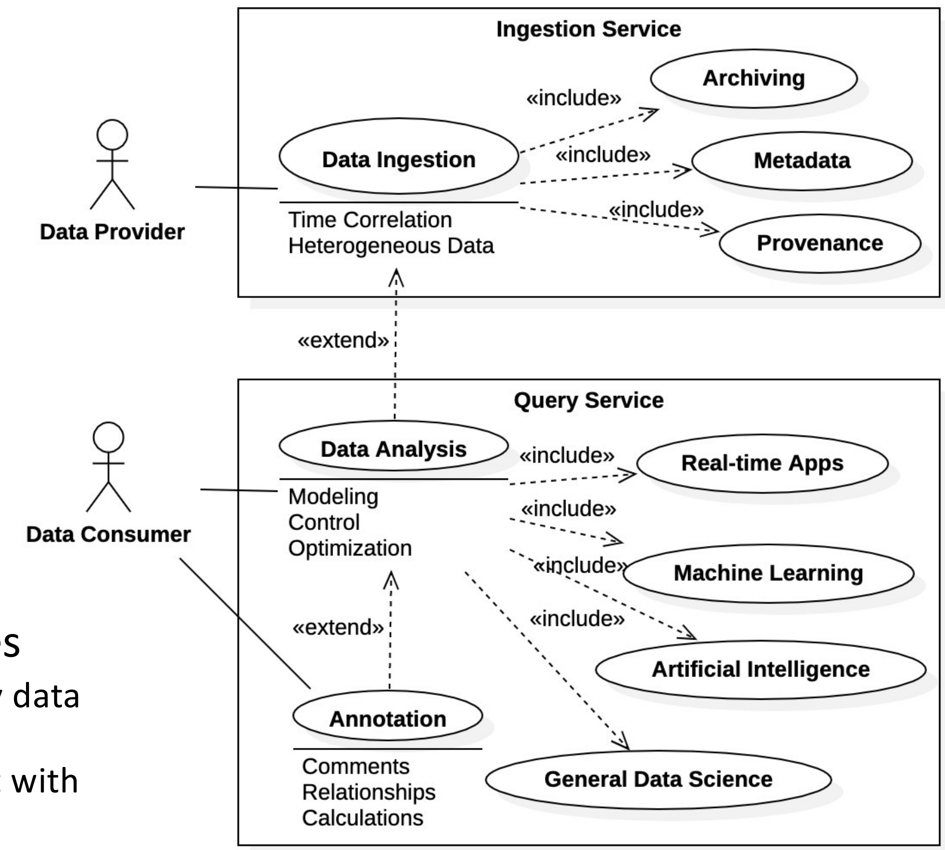
- Tech Stack
- Aggregator / Data Provider
- Data Archive
- gRPC API
- ***Data Platform / Core Services***
  - ***Service Model***
  - ***Ingestion Service***
  - ***Query Service***
  - ***Annotation Service***
- Client Libraries
- Web Application

MLDP Elements

Data Platform- Service Model

Clients are generically divided into
Data Providers - populate data archive
Data Consumers - utilize data archive

- The Data Platform Core Services manage the Data Archive and are the primary interaction point for clients.
- The Data Platform is a **standalone system, i**ndependent of EPICS.
- Fundamental components implemented as services
  - **Ingestion Service** – clients are Data Providers that supply data to the Archive.
  - **Query Service** – clients are Data Consumers that interact with the Data Archive.
    - Engineers, data scientists, physicists, applications, remote users, etc.
  - **Annotation Service** – clients are Data Consumers that supply ”value added” information to the Data Archive.



Data Platform use cases as seen by clients

MLDP Elements

Data Platform / Core Services

Ingestion Service

- The Ingestion Service provides APIs for provider registration, data ingestion, and querying status of ingestion requests.
- Processing is asynchronous to maximize performance.
- Writes time-series data "buckets" to MongoDB, each containing vector of data values for a single PV over a time range.
- Bucket stores data values using serialized Protobuf format, and specifies timestamps for data values using start time, sample period, number of samples (or explicit list of timestamps).

```
rpc registerProvider (RegisterProviderRequest) returns (RegisterProviderResponse);
rpc ingestDataStream (stream IngestDataRequest) returns (IngestDataStreamResponse);
rpc queryRequestStatus(QueryRequestStatusRequest) returns (QueryRequestStatusResponse);
```

MLDP Elements

Data Platform / Core Services

Query Service

- The Query Service provides APIs for retrieving PV time-series data in bucketed or tabular format, and for querying metadata about data sources.
- Data query response contains data buckets matching query's PV and time range criteria.
- Client Library provides tools for assembling bucketed data from response to a tabular structure, if appropriate.

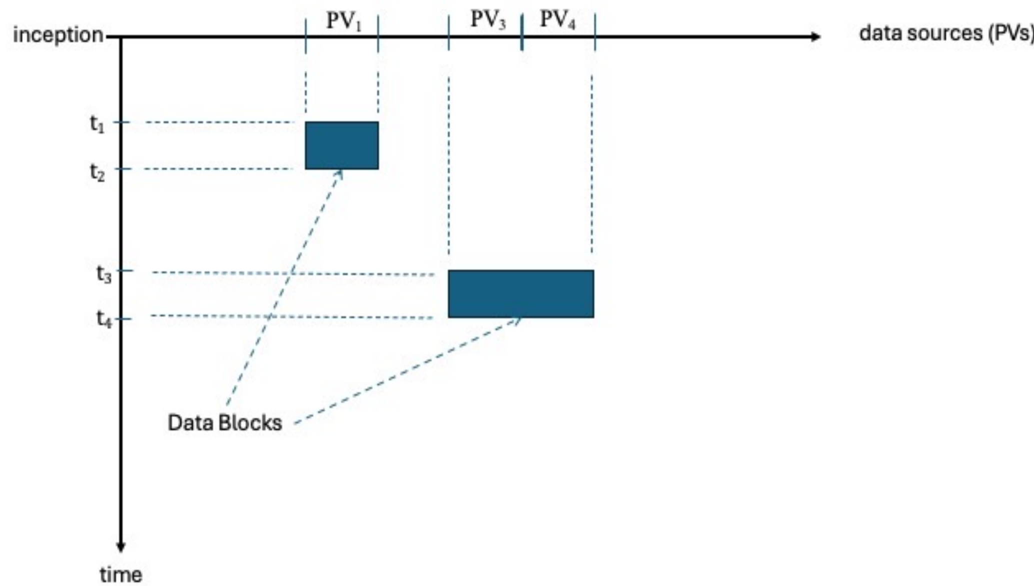
```
rpc queryDataStream(QueryDataRequest) returns (stream QueryDataResponse);
rpc queryTable(QueryTableRequest) returns (QueryTableResponse);
rpc queryMetadata(QueryMetadataRequest) returns (QueryMetadataResponse);
```

