GE VERNOVA

DIGITAL

PROFICY HISTORIAN EDGE

User Guide

Contents

Copyright GE Digital

Chapter 1. Release Notes - Historian for Linux 2.3.1

Release Notes

Table 1. New Features and Enhancements

The following new features and enhancements have been added.

Description	Tracking ID
Support for Array tags in Linux historian Historian for Linux now provides support for array tags.	F70135
Historian Rest Query Service Historian Rest Query service now supports querying the Array tag data type.	F70135
Historian MQTT Collector Historian MQTT Collector now supports Array tags.	F70374

Table 2. Resolved Issues

The following issues have been resolved.

Description	Tracking ID
Previously, when using the MQTT collector, the `Output.msq` buffer file size did not decrease after reconnecting and sending data to the destination Historian server. This issue has been resolved in the following image ID of Historian for Linux: 0ccd57605973	DE136980

Table 3. Known Issues

The following issues are unresolved.

Description	Tracking ID
When using the public REST APIs, an error occurs if you use the host name of the source Historian server. **Workaround:** Use the IP address instead of the host name.	DE130496
When using the REST query, an error occurs if you query the latest data point using the GET method. **Workaround:** Use the POST method instead of the GET method.	DE136935
In the Historian archiver, cyclic archiving does not work.	DE137212
When you query data using the REST query service, the filter condition is not honored when the order is by desc.	DE136984
Sometimes, the server-to-server collector crashes with status code as 139 in the Docker logs. This issue appears when you restart the source Historian server multiple times.	DE102770
When using a tuner, you must run the data management task prior to the data store configuration and tag configuration tasks.	DE103207
When you attempt to back up the archive file using web admin, an error occurs.	DE102197
On web admin, the free-space statistics for time-based archive do not show the value 0 (zero),	DE103473
When using the server-to-server collector, sometimes, the database crashes. This happens when the server-to-server collector is running when the	DE101786

Table 3. Known Issues

The following issues are unresolved.

(continued)

Description	Tracking ID
database restarts and the collector tries to connect to the database. **Work around**: Stop the server-to-server collector, start the database, and then start the server-to-Server collector again.	
Using web admin, if you query a tag that contains { or } in the name, an error occurs.	DE103439

Chapter 2. Historian for Linux - an Overview

Overview of Historian for Linux

Historian for Linux is a high-performance timeseries database designed to store and retrieve time-based information at a high speed. It runs on the Linux platform using Docker.

Historian for Linux is a collection of several Docker images such as the Historian database, REST query, web admin, public REST APIs, and various Historian collectors.

Advantages of Using Historian for Linux

- **Time series data archiving**
- **REST API for data query:** Data query REST APIs are exactly the same as the Predix time series REST APIs.
- **Web admin:** An administrative console to work with the Historian database.
- **OAUTH2 integration:** The web admin, tuner, REST query, and public REST APIs leverage OAUTH2-based authentication and authorization.
- **Collector toolkit library:** Used for implementing collectors that ingest data into Historian for Linux.
- **User API library:** A C library for programmatically adding, deleting, and configuring tags for collectors.
- **Server-to-Server collector:** Stream data of one Historian to another Historian or to Predix Time series. It helps in data filtering.
- **OPCUA DA collector:** Collects data from an OPCUA DA server and ingests the data into Historian.
- **MQTT collector:** Subscribes to an MQTT broker and ingests the data into Historian. This collector helps to integrate Historian for Linux with the data bus of Predix Edge.
- **Configuration:** You can also change the configuration properties of the various applications such as the Historian database, web admin, tuner, REST APIs, REST query, and the various collectors using the JSON files provided with each application.
- **Public REST APIs**: Query data from the Historian for Linux archives. The APIs use a Docker container on a Linux machine. They use the port number 9090 for REST client requests.

Limitations

- Input to the MQTT collector must be in the time-series format.
- The protocol adapters must use the flat_to_timeseries block to translate the data to the required format before adding the data to the MQTT broker.
- Historian for Linux only supports message IDs, not messages.

Supported Operating System Platforms

Any x64 based Linux machine or a Linux virtual machine with Docker installed.

Licensing

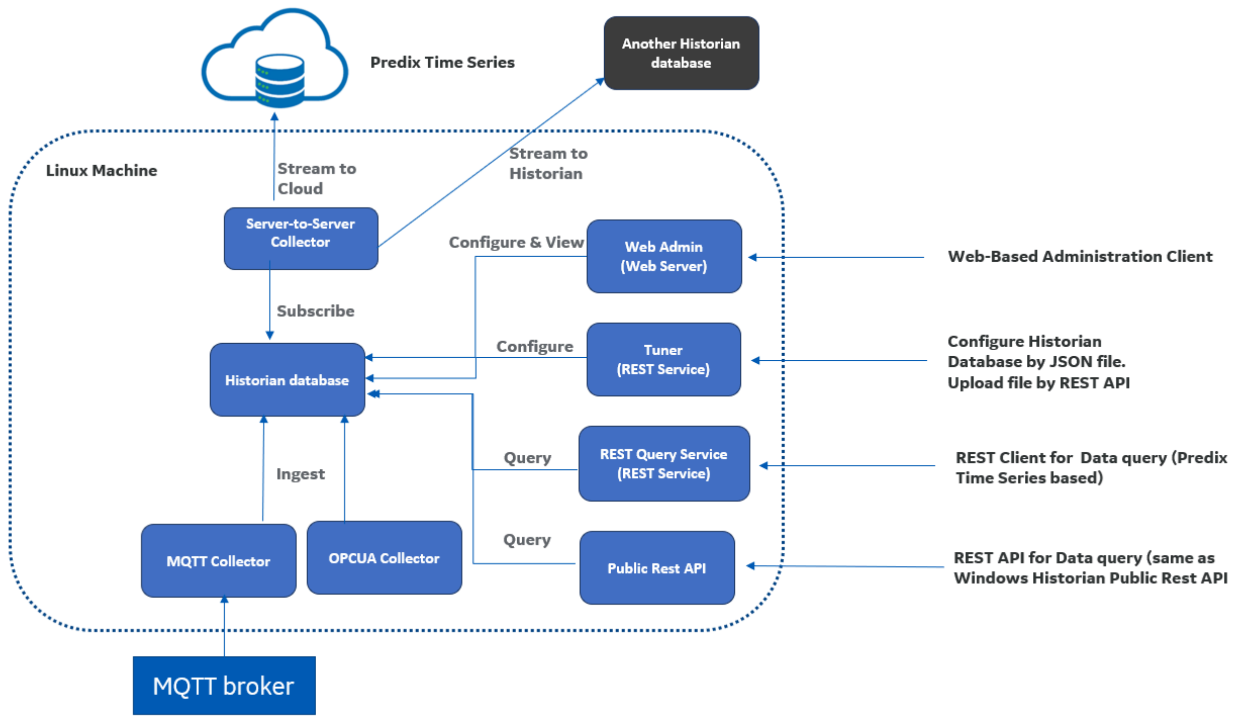
Historian for Linux is licensed separately from Predix Edge. This documentation provides the technical aspects of using Historian in the Predix Edge context. For information on purchasing Historian for Linux perpetual licenses, contact the GE Digital Sales team. We are working on providing additional commercial models for providing a limited time series capability at the Edge, but these are incomplete and not yet ready for publication.

Historian Container Architecture

Historian for Linux is developed using the microservices architecture concept. Microservice architecture is a minimalist approach to modular software development. Modularity is defined as the degree to which a system's components may be separated and recombined.

That is the reason each Historian container performs its unique job and the containers communicate with each other for solving different use cases. For example, the Historian database container is responsible for storing time series data to disk, whereas the REST query is responsible for exposing REST APIs for data query from the Historian database. You can choose to install all or any of these applications.

The following diagram shows the core components of Historian for Linux and how they interact with one another.


Predix Time Series
Another Historian database
Linux Machine
Stream to Cloud
Stream to Historian
Server-to-Server Collector
Configure & View
Web Admin (Web Server)
Web-Based Administration Client
Subscribe
Configure
Tuner (REST Service)
Configure Historian Database by JSON file. Upload file by REST API
Historian database
Query
REST Query Service (REST Service)
REST Client for Data query (Predix Time Series based)
Ingest
Query
Public Rest API
REST API for Data query (same as Windows Historian Public Rest API
MQTT Collector
OPCUA Collector
MQTT broker

Historian for Linux provides the following containerized components:

Docker 1: Historian Database

Historian Database is a C++ based native time series archiver based on the Historian archiving engine. The TCP/IP server listens on port 14000 and is configurable using Docker's port map technology.

For information, refer to Historian Database *(on page 23)*.

Docker 2: REST Query

The REST query is a Java-based REST service. It offers REST APIs for data query from the Historian database. It uses the OAUTH2 server for authentication and authorization. The REST query provides REST APIs that are similar to the Predix Time Series data query APIs for querying data. This implies that any analytics application developed using Predix Time Series data query APIs can work seamlessly with Historian for Linux.

For information, refer to Historian REST Query Service *(on page 34)*.

Docker 3: Web Admin

Web admin hosts a web-based admin console for the Historian database. You can view and edit properties and list of tags, list of collectors, and the list of data stores. It is a

Tomcat-based web service, which listens on port 9443 and uses the OAUTH2 server for authentication and authorization.

It also provides the ability to:

- Start and stop collectors
- Configure collectors
- Browse, add, delete, and rename tags, data stores, and archive files
- View status of the connected collectors
- View the most recently collected data

You can also use a UAA service with web admin.

For information, refer to Historian Web Admin Service *(on page 42)*.

Docker 4: Tuner

Tuner helps you to configure the Historian database properties such data archiving style (daily, hourly, or by size), tag properties (such as collection rate, conditional collection filtering). You must provide these configuration changes in the form of JSON payload in a file. It offers a REST API for uploading JSON file from REST clients to Tuner container. It uses OAUTH2 Server for authentication and authorization filtering.

For information, refer to Historian Tuner *(on page 51)*.

Docker 5: The Server-to-Server Collector

The server-to-server collector streams data from one Historian database (source Historian) to another Historian database (destination Historian) or to Predix time series.

You can use this collector when multiple Historian databases are deployed and you want the data of specific tags from one Historian database to be streamed to another Historian database with some data filtering.

For information, refer to Historian Server to Server Collector *(on page 73)*.

Docker 6: The OPCUA DA Collector

The OPCUA DA collector connects to the OPCUA Data Access (DA) server and can collect polled and asynchronous data. It then streams data to the Historian database. This collector can securely connect with the OPCUA DA server with certificate exchange.

For information, refer to Historian OPCUA DA Collector *(on page 80)*.

Docker 7: The MQTT Collector

The MQTT collector connects to MQTT broker and subscribes for topics. The data in the JSON payload must be in the Predix time series format. The collector automatically adds tags and streams data to the Historian database. Because of this collector, Historian for Linux is well integrated with Predix Edge's data bus for consuming data. Predix Edge's data bus is an MQTT broker.

For information, refer to Overview of the MQTT Collector *(on page 47)*.

Docker 8: Public REST APIs

Public REST APIs query data from the Historian for Linux archives.

For information, refer to Overview of the Public REST APIs *(on page 30)*.

Chapter 3. Set Up Historian for Linux on Predix Edge

Set Up Historian for Linux on Predix Edge

- Create a Predix account, and get access to a Predix Edge OS 2.5 device.
- Ensure that you have a super-user access to the Linux machine on which you want to install Historian. This is required if you want to use the OPC UA DA collector.

Important:
Before you install Historian for Linux on a Predix Edge OS for an ESXi Production image, you must ensure that you have access to the terminal. To get this access, contact Edge_Engineering_Support@ge.com.

This topic describes how to set up Historian on Predix Edge. You can also set up Historian on a generic Linux distribution *(on page 17)*.

1. Access https://artifactory.predix.io/, and download the .tar.gz and .zip files for each application that you want to install.
2. Access Predix Edge Technician Console (PETC).
3. Upload the applications that you want to install.
4. Deploy the applications that you want to install.
   The applications appear in the **DEPLOYED INSTANCES** section.
5. Extract the .zip file for each application that you have installed. This file contains JSON files with the environment variables that will be used by the applications.
6. Set the environment variables for each application that you have installed:
   - Historian database *(on page 24)*
   - REST query *(on page 34)*
   - Web Admin *(on page 42)*
   - Tuner *(on page 52)*
   - Server-to-Server collector *(on page 73)*
   - MQTT collector *(on page 48)*
   - Public REST APIs *(on page 33)*
   - OPC UA DA collector *(on page 82)*

   Environment variables act as command line arguments for the Docker containers, which use the value of these environment variables to configure the application running inside each of them.

Important:
The Historian for Linux product license is deployed as a configuration of Historian Database application. To activate the license, compress your Historian for Linux product license, and apply configuration to Historian database application.

Important:
For OPC UA DA, Server-to-Server, and Server-to-Cloud collectors, ensure that you have set the environment variables correctly before applying the configuration changes. This is because after you have applied the configuration changes for a collector, you cannot set the environment variables. If, however, you must change the values of the environment variables after the applying the changes, you must change the InterfaceName and DefaultTagPrefix values. When you do so, a new instance of the collector is created. As a result, you must add the tags again.

7. Compress the contents of each .zip file that you have extracted.
8. After each application is deployed and running, apply configuration. Choose the .zip file for the application that you want to configure.

