The content of this document is still under validation.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting: http://www.intel.com/design/literature.htm

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at http://www.intel.com/ or from the OEM or retailer.

No computer system can be absolutely secure.

Intel and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

Code Names are only for use by Intel to identify products, platforms, programs, services, etc. ("products") in development by Intel that have not been made commercially available to the public, i.e., announced, launched or shipped. They are never to be used as "commercial" names for products. Also, they are not intended to function as trademarks.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by Intel is under license.

*Other names and brands may be claimed as the property of others.

Copyright © 2019, Intel Corporation. All rights reserved.
Contents

1.0 Introduction ........................................................................................................................................ 5
  1.1 Assumptions .................................................................................................................................. 5
  1.2 Terminology .................................................................................................................................. 6

2.0 Edge Insights Software Architecture .......................................................................................... 8
  2.1 Minimum System Requirements ................................................................................................. 8
  2.2 Architecture ................................................................................................................................ 8
  2.3 Time Series Data Ingestion ......................................................................................................... 10
  2.4 Time Series Storage .................................................................................................................... 10
     2.4.1 Configuring Influx ............................................................................................................... 10
  2.5 Closed Loop Control .................................................................................................................. 10
      StreamSubLib Class ...................................................................................................................... 11
     2.5.1 Time Series Analytics ........................................................................................................ 13
     2.5.1.1 User Defined Functions ............................................................................................... 15
     2.5.1.1.1 Interfaces Exposed for UDFs .................................................................................. 15
     2.5.1.2 Working with Kapacitor ............................................................................................... 15
  2.6 Video Data ................................................................................................................................... 16
     2.6.1 Ingesting Video Data .......................................................................................................... 16
     2.6.1.1 Basler Camera .............................................................................................................. 17
     2.6.1.2 RTSP Camera ............................................................................................................... 18
     2.6.2 USB Camera ....................................................................................................................... 18
     2.6.3 Configuring multiple cameras ............................................................................................. 19
     2.6.4 Image resizing ...................................................................................................................... 20
     2.6.5 Simultaneously storing the image to two different destinations ..................................... 21
     2.6.6 Specifying encoding options ............................................................................................. 21
     2.6.7 Using a trigger ...................................................................................................................... 21
     2.6.7.1 Trigger Algorithm Format ............................................................................................ 22
  2.7 Image Store .................................................................................................................................. 24
     2.7.1.1 Data Ingestion Library ................................................................................................. 24
     2.7.1.2 Configuring the Image store ......................................................................................... 24
  2.8 Video Analytics ........................................................................................................................... 27
     2.8.1 Developing Classifier Code in Data Analytics .................................................................. 27
     2.8.2 Algorithm configuration files .............................................................................................. 29
     2.8.3 Using Accelerators .............................................................................................................. 32
  2.9 EIS Data Agent ............................................................................................................................ 32
     2.9.1 Data Agent Service .............................................................................................................. 32
     2.9.2 Databus Abstraction Configuration .................................................................................... 34
     2.9.3 Stream Manager Configuration ........................................................................................... 34
     2.9.4 GetConfigInt Interface ...................................................................................................... 35
     2.9.5 External Library ................................................................................................................... 35
        Directory Layout ......................................................................................................................... 37
2.9.5.1 Modules.................................................................................................................. 43
2.9.5.2 Classes ................................................................................................................... 44
2.9.5.3 Class Reference....................................................................................................... 44
databus Class .................................................................................................................. 44
  GrpcImageStoreClient Class ......................................................................................... 47
2.9.6 Security .................................................................................................................... 48
  2.9.6.1 Vault..................................................................................................................... 48
  2.9.6.2 Provisioning ........................................................................................................ 49
  2.9.6.2.1 Generating Keys and Certificates .................................................................. 49
  2.9.6.2.2 Provisioning Tool .......................................................................................... 49
  2.9.6.3 Start up and Run Time ....................................................................................... 50
    2.9.6.3.1 Secure Boot .................................................................................................. 50
    2.9.6.3.2 Accessing Secrets ....................................................................................... 50
    2.9.6.4 Retiring ............................................................................................................ 51
    2.9.6.4.1 Certificate revocation ................................................................................. 51

3.0 DEV MODE and Profiling .............................................................................................. 52
  3.1 DEV MODE .................................................................................................................. 52
  3.1.1 Starting the EIS in dev mode ................................................................................... 52
  3.1.2 Security features disabled in dev mode ................................................................. 52
  3.1.3 Faster development cycle while developing classifier algorithm ....................... 52
  3.1.4 Faster development cycle while developing classifier and trigger algorithm ....... 53
  3.2 PROFILING MODE .................................................................................................. 53

4.0 Code Modification Recommendations........................................................................... 54

5.0 Visualizer (IEI Simple Visualizer) ............................................................................... 55

6.0 Working with EIS Docker Containers ......................................................................... 57
  6.1 Docker setup directory structure: ............................................................................ 57
  6.2 Compose Files .......................................................................................................... 58
  6.3 Useful Docker* commands ....................................................................................... 60

7.0 Debugging EIS .............................................................................................................. 62
  7.1 Debugging through the logs ...................................................................................... 62
  7.2 Debugging video ingestion through gst-launch ...................................................... 62
    7.2.1 GStreamer valid plugins / element .................................................................... 62
    7.2.2 Debugging the Video ingestion container, in case of ‘video_src’ getting modified by SI. ............................................................................................................. 63
  7.3 Debugging through DEV MODE ............................................................................... 64

8.0 Known Issues .................................................................................................................. 65

§
1.0 Introduction

Industrial Edge Insights Software (EIS) from Intel is a set of pre-integrated ingredients designed to accelerate the development and deployment of solutions for the industrial sector. EIS helps to address various industrial usages, which include data collection, storage, and analytics on a variety of hardware nodes that span across the factory floor.

