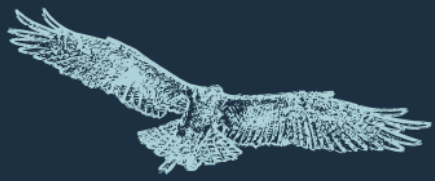


Osprey

Distributed Control Systems

Machine Learning **Data Platform Overview**

Christopher K. Allen
Craig McChesney
Mitch Frauenheim

Topics

- ***MLDP Overview***
- Survey of MLDP Elements
- Project Status and Road Map
- Summary

MLDP Overview

Motivation

Objective: To provide 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.

- Full stack: From high-speed data acquisition to rapid ML/AI application development and deployment.
- MLDP gives data science perspective to archived facility data.

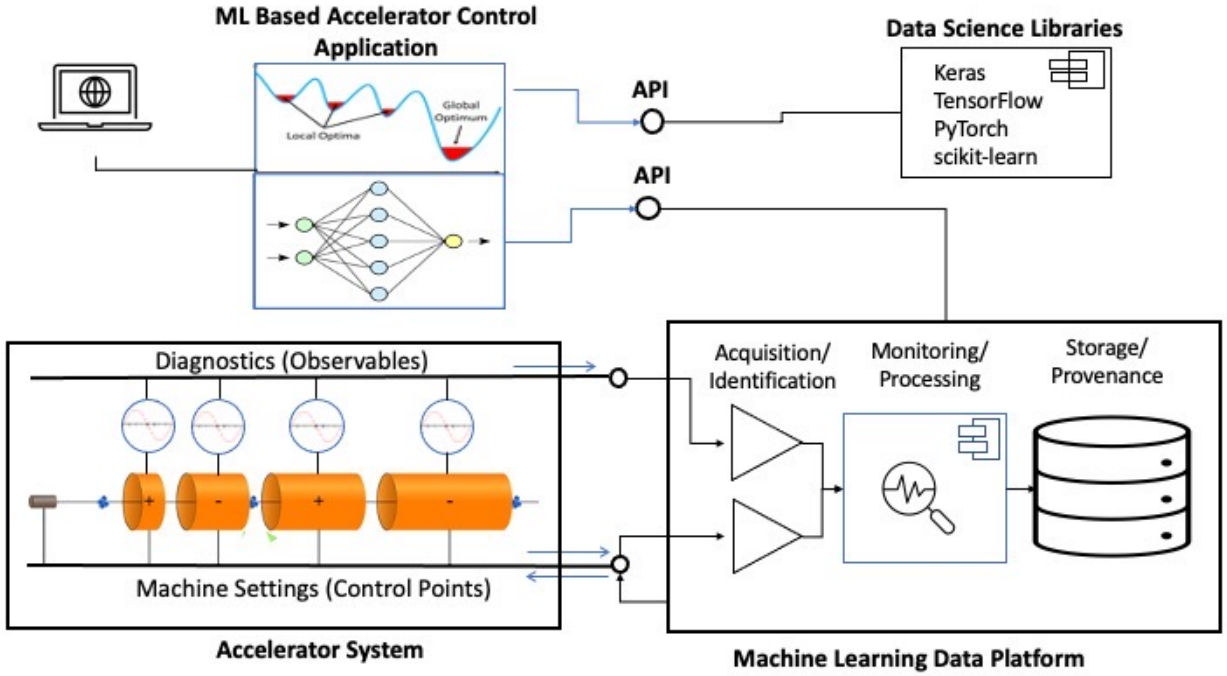
MLDP Overview

Concept

The **Machine Learning Data Platform** (MLDP) has 4 primary functions:

1. High-speed data acquisition.
2. Managing archive of heterogeneous, time-correlated data.
3. Providing a variety of options for retrieving and manipulating data and metadata from the archive.
4. Capturing “value-added” information in the form of annotations to archive data.

These functions are realized by **separate subsystems** each supporting a category of use cases.