MLDP Elements

Data Platform / Core Services

Annotation Service

- A Dataset is comprised of Data Blocks, each specifying PV(s) and time range.
- The Annotation Service provides APIs for identifying "Datasets" in the archive, adding annotations to them, and performing queries.
- A number of different types of annotations are supported (or planned), including basic descriptive information, links between related datasets, calculations, derived datasets, and provenance.




```
rpc createDataSet(CreateDataSetRequest) returns (CreateDataSetResponse);
rpc queryDataSets(QueryDataSetsRequest) returns (QueryDataSetsResponse);
rpc createAnnotation(CreateAnnotationRequest) returns (CreateAnnotationResponse);
rpc queryAnnotations(QueryAnnotationsRequest) returns (QueryAnnotationsResponse);
```

Survey of MLDP Elements

- Tech Stack
- Aggregator / Data Provider
- Data Archive
- gRPC API
- Data Platform / Core Services
- ***Client Libraries***
- Web Application

MLDP Elements

Client Libraries

Libraries are also available for building Data Platform clients (under development).

- Client libraries avoid the complexities of direct gRPC communications.
- Offer additional features, such as default configuration, buffering, dynamic data tables, etc.

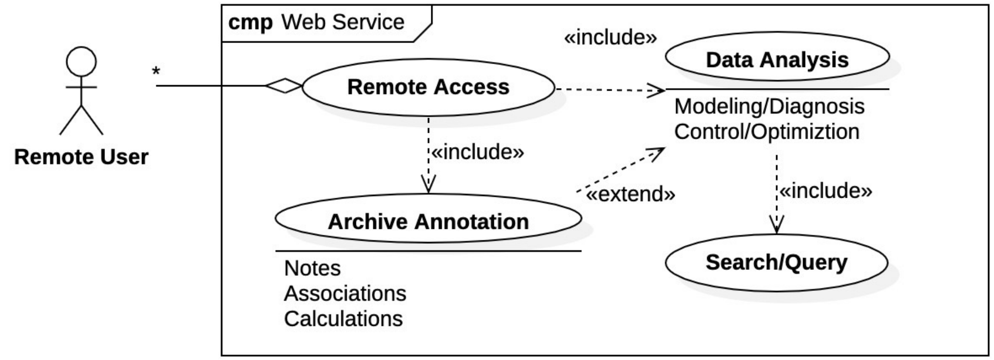
- Currently migrating high-level Java API libraries to new platform architecture.
  - An additional Python query API is planned.
    - Based upon Pandas library common to most data science.
  - An EPICS pvAccess interface to the Ingestion Service is also anticipated.

Survey of MLDP Elements

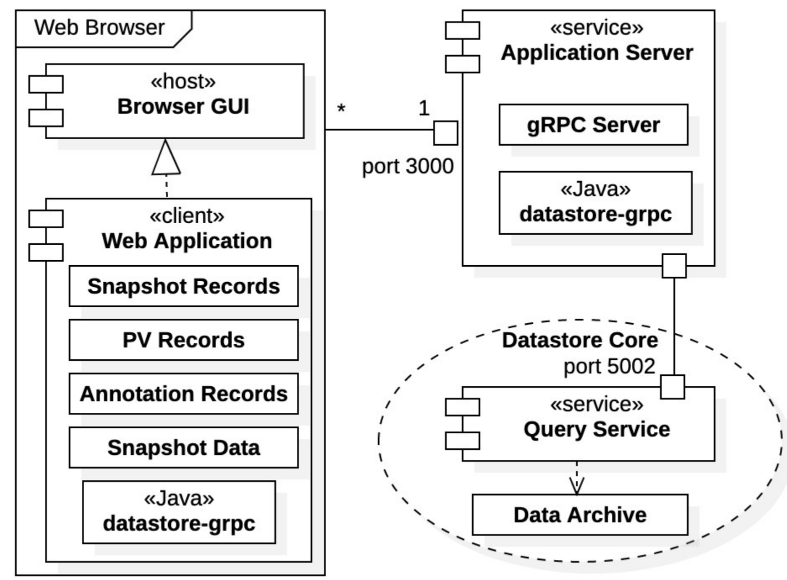
- Tech Stack
- Aggregator / Data Provider
- Data Archive
- gRPC API
- Data Platform / Core Services
- Client Libraries
- ***Web Application***

MLDP Elements
Web Application

- Web Application is a browser-based application built using JavaScript / React / Node.
- Web Application provides remote access to the DP data archive.
  - Facilitates a subset of Query and Annotation Service use cases.
  - Tools for inspection, visualization, downloads, and data analysis.
- JavaScript Node server provides API that augments Data Platform API.
- Envoy proxy translates http traffic from browser application to http2 for consumption by gRPC servers.