Tip:

- If you are using a Predix Edge OS device for an ESXi Developer image, to verify that the applications are running:
    a. Access the Edge OS machine using PuTTY. By default, the username and password is root.
    b. Navigate to the `/var/lib/edge-agent/` folder.
    c. Run the following command: `/opt/edge-agent/app-list`

       A list of applications that are running appears.
- To verify the logs of an application:

a. Access the Edge OS machine using PuTTY. By default, the username and password is root.
b. Navigate to the `/var/lib/edge-agent/app/<name of the application>/data` folder, and then access the logs folder.

For example, for Historian database, the `archiver/archives` folder contains the .iha and .ihc files, and for all the collectors, the `Logs` folder contains the .log and .shw files.

Stop an Application on Predix Edge

You can stop an application if you want to modify the configuration for the application. For example, if you want to modify the properties of tags used by the server-to-server collector, you must first stop the application, modify the properties, and then run the application.

1. Access Predix Edge Technician Console (PETC).
2. In the **DEPLOYED INSTANCES** section, select the check box corresponding to the application that you want to stop.
3. Select **Action > Stop**.
   The application is stopped.

Upgrade an Application on Predix Edge

1. Stop the application that you want to upgrade.
2. Set up the application again *(on page 14)*.

Uninstall an Application on Predix Edge

When you uninstall an application, all the files used by the application are deleted. For the Historian database application, the archive files are deleted as well. Therefore, exercise caution before uninstalling the Historian database.

Delete the application that you want to uninstall.
The application is uninstalled.

Chapter 4. Set Up Historian for Linux on a Generic Linux Distribution

Set Up Historian on a Generic Linux Distribution

- Ensure that you have a super-user access to the Linux machine on which you want to install Historian. This is required if you want to use the OPC UA DA collector.
- Historian for Linux uses Docker containers. Therefore, you must install Docker on an x64 Linux machine on which you want to install Historian for Linux. For instructions on installing Docker, refer to https://docs.docker.com/engine/install/, and then select the platform.
- Install Docker Compose. To verify that Docker is set up, run the following command: `docker pull hello-world`. Or, refer to https://docs.docker.com/config/daemon/systemd/ to set up the proxy for Docker daemon, which pulls Docker images from the Docker hub.
- Install vi editor. To install vi editor on Ubuntu, run the following command: `apt-get install vim`
- Install the zip package. To install the zip package on Ubuntu, run the following command: `apt-get install zip`
- Install the curl package by running the following command: `apt-get install curl`

This topic describes how to set up Historian on a generic Linux distribution such as Ubuntu or CentOS. You can use any Linux distribution that has a Linux kernel with version 3.10 or later. To verify the Linux kernel version, run the following command: `uname -v`

You can also set up Historian on Predix Edge *(on page 14)*.

1. Download the Historian installation package, and place all the contents of the package in a single folder.
2. Access the root node of the Linux machine on which you want to install Historian.
3. Provide the executable permissions to all the scripts by running the following command: `chmod +x *.sh`
4. Depending on the application that you want to install, run the command as provided in the following table.

Application	Command
Historian database and the REST query service	`./install.sh historian-linux`
MQTT collector	`./install.sh mqtt`
OPCUA DA collector	`./install.sh opcua`

Application	Command
Public REST APIs	`./install.sh public-restapi`
Server-to-Server collector	`./install.sh s2s`
Tuner	`./install.sh tuner`
Web Admin	`./install.sh webadmin`
Web Admin UAA	`./install.sh webadmin-uaa`

Note:
If you try to install an application again, the following error messages may appear; ignore them:

- `cp: cannot start '<file name>': No such file or directory`
- `rm: cannot remove '<file name>': No such file or directory`
- `unzip: cannot find or open <file name>`

- The applications are installed. A Docker component is installed for each application that you have installed.

  Tip:
  To verify that the Docker images are installed, run the following command: `docker images`. A list of Docker images for the applications that you have installed appears.

- The `proficy-historian-linux-2.3.0.zip` file provided with the installation package is extracted to a folder named `application-bundles`. This folder contains the .zip files required to configure each application.

  Note:
  A backup of the `proficy-historian-linux-2.3.0.zip` file is located in the `/opt/historian-linux-backup` folder.

5. Run the applications that you have installed.

Application	Command
Historian database and the REST query service	`./run.sh historian`
MQTT collector	`./run.sh mqtt`

Application	Command
OPCUA DA collector	`./run.sh opcua`
Public REST APIs	`./run.sh public-restapi`
Server-to-Server collector	`./run.sh s2s`
Tuner	`./run.sh tuner`
Web Admin	`./run.sh webadmin`
Web Admin UAA	`./run.sh webadmin-uaa`

The Docker components for the applications are started.

Tip:

- To verify that the Docker components are running, run the following command: `docker ps`. A list of Docker components that are running appears.
- To verify the logs of a Docker component, run the following command: `docker logs <ID of the Docker container>`

6. Extract the .zip file for each application that you have installed by running the following command: `unzip <file name>`
7. Set the environment variables for each application that you have installed:
   - Historian database *(on page 24)*
   - REST query *(on page 34)*
   - Web Admin *(on page 42)*
   - Tuner *(on page 52)*
   - Server-to-Server collector *(on page 73)*
   - MQTT collector *(on page 48)*
   - Public REST APIs *(on page 33)*
   - OPCUA DA collector *(on page 82)*

Important:

For OPCUA DA, Server-to-Server, and Server-to-Cloud collectors, ensure that you have set the environment variables correctly before applying the configuration changes. This is because after you have applied the configuration changes for a collector, you cannot set the environment variables. If, however, you must change the values of the environment variables after the applying the changes, you must change the InterfaceName and

DefaultTagPrefix values. When you do so, a new instance of the collector is created. As a result, you must add the tags again.

8. Delete each .zip file that you have extracted by running the following command: `rm <name of the zip file>`
9. Compress the contents of each .zip file that you have extracted by running the following command: `zip <name of the .zip file (same as the original one)> <contents of the .zip file separated by a space>` (for example, `zip proficy-hisotiran-linux-amd64-2.3.0.zip historian-archiver-conf.json historian-license rest-query-config.json`).
10. **Optional:** Delete the contents of each .zip file by running the following command: `rm <file name>`. Since you have compressed these files, they are not required in the extracted folder.
11. As needed, migrate Historian data from Windows to Linux *(on page 23)*.
12. Apply the configuration changes for each application that you have installed.

Application	Command
Historian database and the REST query service	`./apply-config.sh historian-linux`
MQTT collector	`./apply-config.sh mqtt`
OPCUA DA collector	`./apply-config.sh opcua`
Public REST APIs	`./apply-config.sh public-restapi`
Server-to-Server collector	`./apply-config.sh s2s`
Tuner	`./apply-config.sh tuner`
Web Admin	`./apply-config.sh webadmin`
Web Admin UAA	`./apply-config.sh webadmin-uaa`

Stop an Application Deployed on a Generic Linux Distribution

You can stop an application if you want to modify the configuration for the application. For example, if you want to modify the properties of tags used by the server-to-server collector, you must first stop the application, modify the properties, and then run the application.

Depending on the application that you want to stop, run the command as provided in the following table.

Application	Command
Historian database and the REST query service	`./stop.sh historian`
MQTT collector	`./stop.sh mqtt`

Application	Command
OPCUA DA collector	`./stop.sh opcua`
Public REST APIs	`./stop.sh public-restapi`
Server-to-Server collector	`./stop.sh s2s`
Tuner	`./stop.sh tuner`
Web Admin	`./stop.sh webadmin`
Web Admin UAA	`./stop.sh webadmin-uaa`

The application is stopped.

Upgrade an Application on a Generic Linux Distribution

1. Stop the application that you want to upgrade.
2. Set up the application again *(on page 17)*.

Uninstall an Application Installed on a Generic Linux Distribution

When you uninstall an application, the Docker image used by the application is deleted. However, the data and log files of the application are not deleted.

Depending on the application that you want to uninstall, run the command as provided in the following table.

Application	Command
Historian database	`./uninstall.sh historian`
REST query service	`./uninstall.sh query-service`
MQTT collector	`./uninstall.sh mqtt`
OPCUA DA collector	`./uninstall.sh opcua`
Public REST APIs	`./uninstall.sh public-restapi`
Server-to-Server collector	`./uninstall.sh s2s`
Tuner	`./uninstall.sh tuner`
Web Admin	`./uninstall.sh webadmin`

Application	Command
Web Admin UAA	`./uninstall.sh webadmin-uaa`

The application is uninstalled.

Chapter 5. Historian Database

Overview of the Historian Database

The Historian database is a database of native timeseries data. It is a TCP/IP server, which uses the port number 14000 by default. It contains the following types of files:

- **.iha files:** Proprietary binary files, which contain the archived timeseries data.
- **.ihc files:** Proprietary binary files, which contain metadata. These files store information about tags and properties of collectors, data stores, and archive files.

The folder that contains the .iha and .ihc files is volume-mounted on the host file system so that data remains persistent.

You can set up the Historian database on Predix Edge *(on page 14)* or a generic Linux distribution *(on page 17)* such as Ubuntu or CentOS.

You can migrate data *(on page 23)* from Historian for Windows to Historian for Linux.

Important:
The Historian container must have a valid license file to access all the features, including high tag count support. If a valid license is not provided, Historian for Linux switches to demo mode (which supports only 32 tags) while starting the Docker container. Use the `HS_LICENSE_FILE_PATH` environment variable in the `historian-archiver-conf.json` file to provide the absolute file path of the valid license file.

Migrating Historian Data from Windows to Linux

You can migrate data and metadata from an existing Windows-based Historian to Historian for Linux.

- Set up the Historian database on Predix Edge *(on page 14)* or a generic Linux distribution *(on page 17)* such as Ubuntu or CentOS.
- Place all the Windows-based .iha and .ihc files in the archive path.

1. Set the `HS_MODE_OF_OPERATION` environment variable in the `historian-archiver-conf.json` file to `reload`.
2. Rename the .ihc file to <host name of the Docker container>_Config.

If the name of the .ihc file from the Windows machine is `WIN-2BLPS4FOACM_Config.ihc` and the host name of the Historian database Docker container is machine-01, the .ihc file should be renamed `machine-01_Config.ihc`.

Note:
There is no need to rename the .iha files.

3. Start the Historian database Docker container.

Note:
There is no need to set the `HS_MODE_OF_OPERATION` environment variable again when you restart the Docker container subsequently.

Environment Variables Used by the Historian Database

The environment variables used by the Historian database are available in the `historian-archiver-conf.json` file. The following table describes these variables.

Note:
Before applying changes to the environment variables, ensure that the number of archive files do not exceed 1024. Otherwise, the archiver will crash. This is because 1024 is the default number of file descriptors a process can open on Linux. We recommend that you create archive files daily or by size so that you can monitor the number of archive files created.

Environment Variable	Description	Default Value	Valid Values
HS_ARCHIVER_CREATE_-TYPE	The type of the archive file creation.	Days	• BySize: An archive file of a speci-fied size is created each time the size of the file reaches a specified limit.

Environment Variable	Description	Default Value	Valid Values
			• Days: An archive file is created for the duration specified in the HS_-ARCHIVE_-DURATION_-IN_DAYS variable. • Hours: An archive file is created for the duration specified in the HS_-ARCHIVE_-DURATION_-IN_HOURS variable.
HS_DEFAULT_CYCLIC_-ARCHIVING	Indicates whether data must be overwritten after a specified duration. If the value of this variable is true, data is overwritten after the duration specified in the HS_CYCLIC_ARCHIVE_DURATION_HOURS variable. This is used for the SCADA buffer data store. If the value of this variable is true, the default data store is set to SCADA buffer.	false	• true • false
HS_CYCLIC_ARCHIVE_DU-RATION_HOURS	The duration, in hours, after which cyclic archiving (data overwrite) begins. A value is required if the value of the HS_DEFAULT_CYCLIC_ARCHIVING variable is true.	8760	0 to 8760

Environment Variable	Description	Default Value	Valid Values
HS_ARCHIVE_SIZE_IN_MB	The size limit, in MB, of an archive file. After an archive reaches the specified size, a new archive file is created. A value is required if the value of the HS_ARCHIVER_CREATE_TYPE variable is BySize. This is used during start-up or creation of a data store.	100	1 to 99999
HS_ARCHIVE_DURATION_IN_HOURS	The number of hours an archive file is used for archiving data. After this duration, a new archive file is created. A value is required if the value of the HS_ARCHIVER_CREATE_TYPE variable is Hours. This is used during start-up or creation of a data store.	1	1 to 90 * 24
HS_ARCHIVE_DURATION_IN_DAYS	The number of days an archive file is used for archiving data. After this duration, a new archive file is created. A value is required if the value of the HS_ARCHIVER_CREATE_TYPE variable is Daily. This is used during start-up or creation of a data store.	1	1 to 1440
HS_FREE_SPACE_REQUIRED_IN_MB	Defines the free space, in MB, required for the archiver to work. This is used during start-up or creation of a data store. **Tip:** Set this value, in MB, to be at least five times the integral multiple of the archive size.	500	1 to 999999999
HS_USE_ARCHIVE_CACHING	Indicates whether data must be cached. When caching is enabled, when data queries are requested, they	true	• true • false