Conceptual diagram of Machine Learning Data Platform

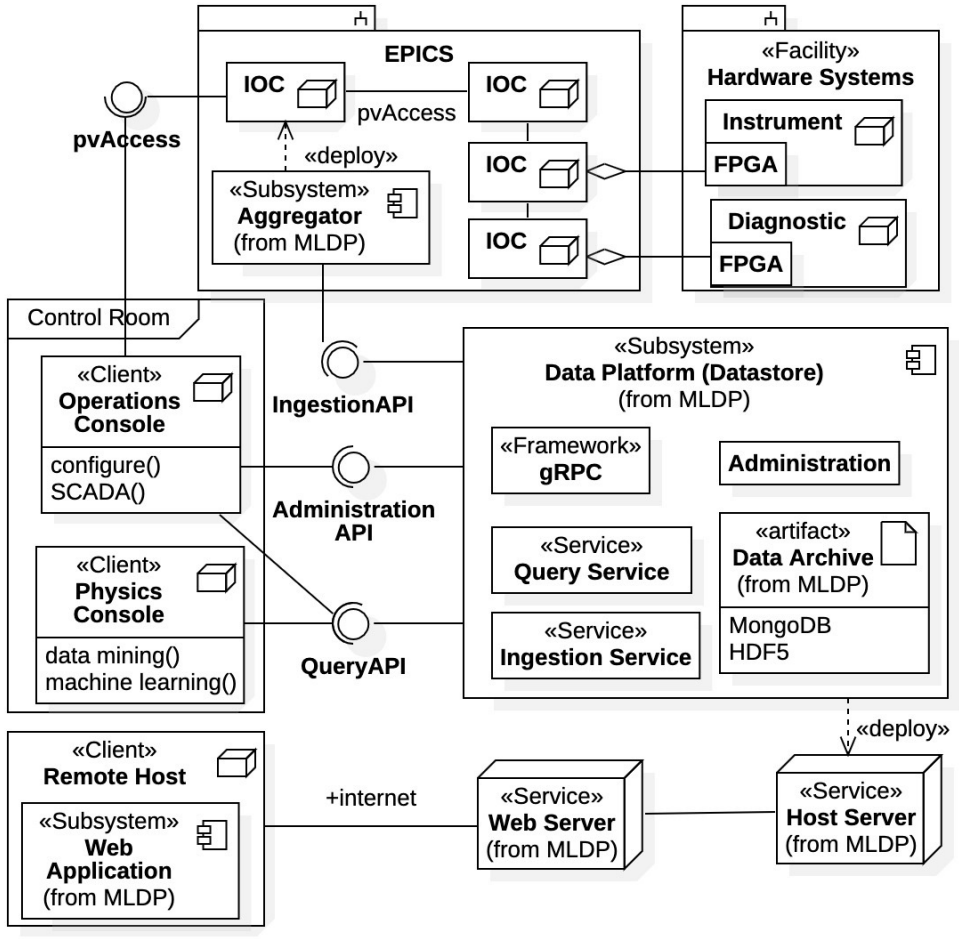
MLDP Overview

Subsystems

1. **Aggregator / Data Provider**
   - High-speed data acquisition and collection of facility hardware data.
   - Interface between control system (e.g., EPICS) and MLDP data ingestion mechanism.

2. **Core Services**
   - Suite of collaborating services.
   - Primary MLDP interaction point for clients.
   - A standalone system that can be installed at any facility, independent of EPICS.
   - Well-defined APIs for communication.

3. **Web Application**
   - Exposes subset of Core Services features via browser app.



Machine Learning Data Platform subsystems and deployment

Topics

MLDP Overview

Survey of MLDP Elements

Project Status and Road Map

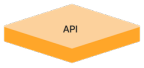
Summary

Survey of MLDP Elements

MLDP System Architecture

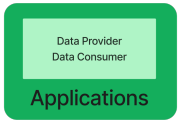
Data Archive

Core Services



API

Client Libraries



Applications

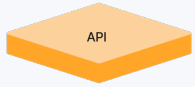
Web Application

MLDP Elements

System Architecture

Data Archive is used by Core Services to manage accelerator data.

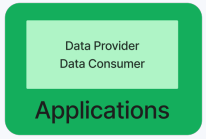
Core Services handle ingestion, query, and annotation APIs.



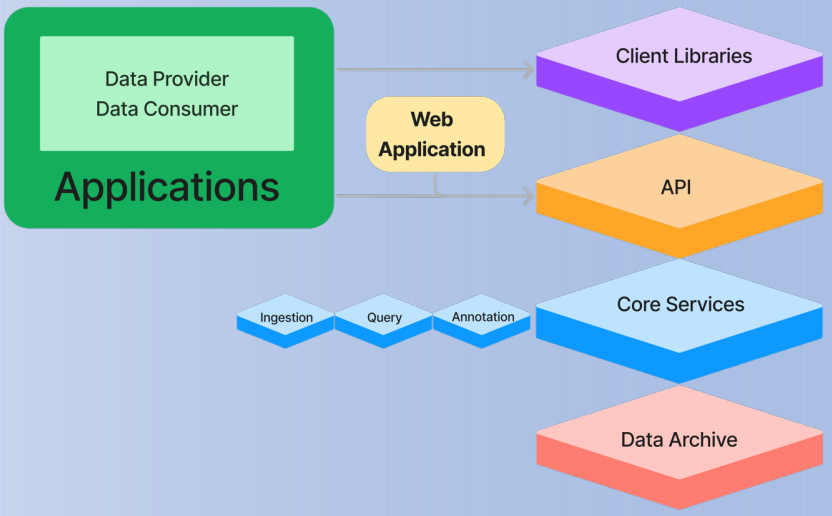
API defines methods and data types for interacting with Core Services.

Web Application provides browser access to Core Service features.

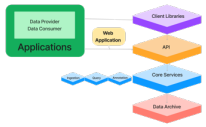
Java and Python Client Libraries are provided for higher-level interaction with Core Services.



Applications communicate with Core Services via API or Client Libraries..



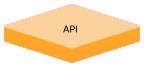
Survey of MLDP Elements



MLDP System Architecture

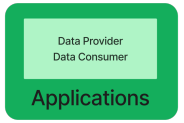
Data Archive

Core Services



API

Client Libraries



Applications

Web Application

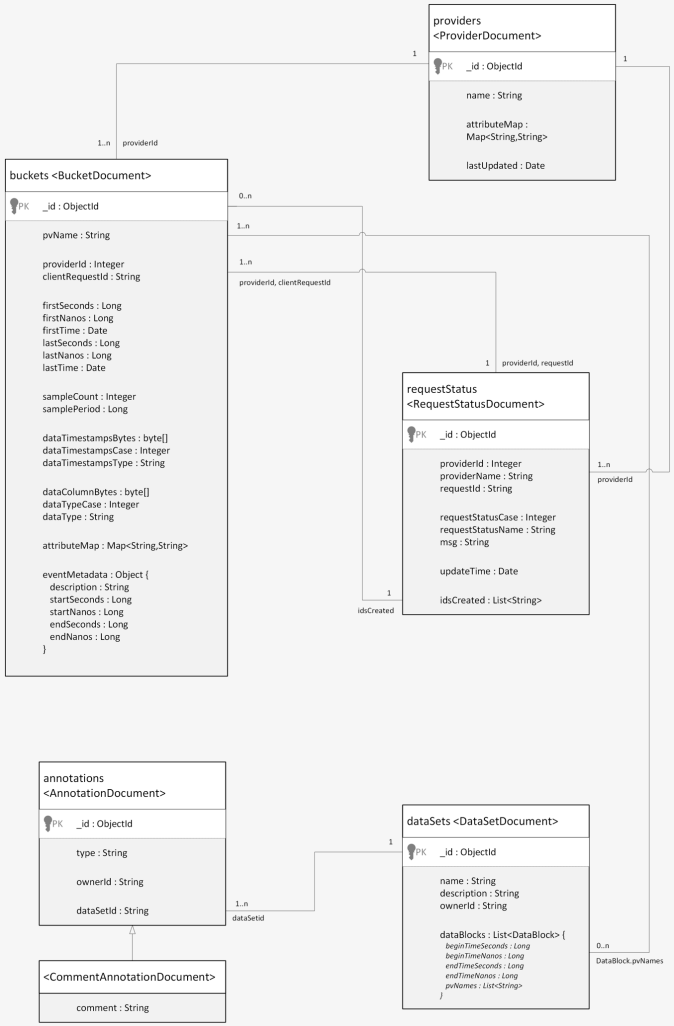
MLDP Elements

Data Archive

The Core Services manage the Data Archive, which contains the following information:

- Heterogenous, correlated, time-series device data.
- Metadata describing characteristics of time-series data and its sources.
- Annotations (notes, relations, calculations) made to datasets in the archive.
- Provenance and relationships between datasets.
- Status information used in monitoring to determine disposition of ingestion requests handled asynchronously.

This information is maintained as document collections in MongoDB.

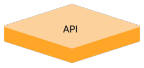


Survey of MLDP Elements

MLDP System Architecture

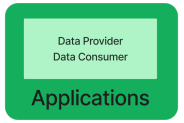
Data Archive

Core Services



API

Client Libraries



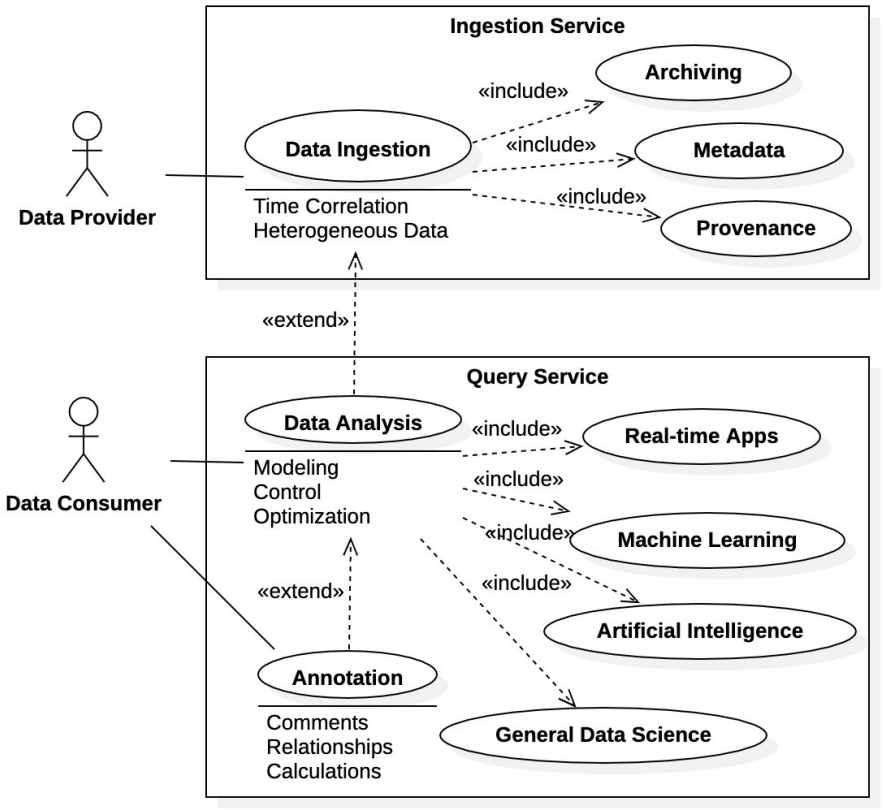
Applications

Web Application

MLDP Elements

Core Services – MLDP Service Model

- The Core Services manage the Data Archive and are the primary interaction point for Provider and Consumer clients.
- There are 3 MLDP services each providing an API covering a subset of the system use cases:
  - **Ingestion Service** is the focal point for capturing data to the Archive.
  - **Query Service** is the focal point for retrieving and processing time-series data and metadata from the Archive.
  - **Annotation Service** is the focal point for supplying ”value added” information to the Archive in the form of annotations.



Data Platform Use Cases

MLDP Elements

Data Platform / Core Services

Ingestion Service

- The Ingestion Service provides APIs for registering data providers, ingesting data, and querying the status of ingestion requests.
- Processing is asynchronous to maximize performance.
- Writes time-series data “buckets” to MongoDB, each containing a vector of data values for a single PV over a time range.
- Buckets store data values using serialized Protocol Buffer format.
- Bucket time range is specified via start time, sample period, and number of samples (or explicit list of timestamps).