Web Application Use Cases



Web Application Architecture

Topics

- Overview of the MLDP - “The Big Picture”
- Survey of MLDP Elements
- ***Project Status and Road Map***
- Summary

MLDP Project Status and Road Map

- Completed - Redesign of Data Platform Archive and Core Services
  - Eliminated many 3rd party dependencies from prototype – now restricted to Java, gRPC, and MongoDB.
  - Redesigned prototype gRPC API.
  - Evaluated C++ gRPC for datastream processing (~ 500 MBps transmission rates).
  - Created deployment tools.
  - Implemented core service API handling for Ingestion, Query, and Annotation Services.
  - Ingestion over 200x faster than prototype at ~ 200 MBps data ingestion rates.

- Roadmap
  - Client Library development (Java and Python)
  - Export mechanism
  - More extensive load and scale testing
  - Additional annotation types
  - Data generator / simulator
  - Support for Real-time consumer use cases via inline ingestion data stream processing / algorithm
  - Framework for measuring data ingestion statistics from capture to archival
  - Token based authentication for query clients and web application
  - Age-based archival of data to reduce database footprint
  - Tools for replication / synchronization of data and index to distributed researchers
  - Investigate MongoDB clustering, sharding, and connection pooling
  - Investigate horizontal scaling for service deployment

Topics

- Overview of the MLDP - "The Big Picture"
- Survey of MLDP Elements
- Project Status and Road Map
- ***Summary***

Summary

- The Data Platform is a standalone subsystem of the larger Machine Learning Data Platform (MLDP).
  - Management of heterogenous data with focus on data science.
  - A working prototype was initially developed – “Datastore”.
  - A new, performance-based version is currently under development – “Data Platform”.
  - Data Platform has multiple components supporting use cases as collaborating services (with well-defined APIs).
- Data Platform has 2 communication methods.
  - Direct gRPC communications
  - Client API libraries for Java and Python (under development).
- A deployment system is available for latest DP releases.
  - Ingestion, Query, and Annotation Services are operational.
  - Only direct gRPC communication is currently available.

Supplemental

Supplemental Material

- ***SBIR Project Background***
- Ingestion Data Flow Diagram
- Bucket Pattern for Time-Series Data
- DP Deployment
- Example Data Flow
- Why use an API?
- Authentication / Authorization
- Performance Benchmarks
- Application Layer: Low-Level API vs. High-Level Library

MLDP Overview

SBIR Project Background

MLDP development is supported by the US Dept. of Energy (DOE) under a Small Business Innovative Research (SBIR) grant starting in 2022.

- A prototype MLDP was completed in Phase I with the following subsystems:
  - Aggregator – EPICS based, high-speed synchronous data acquisition of heterogeneous data.
  - **Datastore** – Standalone system for data ingestion, archive management and access.
  - Web Application – Universal, remote access and interaction with data archive.

- SBIR Phase II awarded in 2023 (Fiscal Year 2024)
  - Redesign of Datastore archive and services with emphasis on performance (Year 1).
  - Support for full archive annotation (Year 1).
  - **Datastore** use case expansions (Year 2)
    - Datastream processing,
    - Algorithm Plugins,
    - Advanced Data Science Applications.

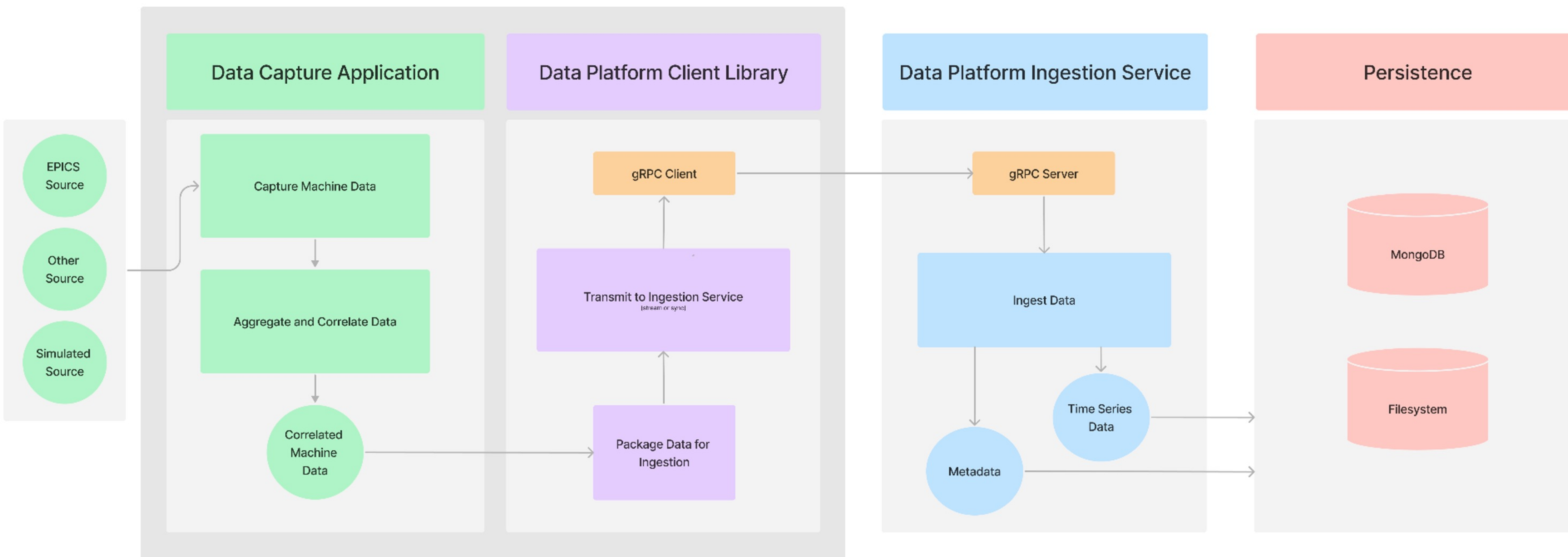
- "**Data Platform**" (DP) references previous "**Datastore**" system of prototype MLDP.
  - Includes upgrades and extensions for Phase II project.