The first release of EIS focuses on the device closest to the data generation, for example the tool or machine assembling a product. This enables the ingestion of video and time series data, data storage, performing analytics, closing the loop by transmitting a control message, and publishing the results.

This guide will detail the capabilities of Edge Insights software ingredients and will serve as a reference for developers working with them. The guide is designed and written to enable system integrators and Independent Software Vendors (ISV) develop custom solutions for their end customers.

1.1 Assumptions

This guide assumes that the user has either already set up the Edge Insights Software release by following the EIS Setup Guide document; or already has purchased a device with EIS pre-installed, and has followed the Getting Started Guide to set the system up. This user guide provides in-depth details on the architecture, libraries, and file layout.

It's assumed that any developer working with EIS has working knowledge of Python.
## 1.2 Terminology

### Table 1. General Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>DL</td>
<td>Deep Learning</td>
</tr>
<tr>
<td>ECN</td>
<td>Edge Compute Node</td>
</tr>
<tr>
<td>gRPC</td>
<td>Google Remote Procedure Call - An open source HTTP/2 based RPC mechanism</td>
</tr>
<tr>
<td>EIS</td>
<td>Industrial Edge Insights Software from Intel</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript® Object Notation - Lightweight text format for storing and transporting data</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine-to-Machine</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>MQTT</td>
<td>Message Queuing Telemetry Transport - Standard publish-subscribe-based messaging protocol</td>
</tr>
<tr>
<td>OPC-UA</td>
<td>OPC Unified Architecture - A popular protocol for communication between machines in industrial applications</td>
</tr>
<tr>
<td>OpenCV</td>
<td>Open Computer Vision - A standard library of computer vision functions</td>
</tr>
<tr>
<td>Intel® Distribution of OpenVINO™ toolkit</td>
<td>SDK developed by Intel for optimizing deep learning based algorithms running on Intel hardware</td>
</tr>
<tr>
<td>RTSP</td>
<td>Real Time Streaming Protocol - Standard protocol used for streaming video</td>
</tr>
<tr>
<td>TICK</td>
<td>Telegraf®, InfluxDB®, Chronograf®, Kapacitor®</td>
</tr>
<tr>
<td>UDF</td>
<td>User Defined Function</td>
</tr>
</tbody>
</table>

### Table 2. Security Related Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIOS</td>
<td>Basic Input/Output System used to startup a platform and initiate the boot process</td>
</tr>
<tr>
<td>CA</td>
<td>Certificate Authority - Source of traceable certificates</td>
</tr>
<tr>
<td>CRL</td>
<td>Certificate Revocation List - List of certificates that have been revoked/invalidated before expiration by a CA</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DAS</td>
<td>Data Agent Service</td>
</tr>
<tr>
<td>mTLS</td>
<td>Mutual TLS authentication - Two-way authentication of both ends of a network connection</td>
</tr>
<tr>
<td>PCR</td>
<td>Platform Configuration Registers - A feature within the TPM to cryptographically record state of software and configuration</td>
</tr>
<tr>
<td>Intel® PTT</td>
<td>Intel® Platform Trust Technology - Provides the capabilities of TPM 2.0 without dedicated hardware</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security - security protocol to encrypt and protect communication over a network</td>
</tr>
<tr>
<td>TPM 2.0</td>
<td>Trusted Platform Module - A standard for providing dedicated hardware-based security</td>
</tr>
</tbody>
</table>
2.0 Edge Insights Software Architecture

2.1 Minimum System Requirements

- Intel processor family
- 8Gb memory
- 25Gb hard disk space

2.2 Architecture

Industrial Edge Insights Software from Intel is composed as a set of containers. In the diagram below, the containers are depicted as a dashed line around the components the container holds.

Figure 1 EIS Ingredients

This configuration of EIS is designed to be deployed near the location of data generation (e.g., a PLC, robotic arm, tool, video camera, etc.) and runs the EIS Docker® containers to perform video and time-series data ingestion, real-time analytics, and allow for closed-loop control.
The following sections will describe how a user can add new ingestion streams, or algorithms within the EIS Docker* containers, and how to connect an external application (such as a Human Machine Interface (HMI), or Manufacturing Execution System (MES)) to EIS.

1 the real-time measurements, as tested by Intel, are as low as 50 milliseconds.
2 the closed loop control functions in EIS do not provide deterministic control.
2.3 Time Series Data Ingestion

The EIS comes with point data ingesting capability with Telegraf, which runs as a separate container ingesting data into InfluxDB. The configuration of Telegraf is available in `docker_setup/config/telegraf.conf`, which may need to be modified for selecting the right input plugin. The sample Telegraf used in the 1.5v EIS release is a sample temperature sensor app which sends temperature data over MQTT. The topic name and type of data is configured to work with this specific device and this needs to be updated in the configuration file for a different factory use case.