Sample Ingestion Service API methods

```
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.
  - querying metadata about PV data sources and characteristics of time-series data.
- Bucketed time-series data query response format contains data buckets matching query's PV and time range criteria.
  - Each bucket contains a data vector for the specified PV over a specific time range.

Sample Query Service API methods

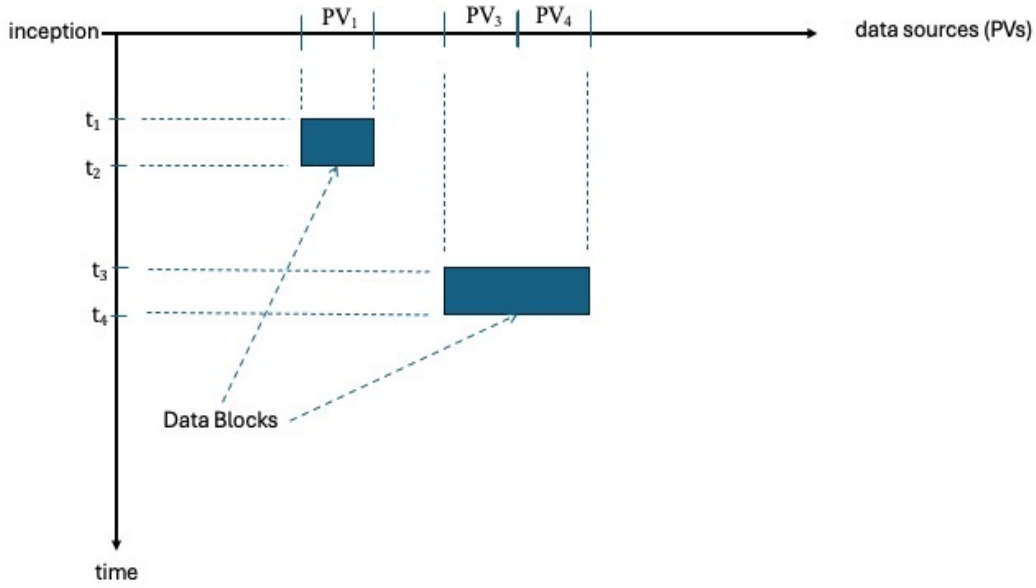
```
rpc queryDataStream(QueryDataRequest) returns (stream QueryDataResponse);
rpc queryTable(QueryTableRequest) returns (QueryTableResponse);
rpc queryMetadata(QueryMetadataRequest) returns (QueryMetadataResponse);
```

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.



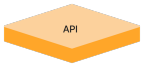

```
rpc createDataSet(CreateDataSetRequest) returns (CreateDataSetResponse);
rpc queryDataSets(QueryDataSetsRequest) returns (QueryDataSetsResponse);
rpc createAnnotation(CreateAnnotationRequest) returns (CreateAnnotationResponse);
rpc queryAnnotations(QueryAnnotationsRequest) returns (QueryAnnotationsResponse);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

Survey of MLDP Elements

MLDP System Architecture

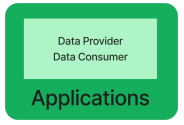
Data Archive

Core Services



API

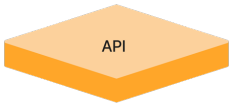
Client Libraries



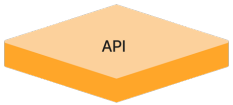
Applications

Web Application

MLDP Elements
API



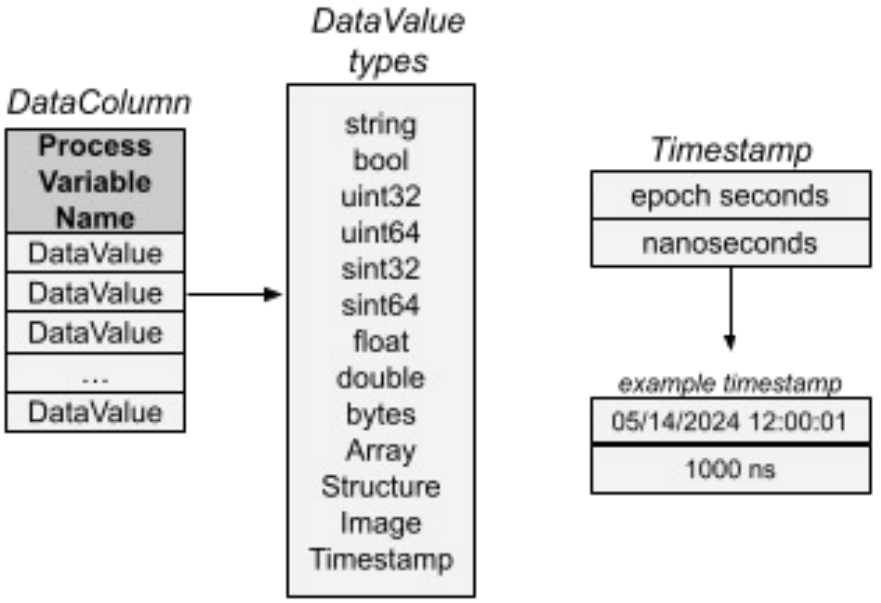
- The gRPC communication framework was originally developed by Google for use in connecting microservices.
- It uses HTTP/2 for transport, and Protocol Buffers as both the interface definition language and message interchange format.
- Supports simple unary single request / response APIs as well as unidirectional 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

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).

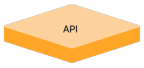


Survey of MLDP Elements

MLDP System Architecture

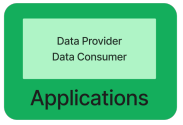
Data Archive

Core Services



API

Client Libraries



Applications

Web Application

MLDP Elements

Client Libraries

Client Libraries provide high-level tools for working with the MLDP service APIs, avoiding the complexities of direct gRPC communication. Libraries are currently under development for building client applications in Java and Python.

Example ingestion library features:

- Utilities for working with EPICS normative times and pvAccess interface.
- Breaking a large dataset into manageable chunks for ingestion.

Example retrieval and processing library features:

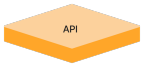
- Subscribing to time-series data for specified PVs with updates at specified time interval.
- Maintaining data for a set of PVs over a rolling window of time.
- Assembling bucketed time-series data from the query response to a dynamic tabular structure.
- Packaging results for use with common 3rd party packages like Pandas and numpy.

Survey of MLDP Elements

MLDP System Architecture

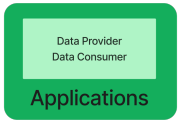
Data Archive

Core Services



API

Client Libraries

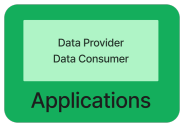


Applications

Web Application

MLDP Elements

Applications: Aggregator / Data Provider



Aggregator / Data Provider is an application that interfaces the EPICS control system to the MLDP Ingestion Service, performing high-speed device data acquisition and collection.