Supplemental Material

- SBIR Project Background
- ***Ingestion Data Flow Diagram***
- Bucket Pattern for Time-Series Data
- DP Deployment
- Example Data Flow
- Why use an API?
- Authentication / Authorization
- Performance Benchmarks
- Application Layer: Low-Level API vs. High-Level Library

MLDP Elements

Ingestion Data Flow Diagram



Supplemental Material

- SBIR Project Background
- Ingestion Data Flow Diagram
- ***Bucket Pattern for Time-Series Data***
- DP Deployment
- Example Data Flow
- Why use an API?
- Authentication / Authorization
- Performance Benchmarks
- Application Layer: Low-Level API vs. High-Level Library

MLDP Elements

Data Platform / Core Services

Ingestion Service: Bucket Pattern for Time-Series Data

- Simple example with 3 measurements, each stored as an individual database record:

```
{ sensor_id: 12345, timestamp: ISODate("2019-01-31T10:00:00.000Z"), temperature: 40 }
{ sensor_id: 12345, timestamp: ISODate("2019-01-31T10:01:00.000Z"), temperature: 40 }
{ sensor_id: 12345, timestamp: ISODate("2019-01-31T10:02:00.000Z"), temperature: 41 }
```

- Same 3 measurements stored in a single data bucket record, saving the overhead of sensor_id and timestamp for each measurement:

```
{ sensor_id: 12345,
  start_date: ISODate("2019-01-31T10:00:00.000Z"),
  sample_period_nanos: 1_000_000_000,
  count: 3
  measurements: [ 40, 40, 41 ] }
```

Supplemental Material

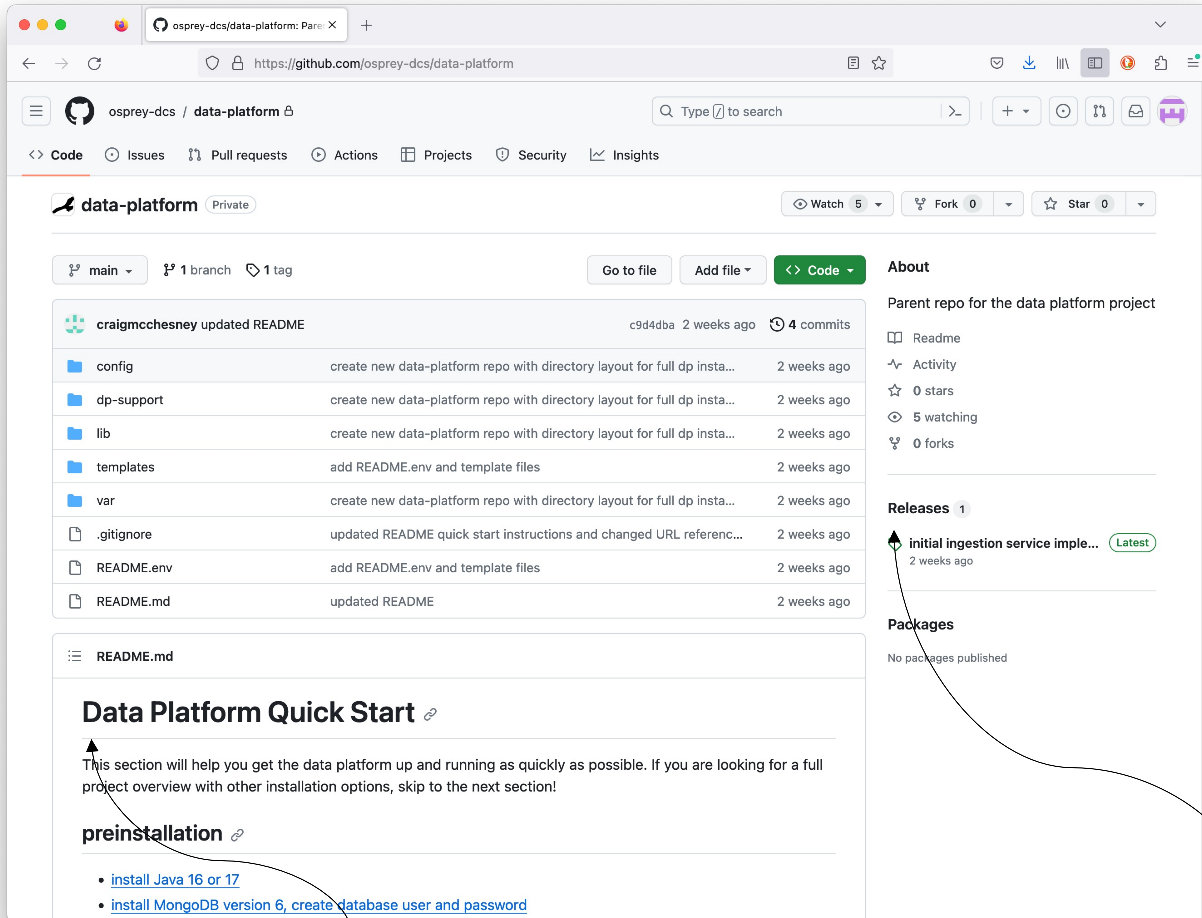
- SBIR Project Background
- Ingestion Data Flow Diagram
- Bucket Pattern for Time-Series Data
- ***DP Deployment***
- Example Data Flow
- Why use an API?
- Authentication / Authorization
- Performance Benchmarks
- Application Layer: Low-Level API vs. High-Level Library

DP Deployment

Installation Repository

https://github.com/osprey-dcs/data-platform

- Installation distributed as a zipped archive.
  - Download and unzip into local installation directory.
- Services deployed as Java "fat jars".
- Services are started with scripts.
  - located in expanded `dp-support` directory.
- Process is well documented.
  - See Github repository page.