There is a Telegraf entry point script that contacts the gRPC internal server and obtains the credentials for connecting to InfluxDB. The entry point script then starts Telegraf after updating the InfluxDB credentials as environment variables.

2.4 Time Series Storage

InfluxDB is the only time series data store of EIS and forms the central connecting piece between the other EIS components.

2.4.1 Configuring Influx

The configuration parameters for the “influxdb” section are:

- **influxdb_port**
  - The port parameter need not be modified unless the default port for the InfluxDB is changed.

- **retention**
  - This field signifies the retention duration of the data stored in InfluxDB. The format is in the form of “1h30m5s”. It is recommended to keep a retention policy in accordance with the use case and the disk size.

- **username, password**
  - Provides an option to change the default username and password used for accessing the InfluxDB database.

- **DBName**
  - The default DB name in InfluxDB used for storing the point data is “datain”. This field can be used to change this default DB name.

- **ssl**
  - When True, use HTTPS instead of HTTP to connect to InfluxDB

- **verify_ssl**
  - When True, verify SSL certificates for HTTPS requests

2.5 Closed Loop Control

Once an insight is determined or inferred, an event should be initiated to act on it. This activity of performing an action as a result of some logic, is called closed loop control. EIS provides the hooks needed in order for a closed loop control application to be
written which is specific to each use case. The Stream Subscribe Library (StreamSubLib) is the interface for creating this closed loop application.

### StreamSubLib Class

The StreamSubLib is used for subscribing to various measurements from the InfluxDB and typically used to create a closed loop control application for the factory. The PCB demo reference application makes use of StreamSubLib and controls a light via Modbus TCP interfaces.

Inherits from: None


**Constructor**

```python
import StreamSubLib as ssLib
myssLib = ssLib.StreamSubLib()
```

This is the first function that needs to be called after making an object of StreamSubLib class. This init function will create a subscription on the InfluxDB instance and starts an http server for listening to the InfluxDB measurements.

- **Parameters:** None
- **Raises:** None

**Attributes**

- `config` · (dict) refers to InfluxDB configuration obtained via GetConfigInt("InfluxDB") internal gRPC call
- `certBlobMap` · (object)
- `database` · (string) database name
- `influx_c` · (object) InfluxDB client
- `log` · (object) logging.logger object for log output
- `initialized` · (boolean) True after object creation, False after deinit() called.
- `listening_thread` · (object) refers to the listener thread
- `maxbytes` · (int) default=1024
- `port` · (int) randomized
- `stream_map` · (object)
• `subscriptionName` - (string) concatenation of the database name and uuid replacing hyphens with underscores.

Methods

```python
init(log_level=logging.INFO)
```

This is the first function that needs to be called after making an object of StreamSubLib class. This init function will create a subscription on the InfluxDB instance and starts an http server for listening to the InfluxDB measurements.

**Parameters:**
- `log_level` - Log level used to track the severity of the events

**Raises:**
- `DAException("Seems to be some issue with gRPC server. Exception: (...)" - exceptions from GrpcInternalClient() or getting the "InfluxDBCfg" or "StrmLibServerCert" configurations.
- `DAException("Failed creating the InfluxDB client Exception: (...)" - failed attempt to create the InfluxDB client
- "Subscription failed due to : ..." - generic failure during subscription to the database.

Initializes an object of DataPoint class and returns to the caller. Caller needs to add the fields and tags in this object by calling DataPoint methods.

**Parameters:** None

**Return value:** (DataPoint object)

**Raises:** None

```python
deinit()
```

Removes the subscription from InfluxDB.

**Parameters:** None

**Return value:** None

**Raises:** None

```python
listen_on_server()
```

Receives data from the socket and converts it to json format and send the formatted data to the callback function.

**Parameters:** None

**Return value:** None

**Raises:**
exceptions from write_certs(), HTTPServer(), ssl.wrap_socket(), and httpd.handle_request()

Subscribe(streamName, cb)

Accepts the stream name to subscribe and the callback function to be triggered on new data

Parameters:

- **streamName** - (string) measurement to which the subscription is made
- **cb** - (function) the callback function to which data (from InfluxDB) will be sent

**Return value:** None

**Raises:** None

### 2.5.1 Time Series Analytics

In the TICK Stack, Kapacitor is a data processing engine. It lets you plug in your own custom logic to process alerts with dynamic thresholds, match metrics for patterns, or identify statistical anomalies. We will use Kapacitor to read data from InfluxDB, generate alerts, and take appropriate action based on the needs. More details and architecture about the Kapacitor can be found at [https://www.influxdata.com/time-series-platform/kapacitor/](https://www.influxdata.com/time-series-platform/kapacitor/)

The Kapacitor source used in EIS is a modified version of Kapacitor v1.5. The modifications done are to support UDFs written in Python3 or Go in addition to the default support of only Python2.x.

**Note:** The build process of EIS requires the Intel® Distribution of OpenVINO™ toolkit to be available in the DataAnalytics directory. The Docker build of EIS takes care of building the Intel® Distribution of OpenVINO™ toolkit and making it available for the Python algorithms to use.