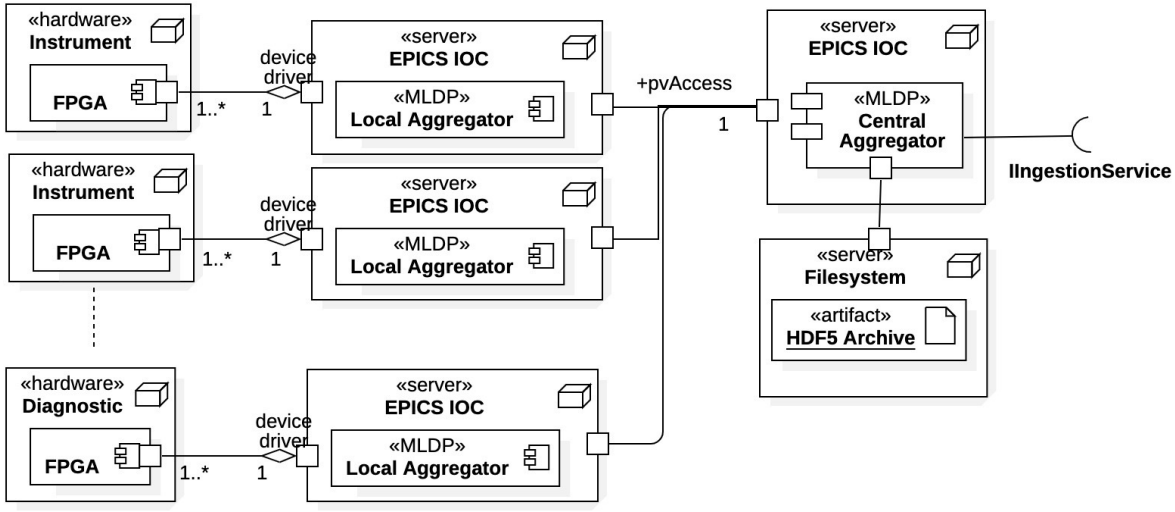
The Aggregator System Architecture includes Local and Central Aggregators.

Local Aggregators

- Proximal to hardware.
- Collect and align local hetero data.
- May have multiple data sources.
- Transport to Central Aggregator.

Central Aggregator

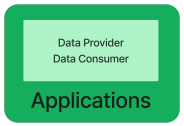
- Coalesce all aggregated data.
- Stage as NTTable formatted snapshots.
- Transport to MLDP Ingestion Service.



Aggregator System Architecture

MLDP Elements

Applications: Example gRPC Client Application



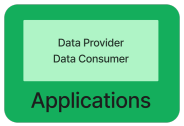
The low-level gRPC Ingestion and Query Service APIs can be used from a wide variety of programming languages to build applications.

```
// 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());
```

Example: *Create ingestion request, set timestamp and add data, build and send request.*

MLDP Elements

Applications: Example Client Library Application



Use of Client Libraries provides higher-level interface for building applications.

```
// 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);
```

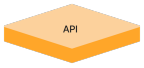
Example: *Open streaming connection to Ingestion Service, create data frame for current interval, send to service.*

Survey of MLDP Elements

MLDP System Architecture

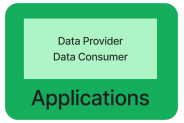
Data Archive

Core Services



API

Client Libraries



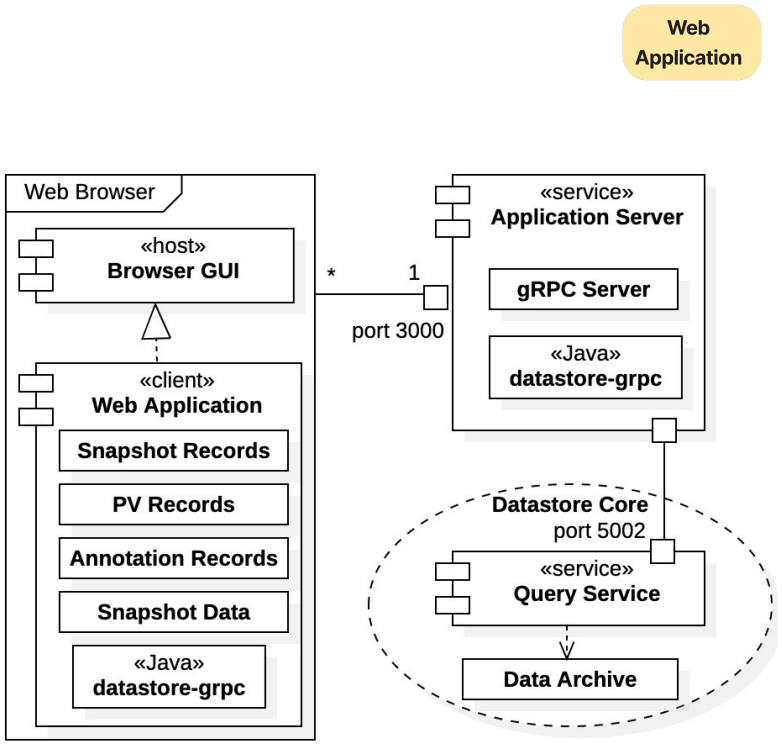
Applications

Web Application

MLDP Elements

Web Application

- MLDP Web Application is a browser-based application built using JavaScript.
- Provides remote access to the MLDP Core Services:
  - Facilitates a subset of Query and Annotation Service features.
  - Provides tools for navigation, inspection, visualization, download, and data processing / analysis.
- Web Application Server provides an API that augments the core Data Platform API.



Web Application Architecture

Topics

MLDP Overview

Survey of MLDP Elements

Project Status and Road Map

Summary

MLDP Project Status

- Fully implemented Core Service APIs:
  - Ingestion: provider registration, ingestion of time-series data, querying status of async ingestion requests.
  - Query: querying time-series data in bucketed and tabular format, querying Data Archive metadata.
  - Annotation: Identifying datasets within the Archive, adding annotations to them, querying annotations and retrieving dataset time-series data
- Eliminated many 3rd party dependencies from original prototype – now restricted to Java, gRPC, and MongoDB.
- Created suite of utilities for managing MLDP ecosystem.
- Created deployment tools and “quick start” installer.
- Ingestion over 200x faster than prototype at ~ 200 MBps data ingestion rates.
- Evaluated C++ gRPC for datastream processing (~ 500 MBps transmission rates).