Instructions/documentation

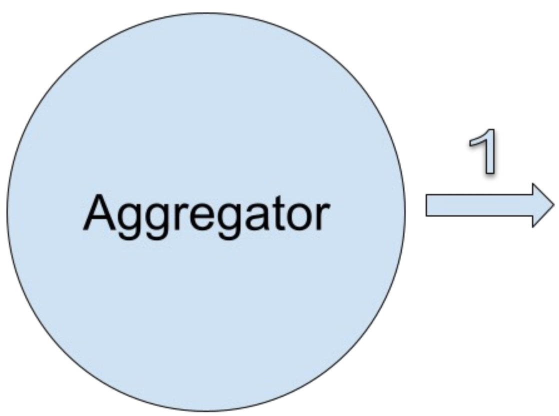
data-platform-installer.tar.gz - Download and unzip.

Supplemental Material

- SBIR Project Background
- Ingestion Data Flow Diagram
- Bucket Pattern for Time-Series Data
- DP Deployment
- ***Example Data Flow***
  - ***Aggregator***
  - ***Provider***
  - ***Ingestion Service***
  - ***Query Service***
  - ***Annotation Service***
- Why use an API?
- Authentication / Authorization
- Performance Benchmarks
- Application Layer: Low-Level API vs. High-Level Library

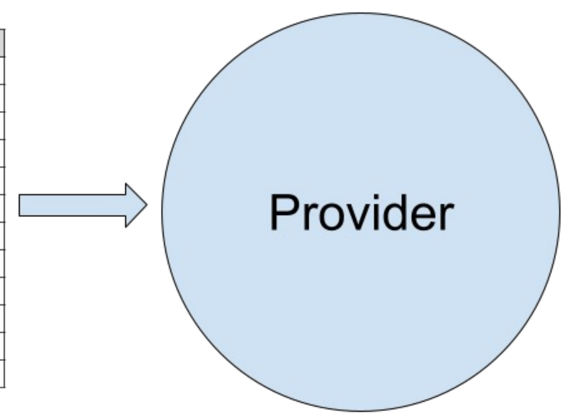
Example Data Flow

Aggregator -> Provider



Correlated PV Time-Series Data

timestamp	PV1	PV2	PV3
12:00:00.000	0.1	[false, false, false, false]	image1
12:00:00.250	0.2	[false, false, false, true]	image2
12:00:00.500	0.3	[false, false, true, false]	image3
12:00:00.750	0.4	[false, false, true, true]	image4
12:00:01.000	1.1	[false, true, false, false]	image5
12:00:01.250	1.2	[false, true, false, true]	image6
12:00:01.500	1.3	[false, true, true, false]	image7
12:00:01.750	1.4	[false, true, true, true]	image8
12:00:02.000	2.1	[true, false, false, false]	image9
12:00:02.250	2.2	[true, false, false, true]	image10
12:00:02.500	2.3	[true, false, true, false]	image11
12:00:02.750	2.4	[true, false, true, true]	image12

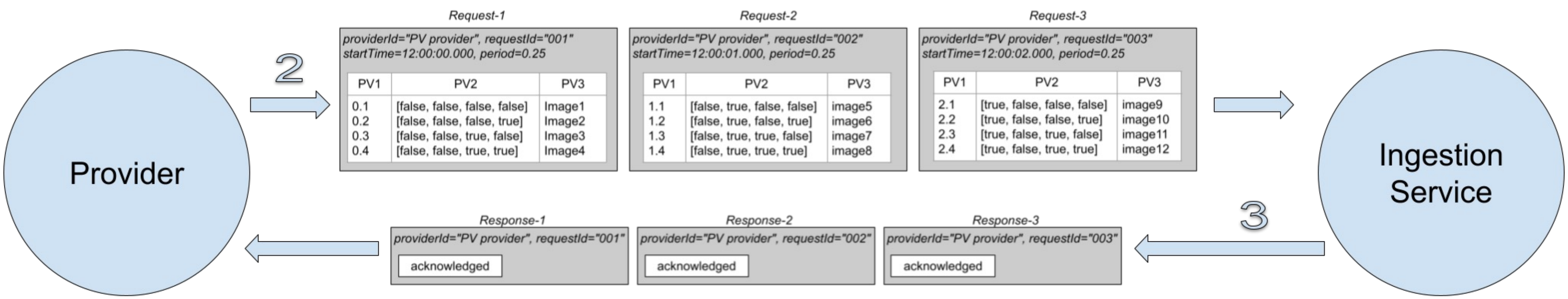


1

External Aggregator app prepares correlated PV time-series data from control systems infrastructure.

Example Data Flow

Provider -> Ingestion Service



2

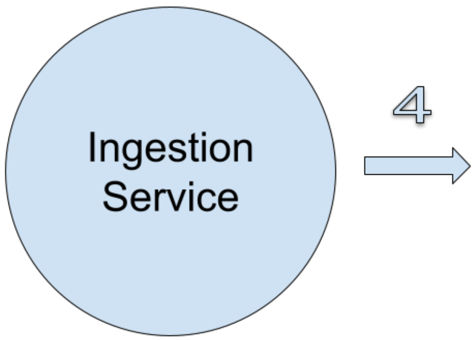
External Provider app invokes Ingestion Service's ingestDataStream() RPC method, sending a stream of ingestion requests each containing a set of PV data vectors for the specified start time.