**Data Flow Diagram**
Influx DB (Point Data)
Kapacitor (Processing Engine)
Kapacitor Configurations with UDF
TICK Scripting
InfluxDB DataIn
Classifier UDF
InfluxDB Out
Python/GO/Protobuf
USER DEFINED FUNCTION INTERFACES
info():
init():
snapshot():
restore():
begin_batch():
point():
End
### 2.5.1.1 User Defined Functions

A UDF is a user-defined function that can communicate with Kapacitor to process data through TICK Scripting. Kapacitor will stream data into it and the UDF can respond with new or modified data.

In the released version of EIS, the UDF does the image defect classification using inference algorithms that use the Intel® Distribution of OpenVINO™ toolkit. The UDF will get point data from InfluxDB and read images from the Image Store using the ImagesStore client APIs. The classifier_results, which has the defect data, is sent as a response point data from the UDF. The TICK script will save this back into InfluxDB.

Note: Usage of the Intel® Distribution of OpenVINO™ toolkit is recommended, as it will optimize the algorithms used for Intel® hardware.

#### 2.5.1.1.1 Interfaces Exposed for UDFs

There are six interfaces exposed for UDFs to communicate with Kapacitor.

1. `info()`:
2. `init()`:
3. `snapshot()`:
4. `restore()`:
5. `begin_batch()`:
6. `point()`:

### 2.5.1.2 Working with Kapacitor

Kapacitor uses a Domain Specific Language (DSL) named TICK script to define tasks involving the extraction, transformation and loading of data. One common task is defining alerts. TICK script is coded in .tick files to define pipelines for processing data. The TICK script language is designed to chain together the invocation of data processing operations defined in nodes.

In TICK script, the fundamental type is the node. A node has properties and, as mentioned, chaining methods. A new node can be created from a parent or sibling node using a chaining method of that parent or sibling node. For each node type the signature of this method will be the same, regardless of the parent or sibling node type. The chaining method can accept zero or more arguments used to initialize internal properties of the new node instance. Common node types are batch, query, stream, from, eval and alert, though there are dozens of others.
The top-level nodes establish the processing type of the task to be defined, either stream and batch, and are simply declared and take no arguments. Nodes with more complex sets of properties rely on Property methods for their internal configuration.

In TICK, each node type needs data in either batch or stream mode. Some can handle both. Each node type also provides data in batch or stream mode. Some can also provide both. This needs/provides pattern is important to understanding how nodes work together. Taking into consideration the needs/provides pattern, four general node use cases can be defined:

- need a batch and provide a stream - for example, when computing an average or a minimum or a maximum.
- need a batch and provide a batch - for example, when identifying outliers in a batch of data.
- need a stream and provide a batch - for example, when grouping together similar data points.
- need a stream and provide a stream - for example, when applying a mathematical function like a logarithm to a value in a point

More details and documentation on the TICK Scripting can be found at

https://docs.influxdata.com/kapacitor/v1.4/tick/introduction/

### 2.6 Video Data

#### 2.6.1 Ingesting Video Data

The Video Ingestion module in the EIS is a user-defined function, which uses the Data Ingestion library to ingest data to InfluxDB and Image store. The current release of EIS supports ingestion from video file, Basler cameras and RTSP cameras.
The sample provided with the EIS supports configuring the camera type and the pre-processing algorithms using the `IEdgeInsights/docker_setup/config/algo_config/factory_pcbdemo.json` file. The type of ingestor, trigger algorithm and the classification algorithm can be configured by updating this configuration file.

Reference files:
- Video ingestion pipeline: `IEdgeInsights/VideoIngestion/VideoIngestion.py`
- Trigger algorithm: `IEdgeInsights/algos/dpm/triggers`
- Data Ingestion Lib: `IEdgeInsights/DataIngestionLib`
- Image Store: `IEdgeInsights/ImageStore`

To tune the Video Ingestion for a new use case or new camera, a Python based application needs to be created similar to `VideoIngestion/VideoIngestion.py`. The library DataIngestionLib is used in the application to add image frames with meta-data into the EIS databases. All of the pre-processing analytics to be done in the Python code and the processed frames can be ingested by calling into the DataIngestionLib.

The file, `DataIngestionLib/test/DataIngestionLib_Test.py`, is a good reference to follow.
This is test code for the DataIngestionLib which reads a video file using OpenCV and uses the DataIngestionLib library to store it into InfluxDB and the Image Store.

### 2.6.1.1 Basler Camera

The Basler camera SDK is a C++ SDK downloaded from the Basler website and is available to use with the purchase of the Basler camera hardware. The Basler camera SDK is used to make the basler_video_capture Python library. This Python library is used by the Video Ingestion module to read video frames from the Basler camera. The complete setup of the Basler SDK, Python library build, etc. is taken care of by the Docker build process of the sample Video Ingestion container within EIS.
2.6.1.2 RTSP Camera

The sample Video Ingestion with EIS supports RTSP cameras to be used for ingestion. The camera ingestion is handled using the OpenCV gstreamer pipeline. The gstreamer pipeline has to be provided by the user in the factory_rtsp_hikevision_ds2.json depending on the type of camera used. Below is a sample configuration:

```
"data_ingestion_manager": {
  "ingestors": {
    "video": {
      "poll_interval": 0.01,
      "streams": {
        "opencv": {
          "capture_streams": "rtspsrc location=rtsp://4d065bbpag88364.ch.intel.com:554/cam/realmonitor?channel=1&subtype=1 user-id=admin user-pw=intel123 latency=100 ! rtph265depay ! h265parse ! mfxhevcdec ! videoconvert ! appsink"
        }
      }
    }
  }
}
```

The “rtspsrc location” and the type of codec parser and codec need to be specified depending on the camera that is being used. The codec that is used by EIS is from an Intel® Media SDK project. For various types of codecs, the gstreamer plugins listed below can be used:

**Parsers and Decoders:**
- h264parse ! mfx264dec
- h265parse ! mfxhevcdec
- mfxmpegvideoparse ! mfxmpeg2dec

2.6.2 USB Camera

Ingestion for USB camera is also handled by OpenCV’s gstreamer API by using ‘v4l2src’ plugin which is gstreamer’s default source plugin for USB Webcam. In case multiple USB cameras, specify the camera using the device property in the factory_usb.json configuration file.