Environment Variable	Description	Default Value	Valid Values
	are cached according to the system RAM size available in the main memory. The cache is released if the RAM is used within a certain limit. This helps querying of data faster for future requests. This is used during start-up or creation of a data store.		
HS_CREATE_OFFLINE_ARCHIVE	Indicates whether to allow writing of past data in an archive file until January 1, 1970.	true	• true • false
HS_ARCHIVE_ACTIVE_HOURS	The number of hours an archive file was used to write data. This is used during start-up or creation of a data store.	8760	1 to hours till January 1, 1970
HS_MODE_OF_OPERATION	Indicates whether .ihc and .iha files must be loaded from a different Historian database (can be from a Windows-based Historian as well). If you set the value to `reload`, .ihc and .iha files from the Historian database running on one machine (Windows or Linux) is loaded to the Historian database on another machine.	normal	• normal • reload
HS_ALLOW_HELD_VALUE_QUERY	Indicates whether the held sample must be queried when Archive compression is enabled.	false	• true • false
HS_LICENSE_FILE_PATH	The absolute path of the license file of Historian for Linux. **Important:** The Historian database container must have a valid license file for all the enabled	`/config/historin-license`	

Environment Variable	Description	Default Value	Valid Values
	features, including high tag count support. If a valid license is not provided, Historian for Linux switches to demo mode (which supports only 32 tags) while starting the Docker container. For example, if you mount a `/data/edgedata` directory with `/data/` with the Historian database container, you can keep the license file in `/data/edgedata/historian-license` on the host machine. But you should set the variable to `/data/historian-license` because in a Docker-container context, the path is `/data/historian-license`.		
HS_NUMBER_OF_LOG_FILES	The maximum number of log files to be created. Once this value exceeds, the oldest file will be deleted to accommodate the new one.	100	1 to 100
HS_SIZE_OF_EACH_LOG_FILE	The maximum size of a single log file, in MB. If this value exceeds, a new log file will be created.	10	1 to 10
debug	Indicates whether debug logs must be enabled.	off	• on • off

About Array Tags

Proficy historian for edge from version 2.3.1 supports array tags. The functionality of array tags is same as On-prem Historian.

You can store a set of values with a single timestamp and single quality and then read the elements individually or as an array.The following conditions apply when using an array tag:

- You need not specify the size of an array tag. Data Archiver will store the number of elements that were written.
- You can change a tag to an array tag later as well. However, when you do so, only the latest data is retrieved. If you want to get the old data, you must change the tag back to its previous type.
- The maximum number of elements that an array tag can store is 10,000.
- You cannot associate an enumerated set or a user-defined data type (UDT) with an array tag.
- Fixed String and Scaled data types are not supported.
- Scaling, collector compression, and archive compression do not apply to an array tag.
- You cannot use an array element as a calculation trigger.
- You cannot plot a trend chart for an array tag.
- TagStats calculation mode is not supported.

Note:
Note that currently Proficy Historian for Edge does not support array tags for variable string data type.

Chapter 6. Historian Public REST APIs

Overview of the Public REST APIs

Historian for Linux is a data archiving system designed to collect, store, and retrieve time-based information efficiently at an extremely high speed. The Historian for Linux environment provides a set of REST APIs to query data from the archives. The APIs use a Docker container on a Linux machine. They use the port number 9090 for REST client requests.

Security and Authentication

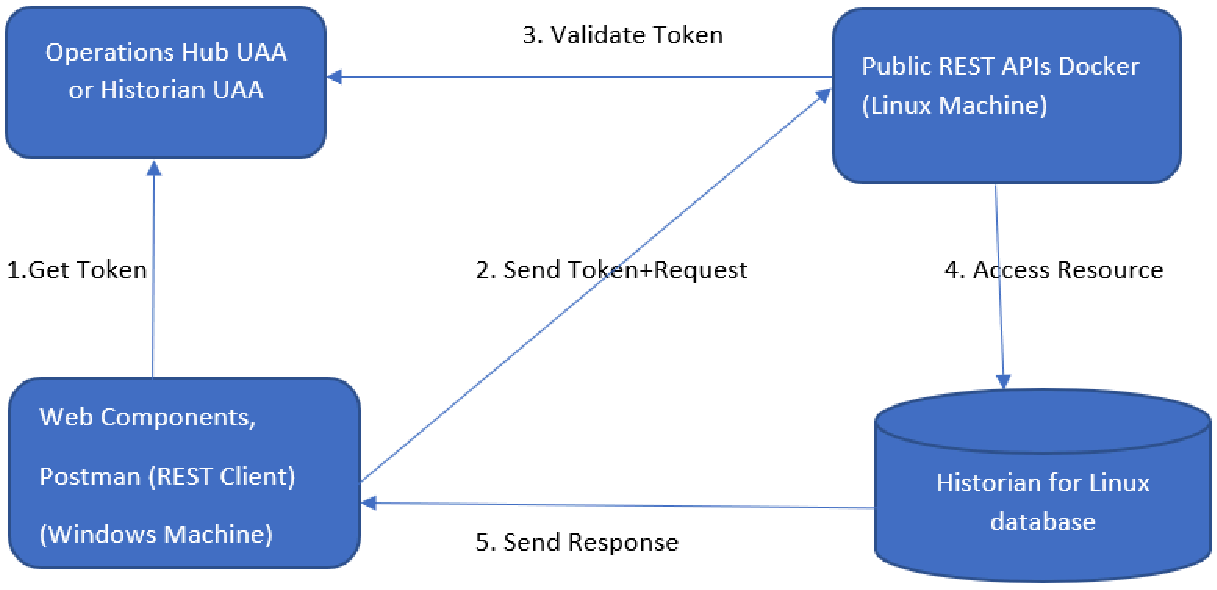
Public REST APIs support both Historian UAA and Operations Hub UAA for authentication and authorization.

Limitations

- UAA provisioning capabilities are not supported. Therefore, for Historian UAA and Operations Hub UAA, the REST APIs rely on an existing client ID to generate an authentication token. This implies that to query the REST APIs, an installation of Historian for Windows Web Clients, pointing to the same Historian server, is required to generate a UAA client ID.
- In the case of Historian for Windows public REST APIs, the tagslist API returns a paginated list of tags in Historian. It creates an index of the tags, and stores it in the PostgreSQL database, which is installed when you install Web Clients. In the case of Historian for Linux public REST APIs, the indexing service is not implemented. Therefore, to use the tagslist API, you must install the Historian for Windows Web Clients, pointing to the same Historian, to use the indexing service.

Data Flow Diagram

The following diagram shows the data flow for authorization of a data-access request using a REST API. As shown in the diagram, the REST API depends on the client ID created by Historian for Windows for generating a token.


Operations Hub UAA or Historian UAA
3. Validate Token
Public REST APIs Docker (Linux Machine)
1.Get Token
2. Send Token+Request
4. Access Resource
Web Components, Postman (REST Client) (Windows Machine)
Historian for Linux database
5. Send Response

Getting Started with the REST APIs

1. Set up the public REST APIs on Predix Edge *(on page 14)* or a generic Linux distribution *(on page 17)* such as Ubuntu or CentOS.
2. Use the `uaacert.pem` file to establish a connection between the UAA server and the REST API client. *(on page 31)*
3. Configure PostgreSQL to accept external connections *(on page 32)*. This step is required if you want to use the tagslist API.

You are now ready to use the REST APIs *(on page 33)*.

Connect to an External UAA Server

After you set up the public REST APIs on Predix Edge or on a generic Linux distribution, you must connect the UAA server with the public REST APIs Docker image. You can use the UAA service that is provided with Historian for Windows or Operations Hub.

Note:
This feature is not implemented for Predix UAA.

1. Access the `/opt/historian/docker-compose-public-restapi.yml` file.
2. Add the extra hosts as shown in the following code sample, and save the file:

```
version: '3.0'

services:

  historian-rest-api:

    hostname: historian-rest-api

    image: "dtr.predix.io/predix-edge/ge-historian-linux-public-restapi-amd64:ubuntu16.04.v2.3.0"

    ports:

    - 9090:8080

    volumes:

    - /data/public-restapi/:/data/

    - /config/public-restapi/:/config/

    extra_hosts:

    -  "VMHISTMONO:10.181.213.95"

networks:

    default:
```

Note:
Modify this file carefully. Any extra space or change in indentation can impact the Docker functionality.

3. Run the Docker Image.
4. On the UAA machine, run the Certificate Management tool.
5. Select **External Trust**, and then import the `uaacert.pem` file.
    - For Predix Edge, this file is located in the `/var/lib/edge-agent/app/<application name>/data/` folder.
    - For a generic Linux distribution, this file is located in the `/data/<application name>/` folder.
6. When prompted to restart GeOphubMasterStarter, select **No**.
7. Restart the GE Historian Tomcat service.

Configure PostgreSQL to Accept External Connections

If you want to use the tagslist API, you must set up PostgreSQL and the indexing service, which are installed when you install the web-based clients provided with Historian for Windows. This topic describes how to configure PostgreSQL to accept external connections, which is required for you to use the tagslist API.

1. Log in to the machine on which the web-based clients are installed.
2. Access the following folder: `<web-based clients installation drive>:\ProgramData\GE\Operations Hub\historian-postgres\data`
3. Access the `pg_hba.conf` file in a text editor.
4. In the IPV4 local connections section, add a line to enable PostgreSQL to trust connections from the local network.

   `host all all 10.0.0.0/8 trust`. In this example, the local network IP address begins with 10. Similarly, you must enter the first 8-bit value of the IP addresses in your local network.
5. Save the file.
6. Access the `postgresql.conf` file (located in the same folder) in a text editor.
7. In the Connection Settings section, modify the listen_addresses key to `listen_addresses='*'`
8. Save the file.
9. Restart the GE Historian PostgreSQL Database service.

Environment Variables Used by the Public REST APIs

The environment variables used by the public REST APIs are available in the `historian-linux-public-restapi-config.json` file. The following table describes these variables.

Environment Variable	Description
UAA_SERVER_MACHINE_IP	The IP address of the Historian UAA or the Operations Hub UAA server. A value is required.
HISTORIAN_HOSTNAME	The IP address of the Historian for Linux server. A value is required.
HRA_UAA_SCHEME_AND_SERVER	The UAA server scheme. Enter a value in the following format: https://<host name of the UAA server>. A value is required.
POSTGRES_MACHINE_IP	The IP address of the machine on which PostgreSQL is installed (via the web-based clients). A value is required if you want to use the tagslist API.

Sample REST URI

The public REST APIs use the port 9090 for client connections. The format of the URI is as follows:

```
http://<IP address of the Historian for Linux machine>:9090/historian-rest-api/v1/<API name>
```

For information on the REST APIs, refer to Historian APIs.

Chapter 7. Historian REST Query Service

Overview of the REST Query

The REST query contains APIs to fetch data from the Historian database. You can fetch data such as the latest data point of a tag or data points for a duration. Using these APIs, you can also work on aggregation techniques like average, minimum, and maximum values. The REST query container exposes port number 8989 for querying the data. These REST APIs are exactly the same as Predix Time Series REST APIs. For more information, see the Predix Time Series service documentation at https://docs.predix.io/en-US/content/service/data_management/time_series/ and the Time Series service API documentation at https://www.predix.io/api.

The Docker image for the Historian database container is bundled with the REST query. You can set up the REST query on Predix Edge *(on page 14)* or a generic Linux distribution *(on page 17)* such as Ubuntu or CentOS.

Environment Variables Used by the REST Query

The environment variables used by the REST query are available in the `rest-query-config` file. The following table describes these variables.

Note:
For a secure connection, add - zones.<zone-id>.query, historian_rest_query_service.user in the UAA scopes and authorities.

Environment Variable	Description	Default Value	Valid Values
HISTORIAN_HOSTNAME	The IP address of the Historian database.		
HISTORIAN_MAX_DATA_QUERY	The maximum number of data points to be retrieved for one tag from the Historian database.	10000	As many tags as you have in the Historian archiver.
HISTORIAN_MAX_TAG_QUERY	The maximum number of tags retrieved from the Historian database.	5000	

Environment Variable	Description	Default Value	Valid Values
DISABLE_REST_QUERY_SECURITY	Indicates whether security must be enabled.	true	• false: The service runs in a secure mode. • true: The service runs in an unsecured mode.
ZAC_UAA_CLIENTID	Client ID of OAUTH2 server. A value is required if the DISABLE_REST_QUERY_SECURITY variable is set to false.		
ZAC_UAA_CLIENT_SECRET	Client secret of the OAUTH2 server. A value is required if the DISABLE_REST_QUERY_SECURITY variable is set to false.		
ZAC_UAA_ENDPOINT	URL of OAUTH2 server. A value is required if the DISABLE_REST_QUERY_SECURITY variable is set to false.		
USE_PROXY	Indicates whether a firewall is present between the OAUTH2 server and the REST query.	true	• false • true
PROXYURL	The URL of the proxy server (along with the port number).		

REST Query Array Tag

REST Query Array Tag now supports array type to get the data samples. It also supports Raw By Number, Raw By Time Sampling modes.

Task	URL	Method	Request Body	Request Headers

Get values for an array tag	http://<IP address of the Linux machine>:8989/v1/datapoints	POST	`{`
			 `"start": "4h-ago",`
			 `"end": "1h-ago",`
			 `"tags": [{`
			 `"name": "G790J9Y2E.SimulationArray00001",`
			 `"limit": 1000,`
			 `"order": "desc",`
			 `"filters": {`
			 `"qualities": {`
			 `"values": [`
			 `"3"`
			 `]`
			 `}`
			 `}`
			 `}]`
			 `}`
				Predix-Zone-Id Can be any value. Not validated.