MLDP Road Map

- Continue adding features to Core Services:
  - Ingestion: Framework for measuring data ingestion statistics.
  - Annotation: Export mechanism, additional annotation types.
- Continue development of Client Libraries.
- Perform more extensive load and scale testing.
- Build data generator / simulator.
- Add mechanism for customizable, real-time processing of ingestion data stream.
- Incorporate authentication and authorization:
  - Token based authentication for query clients and web application.
  - Role-based authorization.
- Create infrastructure for system administration:
  - Age-based archival of data to HDF5 files to manage database footprint.
  - Tools for replication / synchronization of archive to distributed researchers.
- Investigate approaches for performance improvement:
  - MongoDB clustering, partitioning (sharding), and connection pooling.
  - Horizontal scaling for deployment of core services.
  - Optimized tools for data streaming between components e.g., Apache Kafka

Topics

MLDP Overview

Survey of MLDP Elements

Project Status and Road Map

Summary

Summary

The Core Services are the foundation of the MLDP.

- This collaborating suite of services manages an archive of heterogenous data with a focus on data science applications.
- Each service offers a well-defined API.
- The initial APIs for the Ingestion, Query, and Annotation services are defined, implemented and fully functional – and will continue to evolve.

The MLDP offers two modes of communication, supporting a wide range of application types and programming styles.

- Direct gRPC API communications is supported for most programming languages.
- Client Libraries are provided for Java and Python applications.
- APIs support both simple request-response interaction and sophisticated bidirectional streaming.

Ingestion performance is a key consideration and primary focus.

- The current MLDP implementation exceeds the baseline performance objective initially established for the project, and continuous improvements are expected.

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

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.

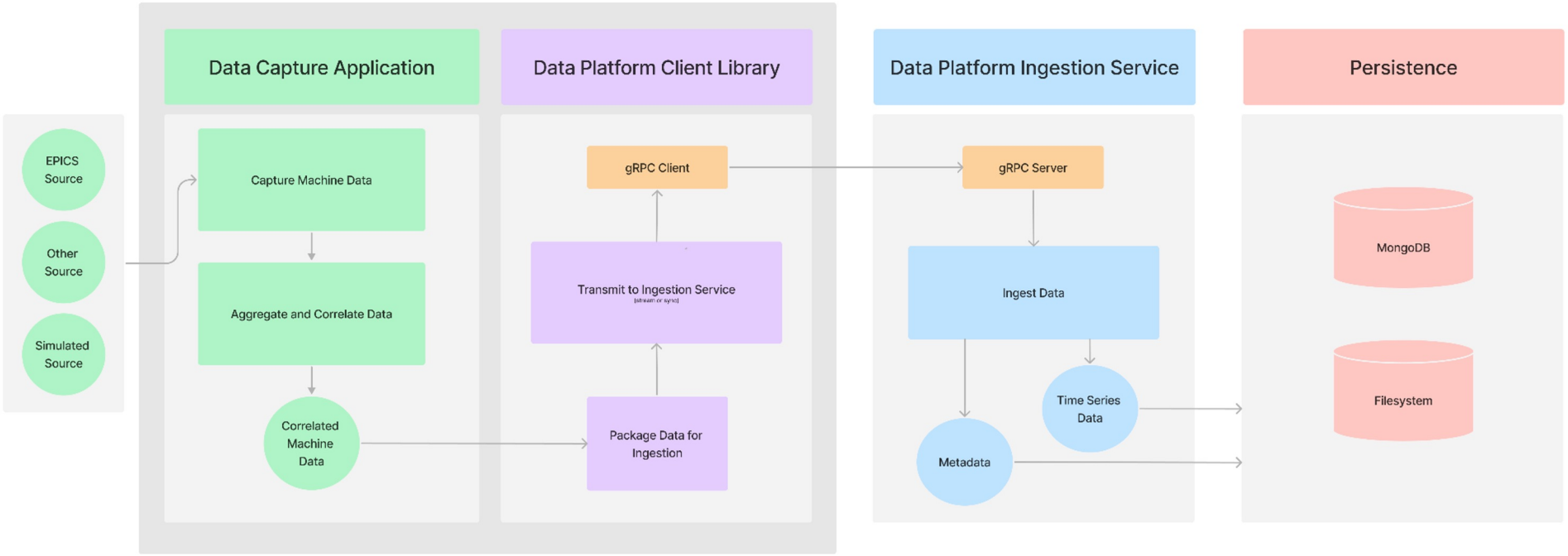
- "**Data Platform**" (DP) references previous "**Datastore**" system of prototype MLDP.
  - Includes upgrades and extensions for Phase II project.