Example configuration to work with two USB cameras, the sample configuration will be

```
"data_ingestion_manager": {
  "ingestors": {
    "video": {
      "streams": {
        "capture_streams": {
          "cam_serial1": {
            "video_src": "v4l2src device=/dev/video0 ! videoconvert ! appsink",
            "encoding": {
            ...
          }
        }
      }
    }
  }
}
```
2.6.3 Configuring multiple cameras

EIS supports Basler, RTSP, USB cameras. EIS can configured with multiple cameras as well. Let's look at the snapshot of a file

```
IEdgeInsights/docker_setup/config/algo_config/factory_multi_cam.json
```

```
"streams": {
    "capture_streams": {
        "cam_serial1": {
            "video_src": "rtspsrc location=\rtsp://localhost:8554/\ latency=100 ! rtph264depay ! h264parse ! mfxdecode ! videoconvert ! appsink",
            "encoding": {
                "type": "jpg",
                "level": 100
            },
            "img_store_type": "inmemory",
            "poll_interval": 0.01
        },
        "cam_serial2": {
            "video_src": "rtspsrc location=\rtsp://localhost:8554/\ latency=100 ! rtph264depay ! h264parse ! mfxdecode ! videoconvert ! appsink",
            "encoding": {
                "type": "jpg",
                "level": 100
            },
            "img_store_type": "inmemory",
```
As mentioned in the above sample snapshot, SI can mention the multiple camera ingestion streams. In this example, cam_serial1, cam_serial2, cam_serial3 are the different camera ingestion streams. The "video_src" attribute mentions that the camera type is RTSP and configuration. The "video_src" attribute's value will differ with respect to camera type. For Basler camera it will look like

```
"video_src": "pylonsrc imageformat=yuv422 exposure=3250 interpacketdelay=1500 ! videoconvert ! appsink"
```

And for the USB camera, it will look like

```
"video_src": "v4l2src ! videoconvert ! appsink"
```

### 2.6.4 Image resizing

Default resolution is that of camera resolution. The image resolution can be changed by mentioning the one attribute called 'resize_resolution'. In case resizing option needs to be enabled add the `resize_resolution` key in the video config files. The `resize_resolution` key takes frame width and frame height as, `resize_resolution = width x height`.

For example

```
"cam_serial1": {
  "video_src": "rtspsrc location=\"rtsp://localhost:8554/\" latency=100 ! rtph264depay ! h264parse ! mfxdecode ! videoconvert ! appsink",
  "encoding": {
    "type": "jpg",
    "level": 100
  },
  "resize_resolution": "1270x720",
  "img_store_type": "inmemory",
  "poll_interval": 0.01
},
```

Changing the resolution can be used to reduce the image size and intern the throughput of the EIS system.
2.6.5 Simultaneously storing the image to two different destinations

Before 1.5 release the key 'img_store_type' has two possible values "inmemory" and "persistent". In **1.5 release** the image can be stored at two different storages in single call. For the key 'img_store_type' one more value is introduced in 15 release, called 'inmemory_persistent'.

For example

```
"cam_serial1": {
  "video_src": "rtspsrc location=\"rtsp://localhost:8554/\" latency=100 ! rtph264depay ! h264parse ! mfxdecode ! videoconvert ! appsink",
  "encoding": {
    "type": "jpg",
    "level": 100
  },
  "img_store_type": "inmemory_persistent",
  "poll_interval": 0.01
}
```

2.6.6 Specifying encoding options

Encoding is specified using the json object as below

```
"cam_serial1": {
  "video_src": "rtspsrc location=\"rtsp://localhost:8554/\" latency=100 ! rtph264depay ! h264parse ! mfxdecode ! videoconvert ! appsink",
  "encoding": {
    "type": "jpg",
    "level": 100
  }
}
```

The type can be either 'jpg' or png.

The level talks about the compression level for a specific encoding type. For 'jpg' encoding, the level can be specified in the range '0-100'. For 'png' encoding, the level can be mentioned in the range '0-9'.

For detailed description of the supported cameras in EIS, please refer IEdgeInsights/VideoIngestion/README (section name “Supported cameras via gstreamer pipeline”). This README describes the different cameras and their configurations.