Below are the examples for Raw By Number, Raw By Time with Request Body.

```
Raw By Time Payload
{
    "start": "4h-ago",
    "end": "1h-ago",
    "tags": [{
        "name": "G790J9Y2E.SimulationArray00001",
        "limit": 1000,
        "order": "desc",
        "filters": {
            "qualities": {
                "values": [
                    "3"
                ]
            }
        }
    }]
}
```

```
Raw By Number
{
    "start": "4h-ago",
    "tags": [{
        "name": "G790J9Y2E.SimulationArray00001",
        "filters": {
            "qualities": {
                "values": [
                    "3"
                ]
            }
        }
    }]
}
```

```
Raw By Number with limit value
{
    "start": "4h-ago",
    "tags": [{
        "name": "G790J9Y2E.SimulationArray00001",
        "limit": 10,
        "filters": {
            "qualities": {
                "values": [
                    "3"
                ]
            }
        }
    }]
}
```

```
Raw By Number with direction backword

{

 "start": "4h-ago",

 "direction": "backward",

        "tags": [{

  "name": "G790J9Y2E.SimulationArray00001",

     "limit": 3

 }]

}
```

Sample Response for Array Tag:

```
{


  "tags": [


       {


           "name": "G790J9Y2E.SimulationArray00001",


           "results": [


               {


                   "groups": [


                       {


                           "name": "type",


                           "type": "array"


                       }


                   ],


                   "filters": {
```

```
        "qualities": {

            "values": [

                "3"

            ]

        }

    },

    "values": [

       [

            1686637046000,

           "[\"3902\"]",

            3

        ],    [

            1686637227000,

           "[\"1769\",\"1769\",\"1769\",\"1769\",\"1769\",\"1769\",\"1769\",\"1769\"]",

            3

        ],

        [

            1686637228000,

           "[\"28289\",\"28289\",\"28289\",\"28289\",\"28289\",\"28289\"]",
                            3

                        ],

                        [

                            1686637235000,

                           "[\"2943\",\"2943\"]",

                            3

                        ],

                    "attributes": {}

                }

            ],

            "stats": {

                "rawCount": 4

            }

        }

   ]

}
```

Example of the REST Query API

The REST query container exposes port 8989 for querying the data that was read from the Historian database. For information on the list of APIs and their usage, refer to the Predix Time Series service API documentation.

Note:
The API /v1/datapoints/latest/v2 is not supported by the REST query.

1. Create a query.

   The header of the query must be in the following format:

   ```
   Headers:

       Authorization: Bearer <token from a trusted issuer>

       Predix-Zone-Id: <tenant>

       Content-Type: application/json
   ```

   The following table provides the URL and other required values for each task that you can perform using the REST query.

Task	URL	Method	Request Body	Request Headers
Get values for a tag	http://<IP address of the Linux machine>:8989/v1/datapoints	POST	{ “start”: “1h-ago”, “tags”: [{ “name”: “”, “order”: “desc” }] }	Predix-Zone-Id Can be any value. Not validated.
Get values for a tag	http://<IP address of the Linux machine>:8989/v1/datapoints?query= { “start”: “1h-ago”, “tags”: [{ “name”: “”, “order”: “desc” }] }	GET		Predix-Zone-Id Can be any value. Not validated.
Get all tags	http://<IP address of the Linux machine>:8989/v1/tags	GET	None	Predix-Zone-Id Can be any value. Not validated.

2. Retrieve a token from the OAUTH2 server by using the following REST API:

   ```
   curl -u <Client-Id>:<Client-secret> https://<IP address of the Predix Edge OS device>:8080/uaa/oauth/token -d
    'grant_type=client_credentials'
   ```

   ```
   curl -u edgeclient:edgesecret https://10.10.10.10:8080/uaa/oauth/token -d 'grant_type=client_credentials'
   ```

Chapter 8. Historian Web Admin Service

Overview of Web Admin

The web admin is a user interface that allows you to monitor and control the Historian archiver.

You can set up web admin on Predix Edge *(on page 14)* or a generic Linux distribution *(on page 17)* such as Ubuntu or CentOS.

You can use a User Authentication and Authorization (UAA) service with web admin *(on page 43)*.

For more information, you can refer https://www.ge.com/digital/documentation/historian/version2023/c_wac_WebAdminConsole.html to understand the overall operations.

Environment Variables Used by Web Admin

The environment variables used by the web admin are available in the `historian-webadmin-config.json` file in the `proficy-historian-linux-webadmin-amd64-2.3.0` folder. The following table describes these variables. If, however, you want to use web admin with UAA, set the environment variables used by the UAA service *(on page 44)*.

Environment Variable	Description	Default Value	Valid Values
HISTORIAN_HOSTNAME	The IP address of the Historian database.		
HISTORIAN_MAX_TAG_QUERY	The maximum number of data points to be retrieved for one tag from the Historian database.	10000	As many tags as you have in the Historian archiver.
HWA_ADMIN_USERNAME	The username set by the Docker administrator while running the web admin Dock-		

Environment Variable	Description	Default Value	Valid Values
	er image. A value is required.		
HWA_ADMIN_PASSWORD	The password set by the Docker administrator while running the web admin Docker image. A value is required.		

Accessing the Web Admin

In a web browser, enter the URL for the web admin.
https://<ip_address>:9443/historian-visualization/hwa
The wed admin console appears.

About Using UAA with Web Admin

You can use web admin with or without UAA. Using UAA with web admin provides more security. You can use Predix UAA *(on page 43)* or any other UAA application *(on page 44)* with web admin.

Use Predix UAA with Web Admin

This topic describes how to use Predix UAA with web admin. You can, however, choose to use any other UAA service *(on page 44)* with web admin.

1. Create a client with the following details:
    - Under **Authorization Grant Types**, select the **authorization_code** and **refresh_token** check boxes.
    - In the **Scopes** and **Authorities** boxes, enter uaa.none,historian_visualization,openid,uaa.resource,historian_tuner.admin. If you want to use the REST Query application securely, add: - zones.<zone-id>.query, historian_rest_query_service.user
    - In the **Redirect URI** box, enter https://<machine ip>:9443/historian-visualization/login.
2. Create a user in the Predix UAA instance.
3. Set the environment variables of the UAA service *(on page 44)*.

4. Restart the Webadmin-uaa Docker image.
5. Log in to the following URL with the user credentials that you have created: `https://<IP address of the machine on which the web admin Docker image is running>:9443/historian-visualization/hwa`

   The web admin home page appears.

Use Other UAA Service with Web Admin

This topic describes how to use an external UAA with web admin application. You can, however, choose to use Predix UAA *(on page 43)* with web admin.

1. Create a client with the following scopes and authorities:
   - uaa.none
   - historian_visualization
   - openid
   - uaa.resource
   - historian_tuner.admin (required if you want to use tuner securely)
   - - zones.<zone-id>.query, historian_rest_query_service.user (required if you want to use the REST query securely)
2. Create a user in the UAA instance.
3. Set the environment variables of the UAA service *(on page 44)*.
4. Restart the Webadmin-uaa Docker image.

Environment Variables Used by Web Admin UAA

The environment variables used by the web admin UAA are available in the `historian-webadmin-config.json` file in the `proficy-historian-linux-webadmin-uaa-amd64-2.3.0` folder. The following table describes these variables. If, however, you do not want to use UAA with web admin, set the environment variables used by web admin *(on page 42)*.

Environment Variable	Description	Default Value	Valid Values
HISTORIAN_HOSTNAME	The IP address of the Historian database.		
HISTORIAN_MAX_TAG_QUERY	The maximum number of data points to be re-	10000	As many tags as you have in the Historian archiver.

Environment Variable	Description	Default Value	Valid Values
	trieved for one tag from the Historian database.		
HV_UAA_CLIENT_ID	The client ID of the UAA service.		
HV_UAA_CLIENT_SECRET	The client secret of the UAA service.		
HV_UAA_SCHEME_AND_SERVER	The UAA server scheme. Enter a value in the following format: https://<host name of the UAA server>. A value is required.		
HV_USE_PROXY	Indicates whether the UAA service uses a proxy server.	true	• true • false
HV_PROXY_USERNAME	The username of the proxy server. A value is required if proxy authentication is used.		
HV_PROXY_PASSWORD	The password of the proxy server. A value is required if proxy authentication is used.		

Environment Variable	Description	Default Value	Valid Values
HV_PROXY_URL	The URL to access the proxy server. A value is required if the UAA service uses a proxy server.		
HV_SKIP_SSL_VALIDATION	Indicates whether SSL validation must be skipped.	true	• true • false

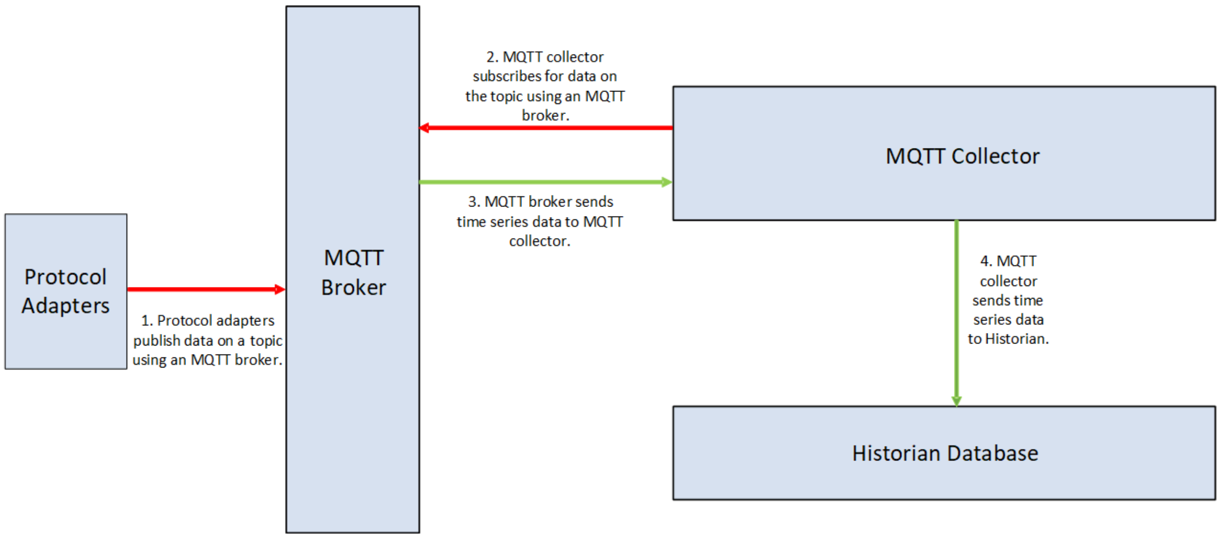
Chapter 9. Historian MQTT Collector

Overview of the MQTT Collector

The Historian MQTT collector collects Predix time series data, and sends it to the Historian database. It works as follows:

1. Connects to an MQTT broker, and subscribes to a topic. In this case, the data bus of Predix Edge serves as the MQTT broker.
2. Collects the data, which is in the Predix time series (JSON) format.
3. Adds relevant tags to the Historian database based on the data received.
4. Sends data to the Historian database.

The following diagram illustrates how the MQTT collector collects and sends data. The red lines indicate the initial, one-time steps. The green lines indicate the steps performed every time data is sent.


Protocol Adapters
1. Protocol adapters publish data on a topic using an MQTT broker.
MQTT Broker
2. MQTT collector subscribes for data on the topic using an MQTT broker.
3. MQTT broker sends time series data to MQTT collector.
MQTT Collector
4. MQTT collector sends time series data to Historian.
Historian Database

With latest enhancements, now we are supporting array datatype tags along with primitive data types. The following is the sample for sending the array data:

```
{"body":[{"attributes":{"machine_type":"machine1"},"datapoints":[[1686040732424,[1645,4177,5674,3504],3]],
 "name":"NewArrayTag1"}],"messageId":"Simulate"}
```

The bolded text shows how to pass array type and the format followed is datapoints:[[timestamp, value or [array of values], quality]].

Limitations

- Linux MQTT collector does not have the facility to provide the port number. If the broker is not running on the port 1883, the collector will not be able to connect with the source.
- Linux MQTT collector does not have the facility to provide the user name and password based authentication.
- Linux MQTT collector does not have the facility to provide the certificate based authentication.
- Spark plug-B format is not supported.
- Only MQTT V311 is supported.
- Message Retention option is not supported.
- Custom QoS is not supported.

Environment Variables Used by the MQTT Collector

The environment variables used by the MQTT collector are available in the `historian-mqtt-collector-config.json` file. The following table describes these variables.

Parameter	Description	Default Value	Valid Values
DebugMode	The debug mode for the MQTT collector.	00	• 00: Indicates that debugging is disabled. • ff: Indicates that debugging is enabled.
HistorianNodeName	The host name of the Historian server to which you want to send data using the MQTT collector. A value is required.		