3

Ingestion Service validates incoming requests, and immediately replies in the API response stream with acknowledgment or rejection for each request. Service handles requests asynchronously.

Example Data Flow

Ingestion Service -> MongoDB



4

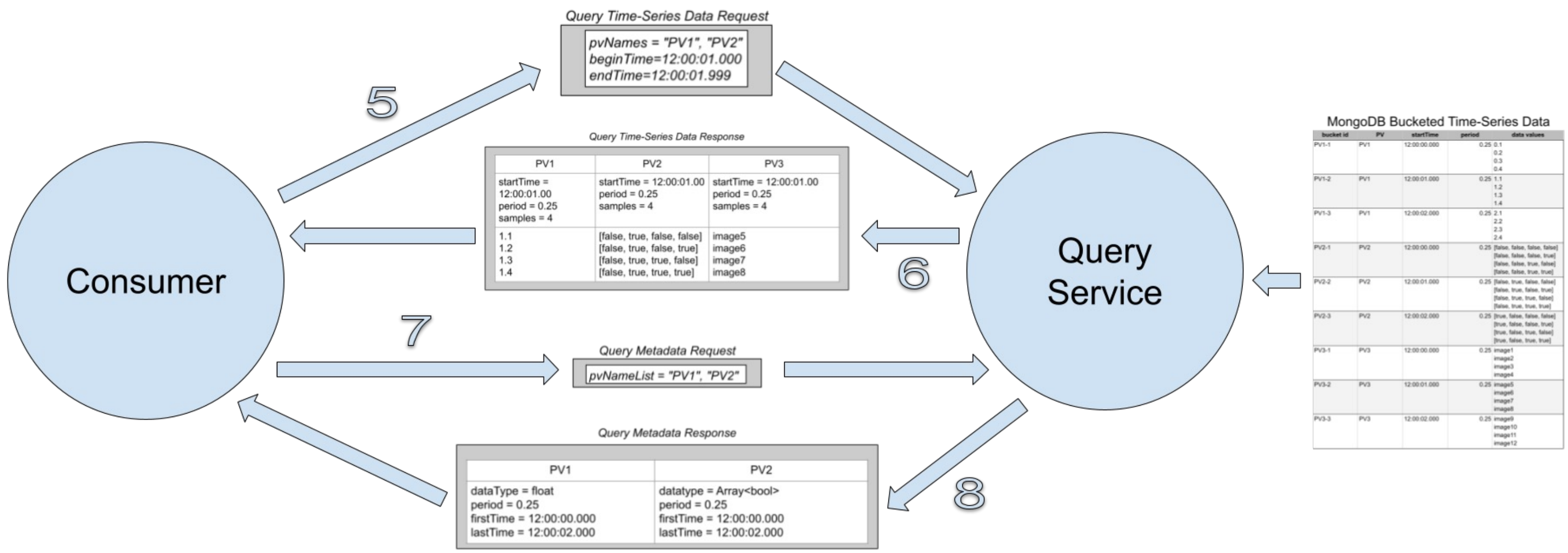
Ingestion Services processes queue of requests, writing bucketed time-series data to MongoDB. Each bucket contains a list of data values for the specified PV and time range.

MongoDB Bucketed Time-Series Data

bucket id	PV	startTime	period	data values
PV1-1	PV1	12:00:00.000	0.25	0.1 0.2 0.3 0.4
PV1-2	PV1	12:00:01.000	0.25	1.1 1.2 1.3 1.4
PV1-3	PV1	12:00:02.000	0.25	2.1 2.2 2.3 2.4
PV2-1	PV2	12:00:00.000	0.25	[false, false, false, false] [false, false, false, true] [false, false, true, false] [false, false, true, true]
PV2-2	PV2	12:00:01.000	0.25	[false, true, false, false] [false, true, false, true] [false, true, true, false] [false, true, true, true]
PV2-3	PV2	12:00:02.000	0.25	[true, false, false, false] [true, false, false, true] [true, false, true, false] [true, false, true, true]
PV3-1	PV3	12:00:00.000	0.25	image1 image2 image3 image4
PV3-2	PV3	12:00:01.000	0.25	image5 image6 image7 image8
PV3-3	PV3	12:00:02.000	0.25	image9 image10 image11 image12

Example Data Flow

Consumer -> Query Service



5 Client sends time-series data query request via queryData() RPC method to retrieve measurements for specified list of PVs and time range. Additional methods offer streaming and tabular formatted results.

6 Query Service responds with a set of data buckets matching query parameters, each including PV name, startTime, sample period, number of samples, and list of PV data values.

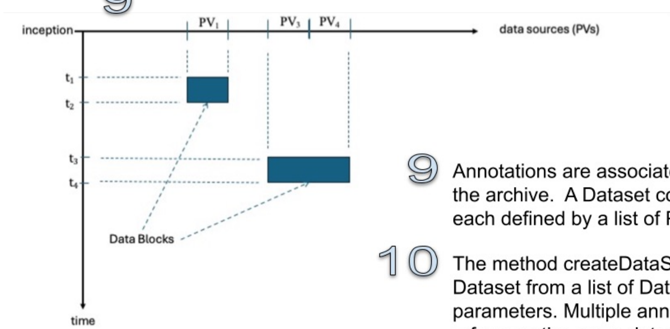
7 Client sends metadata query request via queryMetadata() RPC method to learn about data sources available in the archive, specifying list of PV names or regex pattern.

8 Query Service responds with metadata for each PV matching the query parameters, including data type, sample period, and first and last sample times for each.

Example Data Flow

Consumer -> Annotation Service

9



9 Annotations are associated with a Dataset in the archive. A Dataset contains Data Blocks, each defined by a list of PVs and time range.

10 The method createDataSet() creates a Dataset from a list of Data Block parameters. Multiple annotations may reference the same datasetId.