2.6.7 Using a trigger

The Video Ingestion module in the EIS is a user defined function, which uses the Data Ingestion library to ingest data to InfluxDB and Image store. The current release of EIS supports ingestion from video file, Basler cameras and RTSP cameras.
The samples provided with the EIS supports configuring the camera type and the pre-processing algorithms using the config files in

```
IEdgeInsights/docker_setup/config/algo_config/.
```

The type of ingestor, trigger algorithm and the classification algorithm can be configured by updating this configuration file. Any files needed for classification or trigger can be copied to the same directory location. The configuration file to be used for the current execution of EIS is selected by updating the `CONFIG_FILE` option in

```
IEdgeInsights/docker_setup/.env
```

Reference files:

- PCB Demo configuration file for video file as input:
  `IEdgeInsights/docker_setup/config/algo_config/factory_pcbdemo.json`
- Classification sample configuration file for video file as input:
  `IEdgeInsights/docker_setup/config/algo_config/factory_classification.json`

[* The rest of the documentation uses the filename “factory.json” as a generalized name for these config files.]

Support files needed for classification/trigger:

- `IEdgeInsights/docker_setup/config/algo_config/ref_pcbdemo/`
- `IEdgeInsights/docker_setup/config/algo_config/ref_classification/`

- PCB Demo trigger algorithm: `IEdgeInsights/algos/dpm/triggers/pcb_trigger.py`
- Data Ingestion Lib: `IEdgeInsights/DataIngestionLib`
- Image Store: `IEdgeInsights/ImageStore`

If a video file is used in place of a camera stream, copy the video file to

```
IEdgeInsights/docker_setup/test_videos/ directory.
```

### 2.6.7.1 Trigger Algorithm Format

The second block in the video ingestion block diagram is the trigger algorithm. The purpose of trigger algorithm is to select frames of interest from the camera stream. The algorithms depends on the use case, people monitoring might need classifier algorithms to execute on all frames from the camera whereas a use case like the sample application “pcbdemo” would need the classifier to execute on only the frames where the PCB board is in the center of the frame.
EIS comes with a few sample trigger algorithms in EdgeInsights/algos/dpm/triggers/. The format for trigger algorithm is as below:

```python
import <libraries>
from . import BaseTrigger

class Trigger(BaseTrigger):
    def __init__(self, var1, var2,...):
        # Constructor

    def get_supported_ingestors(self):
        return ['video', 'video_file']

    def on_data(self, ingestor, data):
        ...
        self.send_start_signal(data, -1)
        ...
        self.send_data(data, <user data>)
        ...
        self.send_stop_signal()
        ...
```

The start and stop signal bounds all the frames captured for a specific part/item/product on the factory floor. This ability is keeping in view that some part/item/product might have more than one frame that need to be captured to pass through the classifier. Ex.: In the case of a 3 dimensional object with defects on multiple faces rotating in front of the camera, we need to be able to capture images of all possible faces where defects can occur. The start and stop signal helps combine all the faces captured to one part id.

Each trigger algorithm is associated with its configuration in factory.json

```
"triggers": {
  "<trigger_name>": {
    "var1": "<var1 value>",
    "var2": "<var2 value>",
    ...
  }
}
```

The “triggers” configuration specifies the trigger algorithm(s) implemented and any of the user defined variables (var1, var2, ...) that need to be passed to the Trigger object. Ensure all, if any, file paths input as the user defined variables are located in a volume location for the Video Ingestion container.
In use cases where all the frames need to be classified by the classifier algorithm, we can use the bypass_trigger.py file. Classification sample code uses the bypass_trigger to classify every frame in the video.

2.7 Image Store

The Image Store component of EIS comes as a separate container providing an mTLS-enabled gRPC interface for storing, retrieving and deleting images as binary blobs. The images can be stored in-memory or in persistent storage. The Image Store container internally uses Redis for in-memory storage and Minio for persistent storage. A Python and go client library for the Image Store gRPC interface are also provided. This image store client is used by internal infrastructure components and can also be used by external clients. For usage by external clients, separate client certificates will need to be generated using the cert-tool.

The Image Store is typically used in the following two ways:

- Using the DataIngestionLib interface and adding byte stream data, which gets stored to the Image Store. This will also add the point data (meta-data) for it in InfluxDB. This is the preferred method of using the ImageStore library for ingest ing the video / image frames which need to be processed by the Data Analytics or exported by Stream Manager.

- Directly using the ImageStore library’s read and store APIs. The store API can be used if the UDFs need to keep binary blobs without adding meta-data to the InfluxDB. The read API would be used in the analytics UDF for getting the images to pass to the ML / DL algorithms.

2.7.1.1 Data Ingestion Library

The Data Ingestion library is a Python client library for ingesting data into the InfluxDB and Image Store. This library is used by the user-defined functions for ingesting all types of data – integer, float point, string and binary blobs. The Data Ingestion library takes care of storing image data as binary blobs into the Image Store, generating the meta-data for the data, and storing the meta-data into the InfluxDB.

2.7.1.2 Configuring the Image store

The Image Store DB uses two databases: Redis for in-memory storage and Minio for persistent storage. The configuration parameters for the three configuration sections are outlined below.

Redis configuration “redis”:
- **redis_port**
  - The port parameter specifies the socket port to use when connecting to the Redis and Minio by the Image Store component.
    - **Note:** Do not edit this parameter without making the corresponding change to the configuration files of the Redis and Minio DBs.

- **password**
  - The password used to access the Redis DB.
    - **Note:** Do not edit this parameter without making the corresponding change to the configuration files of the Redis and Minio DBs.