Parameter	Description	Default Value	Valid Values
InterfaceName	The name of the MQTT collector. You must not change this value.	EdgeMQTT	
DefaultTagPrefix	The default prefix for the tags created by the MQTT collector. For example, if you enter mqtt, for a tag named pressure, a tag with the following name is created: mqtt.pressure	EdgeMQTT	
Hostname	The host name of the machine on which you the MQTT broker is running. A value is required.		
Topic	The topic to which you want the MQTT collector to subscribe. A value is required. You can subscribe to multiple topics. For example, if you subscribe to Home/#, you will receive messages published to the child topics as well (such as Home/FirstFloor, Home/SecondFloor).	historian_data	
HS_NUMBER_OF_LOG_FILES	The maximum number of log files to be created. Once this value exceeds, the oldest file will	100	1 to 100

Parameter	Description	Default Value	Valid Values
	be deleted to accommodate the new one.		
HS_SIZE_OF_EACH_LOG_FILE	The maximum size of a single log file, in MB. If this value exceeds, a new log file will be created.	10	1 to 10

Note:
You cannot define tags in the JSON file. Tags are automatically created based on the data collected by the MQTT collector.

Chapter 10. Historian Tuner

Overview of Tuner

Tuner is an application for designing and automating the administrative tasks of the Historian database. You can perform tasks such as:

- Changing the tag properties
- Changing the data store properties
- Creating a back-up of archive files
- Restoring the backed-up archive files
- Purging data

You can perform these tasks using the web admin as well. However, using tuner, you can provide inputs using a JSON file, which allows you to automate these tasks. You can also perform these tasks repeatedly at regular intervals.

You can set up tuner on Predix Edge *(on page 14)* or a generic Linux distribution *(on page 17)* such as Ubuntu or CentOS.

Tuner exposes REST APIs to upload the JSON file. For example, if you want to back up data of the last seven days, you can create a JSON file with the following content:

```
 { "Historian Node": "10.181.213.175",

  "Data Management": {

    "Back Up": [

     {

        "Datastore Name": "User",

        "Back Up Path":"/data/backup",

        "Properties":

        {

         "Number Of Files":7

        }

       }

    ]

  }

}
```

Environment Variables Used by Tuner

The environment variables used by Tuner are available in the `historian-tuner-config.json` file. The following table describes these variables.

Note:
For a secure connection, add historian_tuner.admin in the UAA scopes and authorities.

Environment Variables	Description	Default value	Valid Values
HS_LOG_TO_FILE	Indicates whether logs must be redirected to a file. If set to false, the logs appear in the Docker console.	false	• true • false
HS_NUMBER_OF_LOG_FILES	The maximum number of log files to be created. Once this value exceeds, the oldest file will be deleted to accommodate the new one.	100	1 to 100
HS_SIZE_OF_EACH_LOG_FILE_IN_MB	The maximum size of a single log file, in MB. If this value exceeds, a new log file will be created.	10	1 to 10
TUNER_SECURE	Indicates whether security must be enabled.	true	• true • false
OAUTH2_CLIENT_ID	The client ID of an OAUTH2 server. A value is required if the TUNER_SECURE variable is set to true.		
OAUTH2_CLIENT_SECRET	The client secret of an OAUTH2 server. A value is required if the TUNER_-		

Environment Variables	Description	Default value	Valid Values
	SECURE variable is set to true.		
OAUTH2_URL	The URL of an OAUTH2 server. A value is required if the TUNER_SECURE variable is set to true.		
https_proxy	The URL of a proxy server (along with the port number).		

Use Tuner on Predix Edge

1. Create a JSON file with the required details. For a sample file, refer to JSON File Content Example *(on page 66)*.

   Important:
   The file name must begin with historian.

2. If you want to use tuner securely:

   a. Generate a bearer token.

      Tip:
      To generate a token, you can use POSTMAN or run the following command in an Ubuntu machine: `curl -u <username>:<password> -d 'grant_type=client_credentials' https://<predix uaa instance>/bearer/token`

   b. Run the following command: `curl -H 'Authorization:Bearer <token>' -F 'uploadFile=@<absolute path of the JSON file>' http://<IP address of the Predix Edge device:9000/upload`
      `– curl -H 'Authorization:Bearer h390ufwehqef39vwnf4wehwef09rf' -F 'uploadFile=@ C:\workstation\historian-config.json' http://10.181.212.287:9000/upload`

3. If you do not want to use tuner securely, run the following command: `curl -F 'uploadFile=@<absolute path of the JSON file>' http://<IP address of the Predix Edge device:9000/upload`
   `– curl -F 'uploadFile=@ C:\workstation\historian-config.json' http://10.181.212.287:9000/upload`

Use Tuner on a Generic Linux Distribution

1. Create a JSON file with the required details. For a sample file, refer to JSON File Content Example *(on page 66)*.

   Important:
   The file name must begin with historian.

2. Place the JSON file in the `/data/tuner/incoming` folder.

Examples of Tasks You can Perform Using Tuner

The following section provides a list of tasks that you can perform using tuner, along with a sample JSON code for each task.

To Create a Data Store

Sample JSON code

```
{
  "Historian Node": "10.181.213.175",
  "Data Management": {
    "Create Datastore": [
      {
        "Datastore Name": "Turbine-4",
        "Properties": {
          "Default Datastore": true,
          "Description": "Custom datastore for storing data of Turbine-1"
        }
      }
    ]
  }
}
```

JSON Key description

- **Datastore Name**: Can be a sequence of characters surrounded with ".
- **Default Datastore**: Enter true to set a data store as the default one.

What can you do with the operation?

Create the default data store. You can also create multiple data stores by providing proper details in the JSON file.

Purging a Data Store

Sample JSON code

```
{
                "Historian Node": "10.181.213.175",
                "Data Management": {
                "Purge": [ { "Datastore Name": "Turbine-4" } ]
                }
                }
```

JSON Key description

- **Datastore Name**: Can be a sequence of characters surrounded with ".

What can you do with the operation?

Delete Turbine-4 from your system.

Purging Archives based on Archive Name

Sample JSON code

```
                {
                "Historian Node": "10.181.213.175",
                "Data Management": {
                "Purge": [
                {
                "Datastore Name": "Turbine-10",
                "Properties": {
                "Archive File Names": [
                "Turbine-10_historian-archiver_Archive046.iha",
                "Turbine-10_historian-archiver_Archive1543363199.iha"
                ]
                }
                }
                ]
                }
                }
```

JSON Key description

- **Data store name**: Can be a sequence of characters surrounded with ".
- **Archive File Name**: Can be a sequence of characters surrounded with ".

What can you do with the operation?

Delete Turbine-10_historian-archiver_Archive046.iha and Turbine-10_historian-archiver_Archive1543363199.iha

Purging Archives based on Time stamps

Sample JSON code

```
{
  "Historian Node": "10.181.213.175",
  "Data Management": {
    "Purge": [
      {
        "Datastore Name": "User",
        "Properties": {
          "Start Time": 1543417800,
          "End Time": 1543418220
        }
      }
    ]
  }
}
```

JSON Key description

- **Data store name**: Can be a sequence of characters surrounded with ".
- **Start Time/End Time**: Must be in epoch time format, in seconds.

  What can you do with the operation?

  Delete the data between the given timestamps. This will delete entire archives with/between these timestamps.

Backup of Archive files using File Names

Sample JSON code

```
{
  "Historian Node": "10.181.213.175",
  "Data Management": {
    "Back Up": [
      {
        "Datastore Name": "User",
        "Back Up Path": "/data/",
        "Properties": {
          "Archive File Names": [
            "User_historian-archiver_Archive1543449599"
          ]
        }
      }
    ]
  }
}
```

JSON Key description

- **Data store name**: Can be a sequence of characters surrounded with ".
- **Back-Up Path**: Must be a valid path in context of the Historian docker container.

  Note:
  The Back Up Path parameter must always be set to `/data/`. However, the backup is created in the `/data/database` folder.

- **Archive file name**: Must be valid archive names. You can provide multiple archives separating with comma.

What can you do with the operation?

This will back up the provided archive file to the `data/backup` folder.

Backup of Archive Files using Number of Files

Sample JSON code

```
{
  "Historian Node": "10.181.213.175",
  "Data Management": {
    "Back Up": [
     {
        "Datastore Name": "User",
        "Back Up Path":"/data/",
        "Properties":
        {
         "Number Of Files":2
        }
       }
    ]
  }
}
```

JSON Key description

- **Data store name**: Can be a sequence of characters surrounded with ".
- **Back-Up path**: Must be a valid path in the context of the archiver docker container.

  Note:
  The Back Up Path parameter must always be set to `/data/`. However, the backup is created in the `/data/database` folder.

- **Number of files**: Number of files to be backed up. Should be a numerical value.

What can you do with the operation?

Back up the lat two archive files to the backup folder.

Backup of Archive Files using Start time and End Time

Sample JSON code

```
{
  "Historian Node": "10.181.213.175",
  "Data Management": {
    "Back Up": [
   {
        "Datastore Name": "User",
        "Back Up Path":"/data/",
        "Properties":
        {
         "Start Time" :1540511999,
         "End Time"      :1540598399
        }
       }
    ]
  }
}
```

JSON Key description

- **Data store name**: Can be a sequence of characters surrounded with ".
- **Backup path**: Must be a valid path in the context of archiver docker container.

> **Note:**
> The Back Up Path parameter must always be set to `/data/`. However, the backup is created in the `/data/database` folder.

- **Start/End Time:** Must be an epoch timestamp.

What can you do with the operation?

Back up the data between the given timestamps. This will backup entire archives with/between the timestamps.

Restore

Sample JSON code

```
{
  "Historian Node": "10.181.213.175",
  "Data Management": {
    "Restore": [
      {
        "File Path": "/data/User_historian-archiver_Archive1543507756_Backup.zip",
        "Archive Name": "User_historian-archiver_Archive1543507756",
        "Datastore Name": "User"
      }
    ]
  }
}
```

JSON Key description

- **File Path**: Path of the backed-up file.

> **Note:**
> The File Path parameter must always be set to `/data/<name of the archive file>`. However, the archive file is located in the `/data/database/` folder.

- **Archive Name**: Name to which data to be restored.
- **Data store Name**: Name of the data store for which the archive file must be restored.

What can you do with the operation?

Restore the backed-up files into specific data store.

Data Store options for Archive Type Hours/Days

```
"Datastore Name": "ScadaBuffer",
    "Properties": {
     "Archive Type": "Hours",
     "Archive Duration": 10,
     "Archive Active Hours": 10,
     "Archive Default Backup Path": "/data/archiver/backupfiles/",
     "Datastore Duration": 4
    }
```

JSON Key description

- **Archive type**: Valid values are Hours, Days, and BySize.
- **Archive duration**: Must be a numerical Value.
- **Archive active hours**: Must be a numerical Value.
- **Archive default backup path**: Must be a valid path.
- **Data store duration**: Must be a numerical value.

What can you do with the operation?

Set the data store properties as mentioned in the configuration file.

Data Store options for Archive Type BySize

Sample JSON code

```
"Datastore Name": "DHSSystem",

    "Properties": {

     "Archive Type": "BySize",

     "Archive Default Size(MB)": 200,

     "Archive Active Hours": 744,

     "Archive Default Backup Path": "/data/archiver/backupfiles/"

    }
```

JSON Key description

- **Archive Default Size(MB)**: Must be a numerical value. Rest keys can be referred from the preceding examples.

What can you do with the operation?

Set the data store properties as mentioned in the configuration file for the archive type BySize.

Tag Options-Collection Properties

Sample JSON file

```
{

  "Historian Node": "10.181.213.175",

  "Config": {

    "Tag Options": [

      {

        "Tag Pattern": "US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Byte",

        "Tag Properties": {
          "Collection": {
            "Collection": true,
            "Collection Interval Unit": "sec",
            "Collection Interval": 5,
            "Collection Offset Unit": "sec",
            "Collection Offset": 1,
            "Time Resolution": "sec"
          }
        }
      }
    ]
  }
}
```

JSON Key description

- **Collection**: Must be true/false.
- **Collection Interval Unit:** Must be sec, min, hour, or millisec
- **Collection Offset Unit**: Must be sec or millisec.
- **Collection Interval and Collection Offset**: Must be a numerical value.

Note:
You can filter tags based on the tag names, collector name, and data store name. To do so, replace Tag Pattern with Collector Name or Datastore Name.

What can you do with the operation?

Set the tag properties as mentioned in the configuration file.

Tag Options-Compression Properties

```
{
  "Historian Node": "10.181.213.175",
  "Config": {
    "Tag Options": [
      {
        "Tag Pattern": "US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Byte",
        "Tag Properties": {
          "Collector Compression": {
            "Collector Compression": true,

            "Collector Deadband": "Percent Range",

            "Collector Deadband Value": 80,

            "Collector Compression Timeout Resolution": "min",

            "Collector Compression Timeout Value": 10

          }

        }

      }

    ]

  }

}
```

JSON Key description

- **Collector Compression**: Must be true or false.
- **Collector Deadband Value/Collector Compression Timeout Value**: Must be a numerical value.
- **Collector Deadband**: Must be Percent Range or Absolute.
- **Collector Compression Timeout Resolution**: Must be sec, min, hour, or millisec.

Note:
You can filter tags based on the tag names, collector name, and data store name. To do so, replace Tag Pattern with Collector Name or Datastore Name.

What can you do with the operation?

Set the compression properties as mentioned in the configuration file.