11 The method createAnnotation() associates an annotation with a Dataset. A variety of annotation types is supported including comments, calculations, and links to other datasets.

12 The method queryAnnotations() accepts a list of search criteria, corresponding to the different types of supported annotations.

13 queryAnnotations() returns the list of annotations matching the search criteria. Note that the annotation data blocks correspond directly to the domain of the Query Service's time-series data query methods.

Dataset

datasetId=1, description="sector 1 power fluctuation"

Data Blocks

pvNames	beginTime	endTime
PV1, PV3	12:00:00.000	12:00:00.999
PV1, PV3	12:00:02.000	12:00:02.000

10

Comment Annotation for Dataset

datasetId=1, ownerId="craigmcc"

Comment

"There was a power fluctuation in Sector 1. Bad things happened.

11

Annotations Query Request

Criteria

owner = "craigmcc"
commentText = "Sector 1"

12

Annotations Query Result

Annotations

id	owner	data blocks
1	craigmcc	PV1, PV3: 12:00:00.000 - 12:00:00.999 PV1, PV3: 12:00:02.000 - 12:00:02.999

13

Supplemental Material

- SBIR Project Background
- Ingestion Data Flow Diagram
- Bucket Pattern for Time-Series Data
- DP Deployment
- Example Data Flow
- ***Why use an API?***
- Authentication / Authorization
- Performance Benchmarks
- Application Layer: Low-Level API vs. High-Level Library

Why use an ingestion service API instead of writing directly to the database?

This is a common question and is understandable when the focus is on ingestion performance. Using a service API facilitates:

- Changing the underlying persistence technology or database schema transparently to the clients

- Capturing data from a wide variety of devices without creating a custom capture client for each of them that exposes the details of the underlying persistence mechanism and must be kept in sync with database/schema changes

The performance measured for the initial ingestion service implementation is in the same range as our benchmark for writing data directly to MongoDB, so using the service API doesn't seem to degrade performance significantly.

Supplemental Material

- SBIR Project Background
- Ingestion Data Flow Diagram
- Bucket Pattern for Time-Series Data
- DP Deployment
- Example Data Flow
- Why use an API?
- ***Authentication / Authorization***
- Performance Benchmarks
- Application Layer: Low-Level API vs. High-Level Library

Authentication / Authorization

Assumption is that ingestion clients run behind firewall and that authentication is not required (e.g., similar to EPICS infrastructure components). Authentication is required for query clients. Initial thoughts about how we will handle authentication and authorization in the gRPC service:

- Enable server authentication and secure transport via built in gRPC support for SSL/TLS.
- Create a new "login" gRPC API method in the ingestion (and query) services. The login method will use the infrastructure LDAP service to authenticate the user credentials. It will return a JSON web token (JWT) to the caller.
- In subsequent gRPC API calls, the client will attach the JWT token as metadata to the request header.
- Investigate use of API token authentication for infrastructure applications (not tied to a specific user).

Supplemental Material

- SBIR Project Background
- Ingestion Data Flow Diagram
- Bucket Pattern for Time-Series Data
- DP Deployment
- Example Data Flow
- Why use an API?
- Authentication / Authorization
- ***Performance Benchmarks***
- Application Layer: Low-Level API vs. High-Level Library

NOTE: These measurements were obtained with v1.1 and need to be re-measured using v1.5+

Project Status

Performance Benchmarks

Redesign of Data Platform required extensive testing and benchmarking of existing technologies and methods.

- Performance
- Modularity/dependencies
- Development effort
- Ease of deployment

Final design decisions for Data Platform

- Java – development/performance/deploy
- gRPC – standalone comm. framework
- MongoDB – both time-series and metadata
- HDF5* – reserved for legacy data/performance

Benchmark Description	Data Rates	
	Double Values (vals/sec)	Bytes (bytes/sec)
gRPC network transmission (Java)	22M – 33M	176M – 264M
Archiving, structured - HDF5 large	68M – 77M	544M – 616M
Archiving, structured - JSON files	38M – 47M	304M – 376M
Archiving, buckets - MongoDB	7M – 11M	56M – 88M
Archiving, buckets - MariaDB	4.5M – 5.5M	36M – 44M
Archiving, structured - HDF5 small	1.3M – 2.4M	10.4M – 19.2M
Archiving, points - InfluxDB	750K – 940K	6M – 7.52M
Archiving, points - MongoDB	360K – 410K	2.88M – 3.28M
Archiving, points - MariaDB	140K – 162K	1.12M – 1.3M
Metadata updates - MongoDB	11K to 36K updates/sec	-