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

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

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

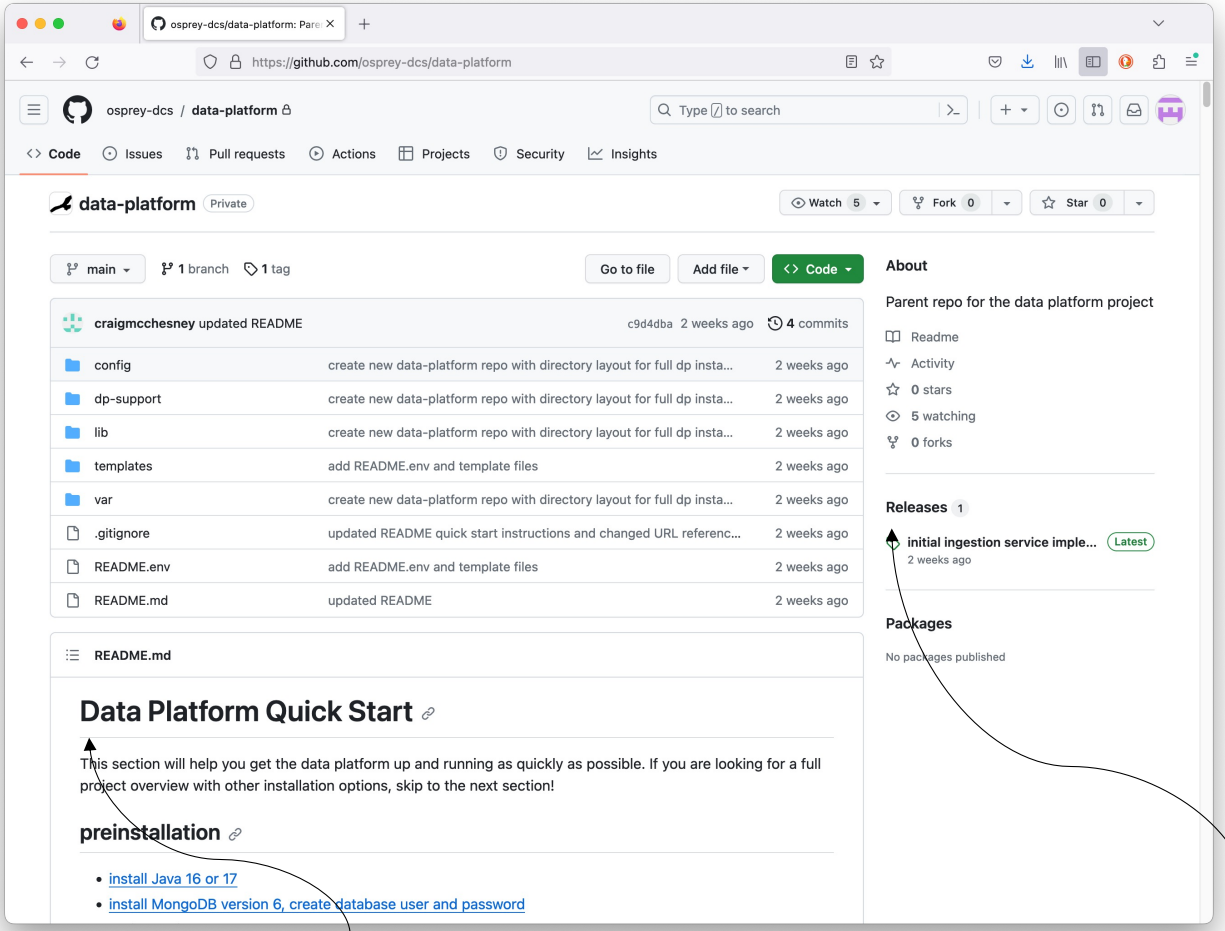
- SBIR Project Background
- Ingestion Data Flow Diagram
- Bucket Pattern for Time-Series Data
- ***DP Deployment***
- Example Data Flow
- Why use an API?
- Authentication / Authorization
- Performance Benchmarks

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

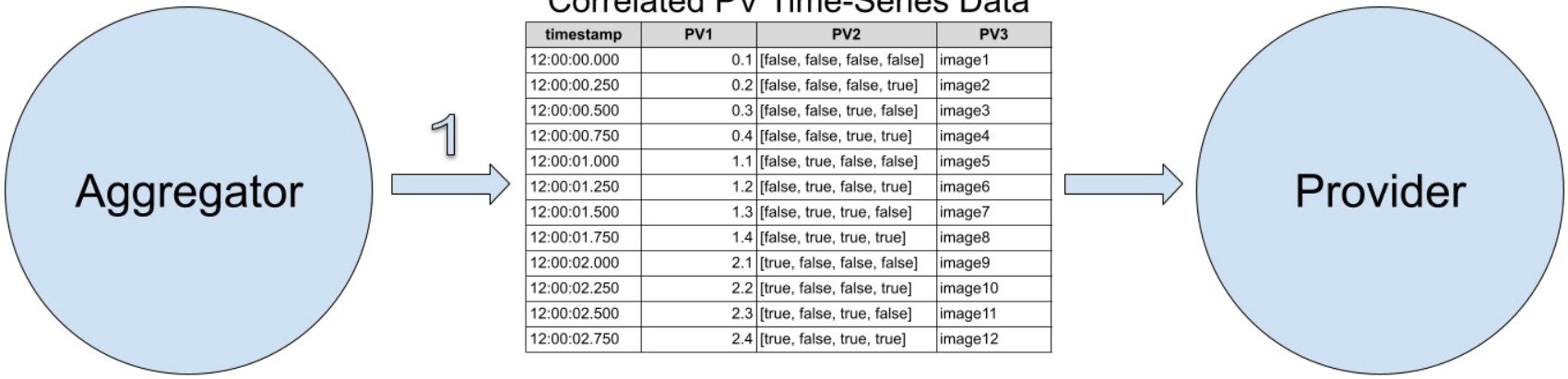
data-platform-installer.tar.gz - Download and unzip.