Tag Options-Scaling

Sample JSON code

```
{

  "Historian Node": "10.181.213.175",

  "Config": {

    "Tag Options": [

      {

        "Tag Pattern": "US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Byte",

        "Tag Properties": {

          "Scaling": {
            "Hi Engineering Units": 100,
            "Low Engineering Units": 0,
            "Input Scaling": false,
            "Hi Scale Value": 0,
            "Low Scale Value": 0
          }
        }
      }
    ]
  }
}
```

Note:
You can filter tags based on the tag names, collector name, and data store name. To do so, replace Tag Pattern with Collector Name or Datastore Name.

What can you do with the operation?

Set the scaling properties as mentioned in the configuration file.

Tag Options-Condition Based Collection

Sample JSON code

```
{
  "Historian Node": "10.181.213.175",
  "Config": {
    "Tag Options": [
      {
        "Tag Pattern": "US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Byte",
        "Tag Properties": {
          "Condition Based Collection": {
            "Condition Based": true,
            "Trigger Tag": "US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Boolean",
            "Comparison": ">=",
            "Compare Value": "50000",
            "End Of Collection Marker": true
          }
        }
      }
    ]
  }
}
```

JSON Key description

- **Trigger Tag**: Must be a valid tag name.
- **Comparison**: =,<,<=,>,>=,!=
- **End of Collection Marker**: true or false

Note:
You can filter tags based on the tag names, collector name, and data store name. To do so, replace Tag Pattern with Collector Name or Datastore Name.

What can you do with the operation?

Set the condition-based collection properties as mentioned in the configuration file.

Tag Options- Using Tag Group

Sample JSON code

```
{
  "Historian Node": "10.181.213.175",
  "Config": {
    "Tag Options": [
      {
        "Tag Group": [
          "US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Boolean",
          "US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Byte"
        ],
        "Tag Properties": {
          "Tag Datastore": "ScadaBuffer",
          "Data Type": "Int16"
        }
      }
    ]
  }
}
```

JSON Key description

- **Tag Group:** Must be a valid tag name. You can provide any number of tags.

What can you do with the operation?

Set the tag properties to the group of tags mentioned in the Tag Group section.

JSON File Content Example

```
{
  "Historian Node": "10.181.212.175",
  "Data Management": {
    "Create Datastore": [
      {
        "Datastore Name": "Turbine-1",
        "Properties": {
          "Default Datastore": false,
          "Description": "Custom datastore for storing data of Turbine-1"
        }
      },
      {
        "Datastore Name": "Turbine-2",
        "Properties": {
          "Default Datastore": true,
          "Description": "Custom datastore for storing data of Turbine-2"
        }
      }
    ],
    "Back Up": [
      {
        "Datastore Name": "User",
        "Back Up Path": "/data/",
        "Properties": {
          "Archive File Names": [
            "User_historian-archiver_Archive1540511999",
            "User_historian-archiver_Archive1540598399"
          ]
        }
      },
      {
        "Datastore Name": "User",
        "Back Up Path": "/data/",
        "Properties": {
          "Number Of Files": 2
        }
      },
      {
        "Datastore Name": "User",
        "Back Up Path": "/data/",
        "Properties": {
          "Start Time": 1540511999,
          "End Time": 1540598399
        }
      }
    ],
    "Purge": [
      {
        "Datastore Name": "User",
        "Properties": {
          "Archive File Names": [
            "User_historian-archiver_Archive1540511999",
            "User_historian-archiver_Archive1540598399"
          ]
        }
      },
      {
        "Datastore Name": "Turbine-1"
      },
      {
        "Datastore Name": "User",
        "Properties": {
          "Start Time": 1540511999,
          "End Time": 1540598399
        }
      }
    ],
    "Restore": [
      {
        "File Path": "/data/User_historian-archiver_Archive1540511999_Backup.zip",
        "Archive Name": "User_historian-archiver_Archive1540511999",
        "Datastore Name": "User"
      },
      {
        "File Path": "/data/User_historian-archiver_Archive1540598399_Backup.zip",
        "Archive Name": "User_historian-archiver_Archive1540598399",
        "Datastore Name": "User"
      }
    ]
  },
  "Config": {
    "Datastore Options": [
      {
        "Datastore Name": "ScadaBuffer",
        "Properties": {
          "Archive Type": "Days",
          "Archive Duration": 1,
          "Archive Active Hours": 99999,
          "Archive Default Archive Name": "ScadaBuffer_historian-archiver_Archive",
          "Archive Default Backup Path": "/data/archiver/archives/",
          "Default Datastore": true,
          "Datastore Duration": 48
        }
      },
      {
        "Datastore Name": "User",
        "Properties": {
          "Archive Type": "Hours",
          "Archive Duration": 1,
          "Archive Active Hours": 744,
          "Automatically Create Archives": false,
          "Overwrite Old Archives": true,
          "Archive Default Backup Path": "/data/archiver/archives/"
        }
      },
      {
        "Datastore Name": "DS1",
        "Properties": {
          "Archive Type": "BySize",
          "Archive Default Size(MB)": 100,
          "Archive Active Hours": 744,
          "Archive Default Backup Path": "<path>"
        }
      }
    ],
    "Tag Options": [
      {
        "Tag Group": [ "Test-Boolean", "Test-Int16" ],
        "Tag Properties": {
          "Tag Datastore": "ScadaBuffer",
          "Data Type": "Int16"
        }
      },
      {
        "Tag Pattern": "Demo.Dynamic.Scalar.*",
        "Tag Properties": {
          "Collection": {
            "Collection": true,
            "Collection Interval Unit": "min",
            "Collection Interval": 10,
            "Collection Offset Unit": "sec",
            "Collection Offset": 0,
            "Time Resolution": "sec"
          },
          "Condition Based Collection": {
            "Condition Based": true,
            "Trigger Tag": "SampleTrigger",
            "Comparison": ">=",
            "Compare Value": "50000",
            "End Of Collection Marker": true
          },
          "Collector Compression": {
            "Collector Compression": true,
            "Collector Deadband": "Percent Range",
            "Collector Deadband Value": 80,
            "Collector Compression Timeout Resolution": "min",
            "Collector Compression Timeout Value": 10
          },
          "Archive Compression": {
            "Archive Compression": true,
            "Archive Deadband": "Percent Range",
            "Archive Deadband Value": 80,
            "Archive Compression Timeout Resolution": "min",
            "Archive Compression Timeout Value": 10
          },
          "Scaling": {
            "Hi Engineering Units": 1000,
            "Low Engineering Units": 0,
            "Input Scaling": false,
            "Hi Scale Value": 0,
            "Low Scale Value": 0
          },
          "Tag Datastore": "DS1"
        }
      },
      {
        "Collector Name": "EdgeMQTT",
        "Tag Properties": {
          "Collection": {
            "Collection": true,
            "Collection Interval Unit": "sec",
            "Collection Interval": 10,
            "Collection Offset Unit": "sec",
            "Collection Offset": 0,
            "Time Resolution": "sec"
          },
          "Archive Compression": {
            "Archive Compression": true,
            "Archive Deadband": "Absolute",
            "Archive Deadband Value": 5,
            "Archive Compression Timeout Resolution": "min",
            "Archive Compression Timeout Value": 15
          },
          "Tag Datastore": "TestDS"
        }
      },
      {
        "Datastore Name": "DS1",
        "Tag Properties": {
          "Collection": {
            "Collection": true,
            "Collection Interval Unit": "min",
            "Collection Interval": 10,
            "Collection Offset Unit": "sec",
            "Collection Offset": "0",
            "Time Resolution": "sec"
          },
          "Condition Based Collection": {
            "Condition Based": true,
            "Trigger Tag": "SampleTrigger",
            "Comparison": ">=",
            "Compare Value": "50000",
            "End of Collection Marker": true
          },
          "Collector Compression": {
            "Collector Compression": true,
            "Collector Deadband": "Percent Range",
            "Collector Deadband Value": 80,
            "Collector Compression Timeout Resolution": "min",
            "Collector Compression Timeout Value": 10
          },
          "Archive Compression": {
            "Archive Compression": true,

            "Archive Deadband": "Percent Range",

            "Archive Deadband Value": 80,

            "Archive Compression Timeout Resolution": "min",

            "Archive Compression Timeout Value": 10

          },

          "Tag Datastore": "DS1"

        }

      }

    ]

  }

}
```

Chapter 11. Historian Server-to-Server Collector

Overview of the Server-to-Server Collector

The Historian server-to-server collector allows you to collect data and messages from a source Historian server to a destination on-premises Historian server or Predix Cloud.

GE recommends installing the server-to-server collector in the source Historian machine. Doing so enables the store and forward capabilities of data collectors to preserve collected data even if the collector and destination archiver are disconnected. The server-to-server collector can also run as a stand-alone component where both the source and destination Historian databases are on remote machines.

You can set up the server-to-server collector on Predix Edge *(on page 14)* or a generic Linux distribution *(on page 17)* such as Ubuntu or CentOS.

Environment Variables Used by the Server-to-Server Collector

The environment variables used by the server-to-server collector are available in the `historian-s2s-collector-config.json` file. The following table describes these variables.

Important:
For OPCUA DA, Server-to-Server, and Server-to-Cloud collectors, ensure that you have set the environment variables correctly before applying the configuration changes. This is because after you have applied the configuration changes for a collector, you cannot set the environment variables. If, however, you must change the values of the environment variables after the applying the changes, you must change the InterfaceName and DefaultTagPrefix values. When you do so, a new instance of the collector is created. As a result, you must add the tags again.

Environment Variable	Description	Default Value	Valid Values
HistorianNodeName	The host name of the Historian server to which you want to send data using the server-to-server collector. A value is required. You can choose to send data to an server		

Environment Variable	Description	Default Value	Valid Values
	that is on-premises or on Predix cloud.		
InterfaceName	The name of the server-to-server collector.	historian-s2c	
DefaultTagPrefix	The default prefix for the tags created by the server-to-server collector. For example, if you enter s2s, for a tag named pressure, a tag with the following name is created: s2s.pressure	historian-s2c.	
General3	The IP address of the Historian archiver.		
OfflineTagConfigurationFile	The absolute path to the offline tag configuration file.	`/config/tag-list.xml`	
HS_NUMBER_OF_LOG_FILES	Maximum number of Log Files. Once this value exceeds, the oldest file will be deleted to accommodate new one. Its default value is 100, and range is from 1 to 100.	100	1 to 100
HS_SIZE_OF_EACH_LOG_FILE	Maximum size of one Log file in Mega Bytes. If this value exceeds, new Log File will be created.	10	1 to 10

Important notes on the Server-to-Server Collector Tasks

When you add a tag by choosing from the list of tags in the server-to-server collector, only certain tag properties are copied from the source tag to the destination tag. If you intend to copy raw samples from the source to the destination, after you add the tag, ensure that you set these properties to the required values. Refer to the following table.

Important tag properties that are not automatically copied when you add the tag include:

- **Input scaling settings:** If you are using input scaling, since the output of the source tag is the input to the destination tag, you may want to match the EGU limits on the source to input limits on the destination, if you are using Input Scaling.
- **Timestamp resolution:** Make sure that the timestamp resolution properties match. For example, do not use the second timestamp resolution on the destination tag, if your source tag uses millisecond timestamp resolution. If your source tag uses millisecond timestamp resolution, then you also want to set your destination tag to also use millisecond timestamp resolution.

The following table describes the tag properties in the Historian Administrator Tags screen that are copied when the destination tag is created via select from the browse. If a property is not listed in this table, it is not copied.

Tab Name	Properties Copied
General	Description EGUDescription
Collection	Data Type DataLength
Scaling	HiEGU LoEGU InputScaling HiScale LoScale
Compression	ArchiveCompression

Tab Name	Properties Copied
	ArchiveDead-band(%)

Streaming data to Predix Time Series

The server-to-server collector needs a list of tags in the form of an .xml file. Data of these tags will be streamed to Predix Time Series by the collector. This is called offline configuration of tags.

Offline configuration also helps you define the configuration properties of a collector (such as the list of tags, tag properties, and collector interface properties). This feature is useful when collectors connect to Predix TimeSeries.

Ensure that the path to the offline tag configuration file is correct in the `historian-s2s-collector-config.json` file.

Create the Offline Configuration File

This topic describes how to create the offline configuration file. We recommend that you add the Collector property section above the Tag property section in the offline configuration file.