- **retention**
  - The retention parameter specifies the retention policy to apply for the images stored in Redis for in-memory and Minio for persistent storage. This parameter needs to be changed based on the use case. Because Redis uses in-memory storage, keeping a large value is not recommended since it will increase system RAM usage. The default value is "40s", which should be updated based on the use case.

Persistent image storage "persistent_image_store":

- **type**
  - This is set to "minio" and must not change because the Minio DB is being used.

Minio configuration "minio":

- **minio_port**
  - The port parameter specifies the socket port to use when connecting to the Redis and Minio by the Image Store component.
    - **Note:** Do not edit this parameter without making the corresponding change to the configuration files of the Redis and Minio DBs.

- **access_key**
  - The Minio access key is similar to a username used to access the Minio DB.
    - **Note:** Do not edit this parameter without making the corresponding change to the configuration files of the Redis and Minio DBs.

- **secret_key**
  - The Minio secret key is similar to a password used to access the Minio DB.
    - **Note:** Do not edit this parameter without making the corresponding change to the configuration files of the Redis and Minio DBs.

- **retention_time**
  - The retention parameter specifies the retention policy to apply for the images stored in Redis for in-memory and Minio for persistent storage. This parameter needs to be changed based on the use case. The
default value is “400s”, which should be updated based on the use case. Because Minio stores data on disk, a larger value can be kept and the retention value can be determined depending on limits on disk usage.

- **retention_poll_interval**
  - This is used to set the time interval for checking images for expiration in the Minio DB. Expired images will become candidates for deletion and no longer retained.

- **ssl**
  - Set to “false” to disable the use of SSL when connecting to the Minio DB.
    - **Note:** Do not edit this parameter without making the corresponding change to the configuration files of the Redis and Minio DBs.
2.8 Video Analytics

The classifier algorithm/UDFs get executed in the ia_video-analytics container. EIS is a deployment tool. As such, it is assumed the classifier algorithm is already developed and validated on a dataset collected at the factory. The following sections describe how to integrate the classifier algorithm to EIS and not how to develop an algorithm.

In the released version of EIS, the UDF does the image defect classification using inference algorithms that use OpenVINO. The UDF will get point data from InfluxDB and read images from the Image Store using the ImagesStore client APIs. The classifier_results, which has the defect data and display information, is sent as a response point data from the UDF. The TICK script will save this back into InfluxDB.

Note: Usage of OpenVINO is recommended, as it will optimize the algorithms used for Intel hardware.

2.8.1 Developing Classifier Code in Data Analytics

The classifier code in EIS is available in Edgelnsights/algos/dpm/classification/classifiers directory. EIS comes with two sample algorithms: "classification_sample" and "pcbdemo" which can be used as a reference for developing new classifiers. The classification_sample algorithms is a direct porting of the python sample code in OpenVINO inference samples.

The format for classifier code is as shown below:

```python
import <libraries>
from algos.dpm.defect import Defect
from algos.dpm.display_info import DisplayInfo
import openvino.inference_engine

class Classifier:
    def __init__(self, var1, var2, ...):
        # Constructor

    def classify(self, frame_num, img, user_data):
        # Main classifier algorithm

        defect_array.append(Defect(<defect id>, (xmin, ymin), (xmax, ymax)))
        info_array.append(DisplayInfo("<string to display in UI>", priority#))

        return info_array, defect_array
```

- **Defect class** to store bounding box coordinates if needed
- **DisplayInfo class** to store any message to be displayed on screen in the UI/visualizer for the frame
- **OpenVINO inference engine library for inferencing using IA optimized IR models**
- **“Classifier” object**
- **Constructor parameters are derived from factory.json**
- **Main classifier algorithm**
- **classify() returns display and defect information**
"Defect" class can be used to return location information of defects or objects of interest in the algorithm. The “Defect” class includes the type of defect/object and the top-left and bottom-right (x, y) coordinates of the region of interest on the current frame that is classified.

"DisplayInfo" class can be used any message or string associated with the current frame while being displayed on UI/visualizer. The “DisplayInfo” class includes the string and the priority [0/1/2] of the string. 0 implies low priority and the string will be displayed in Green text on the visualizer. 1 is medium and the string will be displayed in Yellow text. 2 is high priority and string will be displayed in Red text. Low priority is often used for displaying any informational texts. High priority is for warning about something associated with the frame.

As mentioned before, usage of OpenVINO is recommended for optimizing DL models for Intel architecture. As such, importing OpenVINO inference engine is dependent on whether the classifier algorithm uses the standard DL model or OpenVINO optimized IR files.

The main classifier algorithm goes to “classify” function in the “Classifier” class object. This function is called with the frames selected by the video ingestion container along with any user data passed on by the trigger algorithm. Depending on the algorithm for selecting frames, this could be every frame from the camera stream or specific frames picked out because of something of interest happening in them. In the pcbdemo sample code the pcb_trigger algorithm picks out frames with the board in the center of the frame and classifier algorithm executes only on those selected frames.

The pcbdemo sample application is a combination of computer vision and deep learning algorithms for detecting defects on a PCB board. Please refer to OpenVINO sample codes or the pcbdemo sample for OpenVINO API usage.