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

Example Data Flow

Aggregator -> Provider



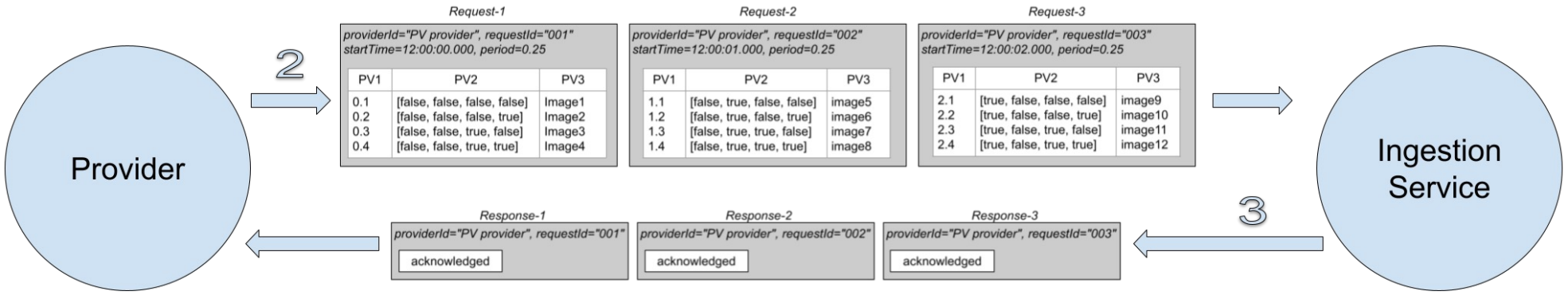
timestamp	PV1	PV2	PV3
12:00:00.000	0.1	[false, false, false, false]	image1
12:00:00.250	0.2	[false, false, false, true]	image2
12:00:00.500	0.3	[false, false, true, false]	image3
12:00:00.750	0.4	[false, false, true, true]	image4
12:00:01.000	1.1	[false, true, false, false]	image5
12:00:01.250	1.2	[false, 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

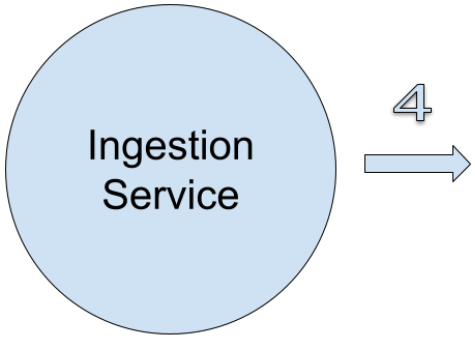
External Provider app invokes Ingestion Service's ingestDataStream() RPC method, sending a stream of ingestion requests each containing a set of PV data vectors for the specified start time.

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

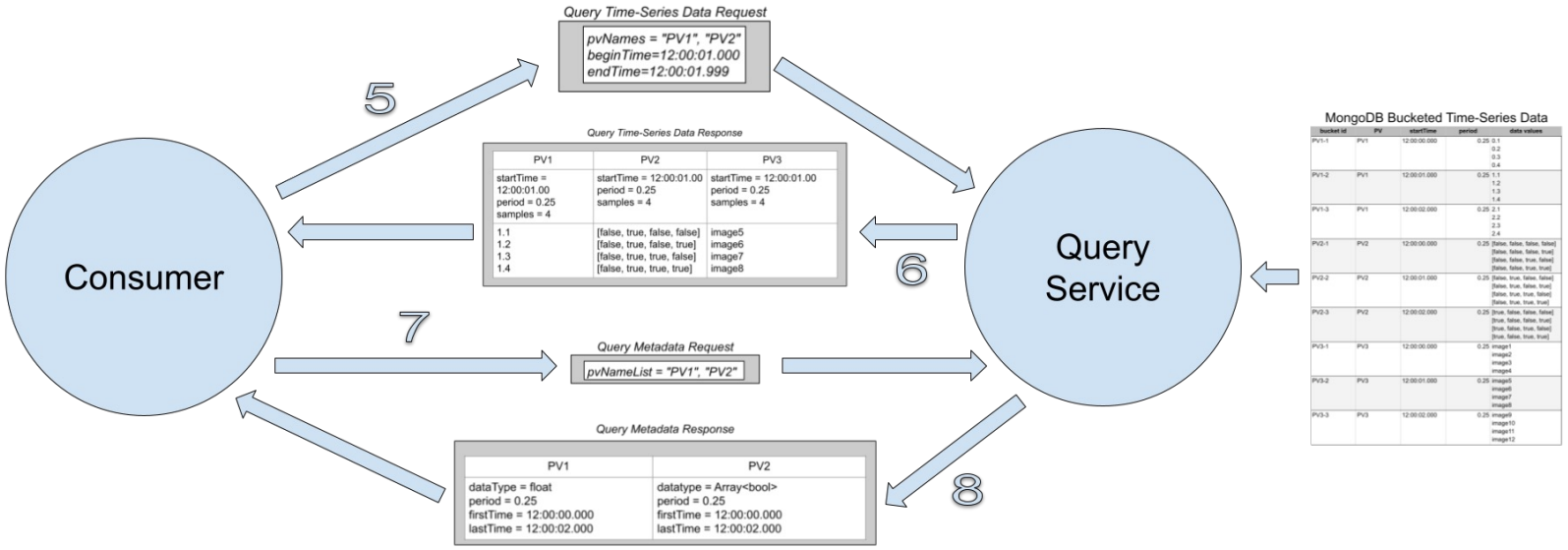
Ingestion Services processes queue of requests, writing bucketed time-series data to MongoDB. Each bucket contains a list of data values for the specified PV and time range.

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.

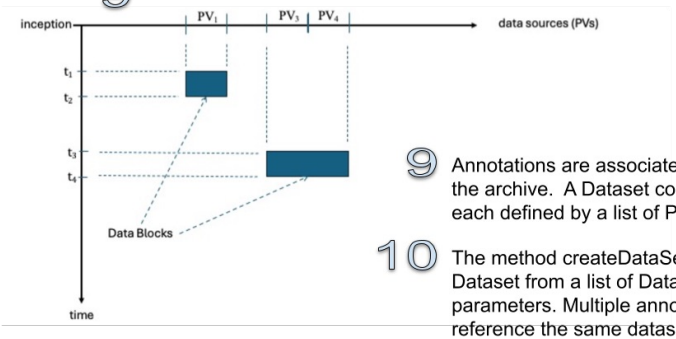
7 Client sends metadata query request via queryMetadata() RPC method to learn about data sources available in the archive, specifying list of PV names or regex pattern.

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

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

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***

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