1. Add the following collector interface properties at the beginning of the offline configuration file.

```
<Import>

<Collectors>

<Collector Name="<Collector Name>">

<InterfaceType>ServerToServer</InterfaceType>

<InterfaceGeneral1>10</InterfaceGeneral1>

......

</Collector>

</Collectors>
```

2. Add the list of tags and their properties.

```
<Collectors>

...

</Collectors>


<TagList Version="1.0.71">


<Tag>
```

```
<Tagname>simCollector1</Tagname>

<SourceAddress>Result = CurrentValue("SJC1GEIP05.Simulation00002")</SourceAddress>

...

</Tag>


<Tag>

<Tagname>simCollector2</Tagname>

<SourceAddress>Result = CurrentValue("SJC1GEIP05.Simulation00002")</SourceAddress>

...

</Tag>

...

</TagList>

</Import>
```

3. Add the closing `</Import>` tag to the end of offline configuration file.

Sample OfflineConfiguration.xml file

```
<Import>

  <TagList Version="1.0.71">

    <Tag>

      <Tagname>HistS2SInt16</Tagname>

      <SourceAddress>Result = CurrentValue("US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Int16")</SourceAddress>

      <DataType>SingleInteger</DataType>

      <CollectionType>Unsolicited</CollectionType>

      <TimeResolution>Milliseconds</TimeResolution>

      <CollectionInterval>1000</CollectionInterval>

      <Description>US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Int16</Description>

      <HiEngineeringUnits>200000.00</HiEngineeringUnits>

      <InputScaling>false</InputScaling>

      <InterfaceCompression>1</InterfaceCompression>

      <InterfaceDeadbandPercentRange>80.00</InterfaceDeadbandPercentRange>

      <InterfaceCompressionTimeout>30000</InterfaceCompressionTimeout>t>

      <TimeStampType>Source</TimeStampType>

      <CalculationDependencies>

        <CalculationDependency>US-TestTagsChange1.Objects.Demo.Dynamic.Scalar.Int16</CalculationDependency>

      </CalculationDependencies>

      <SpikeLogic>1</SpikeLogic>

      <LoScale>0</LoScale>
```

```
       <HiScale>32767.00</HiScale>

       <NumberOfElements>0</NumberOfElements>

     </Tag>

  </TagList>

</Import>
```

Predix Time Series Information Fields in the `historian-s2s-collector-config.json` File

The environment variables to set the Predix Time Series ingestion URL and Predix UAA credentials are available in the `historian-s2s-collector-config.json` file. The following table describes these variables. For information on the values to provide, contact your Predix Time Series administrator.

Field	Description
Cloud Destination Address	The URL of a data-streaming endpoint exposed by the Predix Time Series instance to which the data should go. Typically, it starts with wss://.
Identity Issuer	The URL of an authentication endpoint for the collector to authenticate itself and acquire necessary credentials to stream to the Predix Time Series. Typically, it starts with https:// and ends with /oauth/token.
Client ID	This variable identifies the collector when interacting with the Predix Time Series. This is equivalent to the username in many authentication schemes. The client must exist in the UAA identified by the identity issuer, and the system requires that the timeseries.zones. {ZoneId}.ingest and timeseries.zones.{ZoneId}.query authorities are granted to the client for the Predix Zone ID specified.
Client Secret	The secret to authenticate the collector. This is equivalent to Password in many authentication schemes.
Zone ID	The unique instance to which the collector will stream data among the many instances of the Time Series service that the Predix system hosts.

Field	Description
Proxy	The URL of the proxy server to be used for both the authentication process and for streaming data. A value is required if the collector is running on a network where proxy servers are used to access web resources outside of the network. However, it does not affect the proxy server used by Windows when establishing secure connections. As a result, you must still properly configure the proxy settings for the Windows user account under which the collector service runs.

Chapter 12. Historian OPCUA DA Collectors

Overview of the OPCUA DA Collector

The OPCUA DA collector collects data from any OPCUA-compliant OPCUA DA server, and sends it to the Historian database. The collector automatically determines the capability of the OPCUA server to which it is connected. For more information on the capabilities of the collector, refer to Capabilities of the OPCUA DA Collector *(on page 80)*.

You can set up the OPCUA DA collector on Predix Edge *(on page 14)* or a generic Linux distribution *(on page 17)* such as Ubuntu or CentOS.

Capabilities of the OPCUA DA Collector

The following table outlines the capabilities of the Historian OPCUA DA Collector.

Feature	Capability
Browse Source For Tags	Yes (on OPCUA servers that support browsing)
Browse Source For Tag Attributes	Yes
Polled Collection	Yes
Minimum Poll Interval	100 ms
Unsolicited Collection	Yes. If you are using an OPCUA DA collector with unsolicited data collection and the collector compression is disabled, new values must produce an exception. When the OPCUA DA collector performs unsolicited collection, the deadband percentage is determined by the collector deadband percent. You can only configure the collector deadband percent by enabling compression.
Timestamp Resolution	100ms
Accept Device Timestamps	Yes
Floating Point Data	Yes
Integer Data	Yes
String Data	Yes
Binary Data	Yes

Feature	Capability
Python Expression Tags	No

Note:
You must set Time Assigned by field to Source if you have unsolicited tags getting data from an Historian OPCUA DA Collector.

Tag Attributes Available on Browse

The following table outlines the tag attributes available when browsing.

Attribute	Capability
Tag name	Yes
Source Address	Yes
Engineering Unit Description	Yes, varies by OPCUA Server Vendor.
Data Type	Yes. See Selecting Data Types.
Hi Engineering Units	Yes, varies by OPCUA Server Vendor.
Lo Engineering Units	Yes, varies by OPCUA Server Vendor.
Hi Scale	Yes
Lo Scale	Yes
Is Array Tag	No

Note:
While some of these attributes are queried on a browse, they are not shown in the browse interface. These attributes are used when adding a tag, but you will not be able to see whether or not all attributes come from the server.

Selecting Data Types

The following table lists the data types recommended for use with Historian.

OPCUA Data Type	Recommended Data Type in Historian
I1 - 16 bit signed integer	Single Integer

OPCUA Data Type	Recommended Data Type in Historian
I4 - 32 bit signed integer	Double Integer
R4 - 32 bit float	Single Float
R8 - 64 bit double float	Double Float
UI2 - 16 bit unsigned single integer	Unsigned Single Integer
UI4 - 32 bit unsigned double integer	Unsigned Double Integer
UI8 - 64 bit unsigned quad integer	Unsigned Quad Integer
I8 - 64 bit quad integer	Quad Integer
BSTR	Variable String
BOOL	Boolean
I1 - 8 bit single integer	Byte

Note:
The Historian OPCUA Collector requests data from the OPCUA DA server in the native data type. Then the Historian OPCUA DA collector converts the received value to a Historian Data Type before sending it to the Historian database.

OPCUA Group Creation

To increase performance, it is recommended that you limit the number of OPCUA groups created by the Historian system. To do this, consider grouping Historian tags (collected by the Historian OPCUA DA Collector) using the least amount of collection intervals possible.

Environment Variables Used by the OPCUA DA Collector

The environment variables used by the OPCUA DA collector are available in the `opcua-collector-config.json` file. The following table describes these values.

Important:
For OPCUA DA, Server-to-Server, and Server-to-Cloud collectors, ensure that you have set the environment variables correctly before applying the configuration changes. This is because after you have applied the configuration changes for a collector, you cannot set the environment variables. If, however, you must change the values of the environment variables after the applying

the changes, you must change the InterfaceName and DefaultTagPrefix values. When you do so, a new instance of the collector is created. As a result, you must add the tags again.

Environment Variable	Description	Default Value	Valid Values
DebugMode	The debug mode for the OPCUA DA collector.	00	• 00: Indicates that debugging is disabled. • ff: Indicates that debugging is enabled.
HistorianNodeName	The host name or IP address of the destination Historian machine.		
InterfaceName	The name of the OPCUA DA collector. A value is required and must be unique.	Historian-opcua-collector-200	
DefaultTagPrefix	The default prefix for the tags created by the OPCUA DA collector. For example, if you enter opcua, for a tag named pressure, a tag with the following name is created: opcua.pressure.	historian-opcua.	

Environment Variable	Description	Default Value	Valid Values
General1	The URL of the OPCUA server.		
General2	Identifies whether to enable secure connection.	true	• true • false
HS_NUMBER_OF_LOG_FILES	The maximum number of log files to be created. Once this value exceeds, the oldest file will be deleted to accommodate the new one.	100	1 to 100
HS_SIZE_OF_EACH_LOG_FILE	The maximum size of a single log file, in MB. If this value exceeds, a new log file will be created.	10	1 to 10

Sample `ClientConfig.ini` File Used by the OPCUA DA Collector

```
[UaClientConfig]

ApplicationName =OPCUACollector



; TrustCertificate value (only used in secured connection):

; 0 (No trust),

; 1 (Trust temporarily)

; 2 (Default, trust permanently and copy the server certificate into the client trust list)

TrustCertificate =2

CertificateTrustListLocation      =/data/pkiclient/trusted/certs/

CertificateRevocationListLocation =/data/pkiclient/trusted/crl/

IssuersCertificatesLocation       =/data/pkiclient/issuers/certs/

IssuersRevocationListLocation     =/data/pkiclient/issuers/crl/

ClientCertificate                 =/data/pkiclient/own/certs/domain.der

ClientPrivateKey                  =/data/pkiclient/own/private/domain.key

RetryInitialConnect               =true

AutomaticReconnect                =true
```

Note:

- If you do not provide the certificate and key pair, the collector generates one.
- If the General2 variable in the `opcua-collector-config.json` file is set to false, the collector connects to the OPCUA DA server in the unsecured mode (without any certificate exchange).
- The `RetryInitialConnect` parameter is used to retry connecting to the OPCUA DA server when the collector fails to connect to the server initially. The `AutomaticReconnect` parameter is used to retry connecting to the OPCUA DA server when the collector fails to connect to the server subsequently.

Secured OPCUA Collector Connectivity

When secured connectivity is enabled between the OPCUA DA server and collector, you must add the client certificate to the trusted list of certificates of the OPCUA DA server.

1. Start the OPCUA DA collector in secured mode. The collector will not start.
2. Locate the OPCUA DA local or remote server installation location on the machine where the OPCUA server is running. Typically, this can be found under `C:\ProgramFiles\OPCUA Server Name\` in Windows Host.
3. Locate the folder named `rejected` in the OPCUA DA server installation folder. If you are unable to locate it, check the OPCUA DA server manual for assistance.
4. The client certificate is in the `rejected` folder. Copy and paste this certificate into the trusted list of certificates of the OPCUA server. The OPCUA DA server manual provides the folder where the trusted certificates are located.
5. Restart the OPCUA DA collector.

Chapter 13. Security for Historian for Linux container Ecosystem

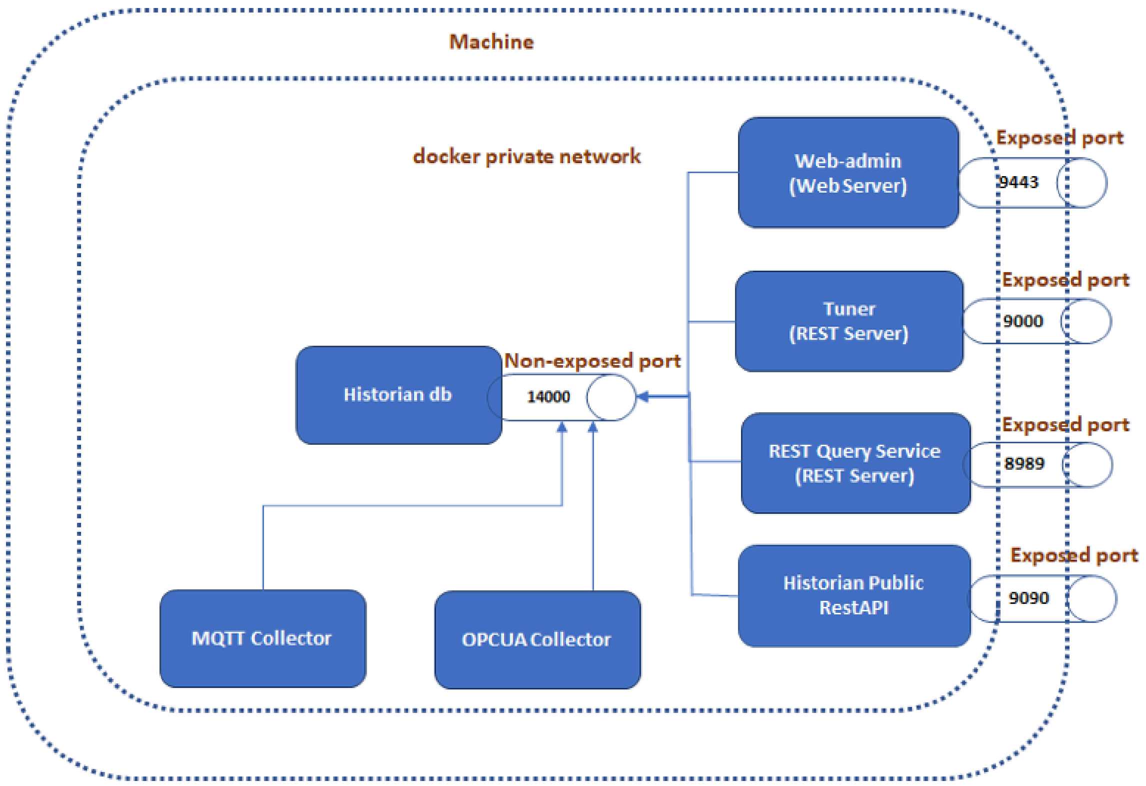
Security for Historian for Linux container Ecosystem

This section describes the security mechanism for the Historian for Linux container ecosystem. The main objective here is to protect the Historian database, which is the custodian of data. The security is implemented in two tiers:

- **Tier 1- Docker private network**

  Docker private network is a technology that enables a group of Docker containers to perform network communication with one another. The ports on which applications of this groups are listening is available for view-only to member applications of this Docker private network. If any Docker container wants to expose its port outside of the Docker private network, that port must be exposed.

  The following diagram shows the network ports on which various Historian containers are listening.