Data Platform component evaluation and benchmarking

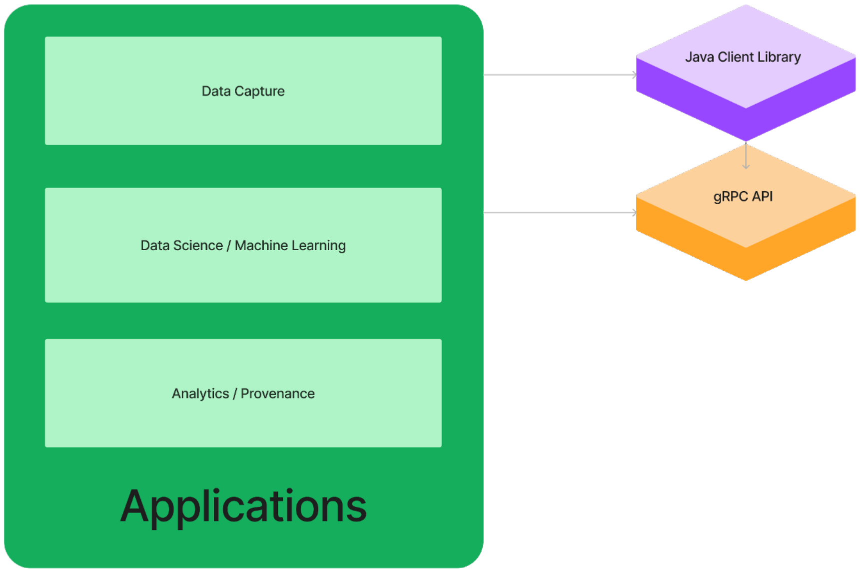
Supplemental Material

- SBIR Project Background
- Ingestion Data Flow Diagram
- Bucket Pattern for Time-Series Data
- DP Deployment
- Example Data Flow
- Why use an API?
- Authentication / Authorization
- Performance Benchmarks
- ***Client Application Communication Options***
    - ***gRPC API vs. Client Library***
    - ***gRPC API Client***
    - ***Java Client Library Ingestion Example***
    - ***Java Client Library Query Example***

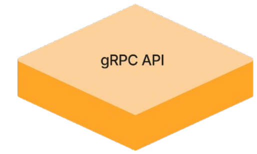
Client Application Communication Options

gRPC API vs. Client Library

Applications can be built at two levels, either using the lower-level gRPC API directly or using the higher-level client library.



Client Application Communication Options



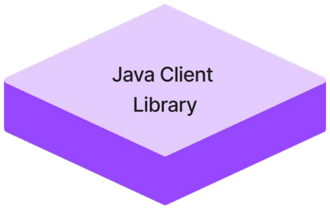
gRPC API Clients

- The low-level gRPC Ingestion and Query Service APIs can be used from a wide variety of programming languages to build applications.
- The "protoc" compiler generates appropriate code for using the data structures and invoking procedures defined in the service "proto" API files.

```
// Create gRPC request message.
IngestionRequest.Builder requestBuilder = IngestionRequest.newBuilder();

// Set event timestamp in request.
Timestamp.Builder snapshotTimestampBuilder = Timestamp.newBuilder();
snapshotTimestampBuilder.setEpochSeconds(params.snapshotTimestampSeconds);
snapshotTimestampBuilder.setNanoseconds(params.snapshotTimestampNanos);
snapshotTimestampBuilder.build();
requestBuilder.setSnapshotTimestamp(snapshotTimestampBuilder);

// Set data in request.
IngestionDataFrame dataFrame = ingestionDataFrameForTable(pvDataTable);
requestBuilder.setIngestionDataFrame(dataFrame);

// Build and send request.
IngestionResponse = blockingStub.ingestData(requestBuilder.build());
```

Client Application Communication Options

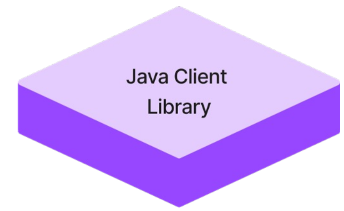
Client Library



- A Java client library provides a high-level data science oriented interface to the data platform.
- It hides the gRPC API implementation from developers, allowing them to focus on the application instead of communication details.

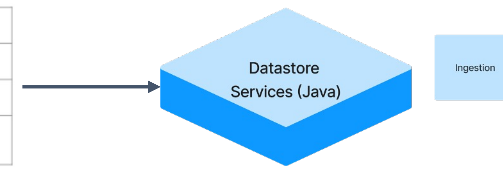
Client Application Communication Options

Java Client Library Ingestion Example



Example: Open streaming connection to Datastore Ingestion Service, create data frame for current interval, send to service.

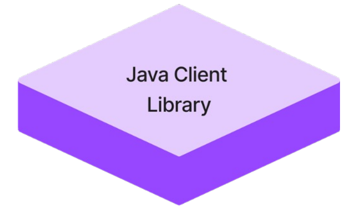
Time	scalar PV	array PV	image PV	structure PV
2023-04-18 08:30:00.000	scalar value	array value	image value	structure value
2023-04-18 08:30:00.100	scalar value	array value	image value	structure value
2023-04-18 08:30:00.200	scalar value	array value	image value	structure value




```
// Create interface to streaming ingestion service.
IIngestionStream ingStream = DsIngestionServiceFactory.connectStream();

// Open stream for PV data provider registration.
ingStream.openStream(pvProviderRegistration);

// Create DataFrame from PV table for current interval.
DataFrame dataFrameCurrentInterval = DataFrame.from(pvValueTable);

// Send data frame to ingestion service in current stream.
ingStream.streamData(dataFrameCurrentInterval);
```

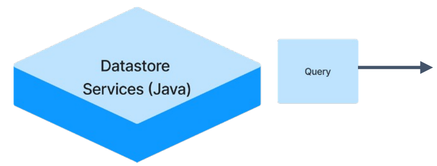
Client Application Communication Options

Java Client Library Query Example



Objective: Retrieve PV data from Datastore Repository to train a machine learning model, or feed data to predictive ML model embedded in a control application.

Approach: Use Datastore streaming query API to retrieve latest BPM values for relevant PVs and feed them to ML model.



Time	scalar PV	array PV	image PV	structure PV
2023-04-18 08:30:00.000	scalar value	array value	image value	structure value
2023-04-18 08:30:00.100	scalar value	array value	image value	structure value
2023-04-18 08:30:00.200	scalar value	array value	image value	structure value

```
/ Create interface to query service.
IQueryServiceData qrySvc = DsQueryServiceFactory.connectData();

// Create query request for relevant PVs starting from now.
qryRequest = qrySvc.newRequest();
qryRequest.rangeAfter(Instant.now()); // or use "rangeBetween" to specify time range
qryRequest.selectPvs(”S01-GCC01", ”S01-GCC02");

// Send query request and invoke callbackFunction with query.
IDataTableDynamic tblResult = qrySvc.requestDataAsync(qryRequest, callbackFunction);
```