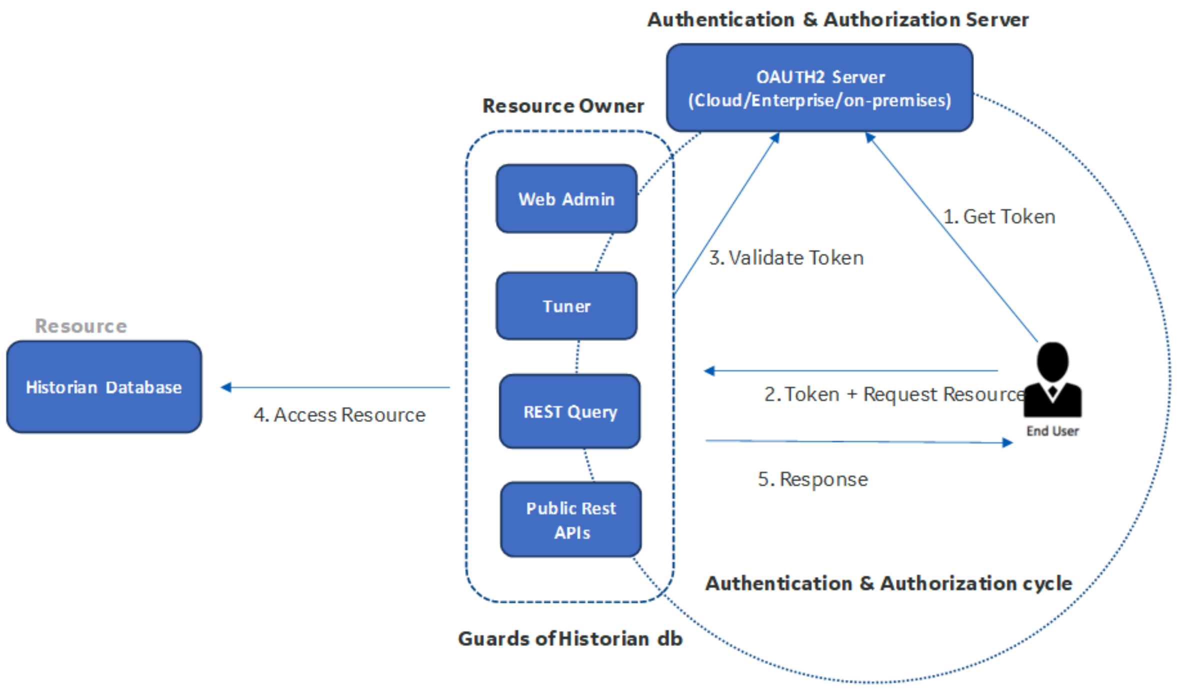
As shown in the diagram, the ports 9443, 9090, 9000 and 8989 are exposed to outside of docker private network. So, the clients of web admin, tuner, REST query, and public REST APIs can interact with these applications either from outside of the machine or from Docker private network.

The port 14000 is not exposed if you are using Predix Edge. If, however, you are using a generic Linux distribution, the port 14000 is exposed. This port (TCP/IP port) is secured via Docker private network. The members of this network like tuner, web admin, REST query service, public REST APIs, and collectors (MQTT and OPCUA) can only connect to port 14000 of the Historian database.

- **Tier 2 (OAUTH2 mechanism)**

  In tier 1, the ports 9443, 9090, 9000, and 8989 are not protected. If, however, you want to protect these ports, the REST query, tuner, web admin, and the public REST APIs can use the OAUTH2 authentication and authorization mechanism.

  The following diagram shows the network ports that use the OAUTH2 authentication and authorization mechanism.



As shown in the diagram, the Historian database does not use any OAUTH2 authentication and authorization mechanism directly. You interact with the Historian database using the web admin, tuner, public REST APIs, and the REST query applications.

The Historian database is the ultimate resource we want to protect (the analogy here can be to that of a vault in a bank), while web admin, tuner, public REST APIs, and the REST query act as resource owners and are guards of the Historian database (analogy with the guard of the vault in a bank).

To provision the tier 2 security mechanism, you must set up an OAUTH2 server or you can use Predix UAA (OAUTH2 server on Predix Cloud).

Note:
The public REST APIs perform authentication and authorization with Historian UAA or Operations Hub UAA.

Web admin, tuner, public REST APIs, and REST query offer Docker environment variables by which the users can provide OAUTH2 credentials to these Docker containers, so that these applications can validate the token from the specified OAUTH2 server.

The following code sample provides the environment variables to set for each application to leverage OAUTH2 authentication and authorization:

```
For Tuner

----------------------

TUNER_SECURE=true

OAUTH2_CLIENT_ID=my-uaa-client

OAUTH2_CLIENT_SECRET= my-uaa-secret

OAUTH2_URL= https://28649aab-0fd3-456c-baea-335d1b907668.predix-uaa.run.aws-usw02-pr.ice.predix.io

https_proxy=http://my-proxy.ge.com:80


For REST Query Service

----------------------

- DISABLE_REST_QUERY_SECURITY=false

- ZAC_UAA_CLIENTID=my-uaa-client

- ZAC_UAA_CLIENT_SECRET=my-uaa-secret

- ZAC_UAA_ENDPOINT=https://28649aab-0fd3-456c-baea-335d1b907668.predix-uaa.run.aws-usw02-pr.ice.predix.io

- USE_PROXY=true

- PROXYURL=http://my-proxy.com:80


For Web-admin

----------------------

HV_UAA_CLIENT_ID=my-uaa-client

HV_UAA_CLIENT_SECRET=my-uaa-secret

HV_UAA_SCHEME_AND_SERVER=https://28649aab-0fd3-456c-baea-335d1b907668.predix-uaa.run.aws-usw02-pr.ice.predix.io

HV_USE_PROXY=true

HV_PROXY_URL= http://my-proxy.com:80


For Historian Public REST API

------------------------------

"HRA_UAA_SCHEME_AND_SERVER":https://Excel-2010

"UAA_SERVER_MACHINE_IP":"10.181.214.218",
```

For information on how to set these environment variables, refer to:

- Environment Variables Used by Tuner *(on page 52)*
- Environment Variables Used by the Public REST APIs *(on page 33)*

- Environment Variables Used by Web Admin *(on page 42)*
- Environment Variables Used by the REST Query *(on page 34)*

You can also refer to the `docker-compose.yml` file provided with the Historian Docker application bundles for the aforementioned environment variables.

Chapter 14. Key differences between Historian for Linux and Historian for Windows

Key differences between Historian for Windows and Historian for Linux

The following table provides a list of the key differences between Historian for Windows and Historian for Linux:

Feature	Historian for Linux	Historian for Windows
Predix Time series style REST APIs	Yes	No
Tuner (for the configuring Historian database)	Yes	No
Array datatypes	Yes	Yes
User-defined datatypes (custom structure)	No	Yes
Enumeration datatype	No	Yes
Collector redundancy	No	Yes
Alarm and Event Archiver	No	Yes
Diagnostic Manager for detecting faulty collectors and clients	No	Yes
Proficy common licensing	No	Yes
Expose data as per OPCHDA server standards	No	Yes
Mirroring	No	Yes
Alerts and messages are not verbal	No	Yes

Collector portfolio of Historian for Windows and Historian for Linux

Important:
All collectors in Historian for Windows can connect to the Historian for Linux database and vice versa. The following table provides a list of collectors that run on Windows and/or Linux operating systems.

Collectors	Linux Host	Microsoft Windows Host
OPCUA collector	Yes	Yes
Server-to-Server collector	Yes	Yes
Server-to-Cloud collector	Yes	Yes
MQTT collector	Yes	Yes
Windows Performance collector	Not applicable	Yes
OPC collector	No	Yes
OPC Historical Data Access collector	No	Yes
Calculation collector	No	Yes
OSI PI collector	No	Yes
iFIX collector	No	Yes
CygNet collector	No	Yes
Wonderware (Schneider Electric) collector	No	Yes
File collector	No	Yes

Clients that cannot run on Linux Host

Important:
Clients listed below can connect to Linux Historian database and operate but they cannot run on Linux Host.

- Historian Administrator (VB admin console)
- ihSQL client

- Excel add-in
- Trend client

Chapter 15. Historian for Linux Client Libraries

Historian for Linux Libraries

The following libraries are available in Artifactory. Use the following information to ensure that you can access the files.

For Predix Users

To access Artifactory links, you must first create an account on predix.io. Your predix.io account sign-in credentials will be used to access Artifactory.

Collector Toolkit

The Collector toolkit is a C++-based library you can use to write custom Historian collectors. You must write the source code to handle interactions with respective sources (for example, OPCUA, Modbus, and so on). Collector interaction with Historian and features like store and forward and auto-reconnect to Historian is automatically handled by this library.

The library is compiled in Ubuntu 16.04 x64.

User API

The user API is a C library used for adding, deleting, and updating tags for a collector. Usually, you can manage tags using web admin, but alternatively, you can use this library to do this programatically.

The library is compiled in Ubuntu 16.04 x64.

Overview of the Collector Toolkit

Using the collector toolkit, you can write programs that integrate with Historian for Linux and leverage the same configuration tools, redundancy schemes, and health monitoring as collectors that are provided with Historian for Linux. A custom collector is a collector developed using the collector toolkit. It collects data and messages from a data source and writes them to a data archiver. Using the collector toolkit, you can create custom collectors that:

- Collect data and messages from any data source
- Perform collector compression and buffer collected data
- Report data and messages to a local or remote data archiver

Custom collectors can be developed to function much like the standard OPC and iFIX collectors that come with Historian for Linux. The collected data can be used in any application that connects to Historian for Linux.

The toolkit enables development of programs that collect data at the current time. It is not suitable for developing migration programs, file import programs, SQL import programs or other programs that produce data which has timestamps in the past. Use other Historian for Linux toolkits to accomplish these tasks.

The toolkit supports pre-processing raw data with Python Expression Tags during collection, provided that you enable this.

Historian for Linux User API - an Overview

The Historian for Linux User API is a C library intended to provide high speed read/write access to Historian data and tags. It does not provide access to archives, alarms, events, or messages.

Use the API to develop applications, which read and write data to the Historian server when the REST query and web admin do not meet the requirement for performance or programming language.

Sample programs are provided with the API, demonstrating how to perform common tasks. You can find additional information in the comments in the sample code and API header.

Note:
To use this API, you must be familiar with the Historian for Linux features and functionalities.

Supported platforms

The Historian for Linux API is compiled on Ubuntu 16.04 LTS and Alpine 3.6.

Related Documentation

Historian

- Historian Collector Toolkit *(on page 94)*
- Historian User API *(on page 95)*

Predix Time Series Service

- Predix Time Series Service
- Predix Time Series Service API Documentation

Predix UAA Service

- Predix User Account and Authentication Security Service

Chapter 16. Troubleshoot Historian for Linux

General Troubleshooting Tips

The following are general issues you may experience when using Historian for Linux.

Historian for Linux Starts in Demo Mode

Cause

The Historian for Linux database container must have a valid license file for all enabled features, including high tag count support. If a valid license is not provided Historian for Linux switches to demo mode (which supports only 32 tags) while starting the Docker container.

Solution

Use the `HS_LICENSE_FILE_PATH` environment variable in the `historian-archiver-conf.json` file to provide the absolute file path of the valid license file.

For example, if you mount a `/data/edgedata` directory with `/data/` with Historian archiver Docker, you can keep the license file in the `/data/edgedata/historian-license` folder on the host machine. But you must set the environment variable to `/data/historian-license` because in a Docker-container context, the path is `/data/historian-license`.

Web Admin Fails to Start

Cause

The web admin is a web application hosted in a Tomcat server inside a Docker container. Sometimes the host machine has its own Tomcat server running and bound to a standard Tomcat port like 8080 or 8443, so when the web admin Docker container is started, a port conflict occurs between the Tomcat server running inside the Docker container and the Tomcat server running directly on the host machine.

Solution

Stop the Tomcat service of the host machine.

As a protective measure, in the `docker-compose.yml` file of web admin, port 9443 of the host machine is mapped with port 8443 of the web admin Docker container, so that the web admin Docker does not conflict with port 8443 of any other service on the host machine.

Server-to-Server Collector fails to connect to the Destination and Source Historian

Cause

The Historian archiver Docker is a TCP/IP server, listening on port 14000. The server-to-server collector is a Historian for Linux client, which connects to port 14000 of the machine on which the Historian archiver Docker is running. Usually, we do not publish port 14000 of the Historian archiver. Because of this, the Historian for Linux client Dockers (such as the server-to-server collector) which do not join the Docker network of Historian archiver, will not be able to use the port 14000.

Solution

Publish port 14000 of the Historian archiver, if you want Historian for Linux clients to be connected to Archiver from outside of Historian Archiver's docker network.

Detailed Logs of the MQTT Collector, Server-to-Server Collector, and Historian Archiver

Cause

The Historian archiver, the MQTT collector, and the server-to-server collector logs do not go to Docker logs. Therefore, it is difficult to analyze the operations logs of these applications.

Solution

Historian archiver logs are available in the `<mounted host's data directory with docker container>/archiver/Logs` folder. The MQTT collector and server-to-server collector logs are available in the mounted host's data directory with the Docker container folder.

Connection to the OPCUA DA Server Fails

Cause

The session may have timed out when the OPCUA DA collector tried to connect to the OPCUA DA server.

Solution

Ensure that the `RetryInitialConnect` and `AutomaticReconnect` parameters are set to true in the `ClientConfig.ini` file.