Each classifier algorithm is associated with its corresponding configuration in factory.json.

```json
"classification": {  
  "max_workers": 1,  
  "classifiers": {  
  "<classifier_name>": {  
    "trigger": ["<trigger1_name">", "<trigger2_name>"]},  
    "config": {  
    "var1": <var1 value>,  
    "var2": <var2 value>,  
    ...  
  }
  }
}
```

The classifier configuration specifies the classifier name, the trigger algorithm(s) associated with the specific classifier and any of the user defined variables (var1, var2,
...) that need to be passed to the Classify object. Ensure all, if any, file paths input as the user defined variables are located in a volume location for the DataAnalytics container, the default location assigned is in `IEdgeInsights/docker_setup/config/algo_config` directory.

Please refer `IEdgeInsights/docker_setup/config/algo_config/factory_pcbdemo.json` for sample file paths. The input files for the pcbdemo application is located in the same directory, `ref_pcbdemo/*`. Please note the file paths in `factory.json` is the path inside the ia_data_analytics container.

### 2.8.2 Algorithm configuration files

Each algorithm in EIS is associated with a configuration file located in `IEdgeInsights/docker_setup/config/algo_config/`.

The following are some sample `factory.json` files for different scenarios that can be used as reference while developing and integrating a new algorithm:

- **factory_pcbdemo.json** – config file for executing pcbdemo sample algorithm with a video file (pcb_d2000.avi)
- **factory_basler.json** – sample config file for executing pcbdemo sample algorithm with a single basler camera setup.
- **factory_rtsp_hikvision_ds2.json** – sample config file for working with physical hikvision ds2 rtsp camera setup
- **factory_rtsp_cvlc.json** - sample config file for working with stimulated cvlc rtsp camera setup
- **factory_usb.json** - sample config file for working with usb webcam.
- **factory_multi_cam.json** - sample config file for working with multiple streams coming from the same or diff sources (rtsp, basler, usb)

*Please note:* the support files (reference file, ROI json file and DL models) would need to be updated for the new setup. The DL models in `ref_pcbdemo` are only trained for the video `pcb_d2000.avi`. Any new setup would require a trained DL model for that setup.

- **factory_classification.json** - config file for executing classification_sample sample algorithm with a video file (classification_vid.avi)
"machine_id": "<tool # / ID>",
"trigger_threads": <#>,
"data_ingestion_manager": {  
    "ingestors": {  
        "video_file": {  
            "video_file": ".//test_videos/<video file>",  
            "encoding": {  
                "type": "jpg",  
                "level": 100  
            },  
            "img_store_type": "inmemory_persistent",  
            "loop_video": true  
        },  
    },
    "triggers": {  
        "<trigger_name>": {  
            "var1": "<var1 value>",  
            "var2": "<var2 value>",  
            ...  
        },  
    },
    "classification": {  
        "max_workers": 1,  
        "classifiers": {  
            "<classifier_name>": {  
                "trigger": ["<trigger1_name>", "<trigger2_name>"],  
                "config": {  
                    "var1": <var1 value>,  
                    "var2": <var2 value>,  
                    ...  
                }  
            }  
        }  
    }  
}
2.8.3 Using Accelerators

EIS supports the following Data Analytics accelerators: CPU, iGPU, Myriad Neural Compute Stick 2, and PCIe based vision accelerators based on Movidius VPUs (HDDL). By default, EIS is configured to run inference on the CPU, but this can be changed by changing the "device" value while calling the OpenVINO inference engine API : IEPlugin. Please refer to OpenVINO sample codes/website for API usage information.

If the user wants to extract the ability to assign run time devices from the classifier algorithm file to the factory.json, a sample integration is shown in pcbdemo application.

```
# Select run time device [CPU/GPU/MYRIAD], default : CPU
r_device = "CPU"
if device.upper() == "GPU":
    r_device = "GPU"
if device.upper() == "MYRIAD":
    r_device = "MYRIAD"

# Load OpenVINO model
self.plugin = IEPlugin(device=r_device, plugin_dirs="")
```

The ‘device’ field in factory.json can be changed to one of the supported options (not case sensitive): CPU, GPU, Myriad, or HDDL. There is one additional step enable HDDL as the inference accelerator. To use HDDL, the user must also uncomment lines 12 & 13 in DataAnalytics/VideoAnalytics/va_classifier_start.sh. These lines should be commented out when using any other accelerator, because it starts the HDDL daemon in a background process.

2.9 EIS Data Agent

The Data Agent includes the Data Agent Service, Stream Manager, the data security framework and Databus Abstraction in a single executable, The Hashicorp Vault is included in the container but is a separate process. These modules are differentiated in the block diagram as they are considered logically independent modules. The Stream Manager and Databus Abstraction are libraries and packages that get included into the Data Agent.

2.9.1 Data Agent Service

The Data Agent Service performs the configuration of the EIS stack and manages the various data streams produced by the ingestion and analytics functions. The major functions of Data Agent Service are:

- Accept configuration of the Edge Compute Node (via configuration file)
- Run the Vault Server to store the secrets in encrypted form
Initialize the InfluxDB and create the database for point data storage.
Initialize the Stream Manager component, which handles the export of streams out of the Edge Compute Node.
Expose a GetConfigInt gRPC interface for all other internal components to get the provisioned secrets (InfluxDB, Redis*, and Minio* DB credentials, private keys and certificates for various servers).
Provide the private key and certificate for accessing GetConfigInt to all other containers.
Initialize the Databus Abstraction module which in-turn makes the OPC-UA Server for doing data export.

The Data Agent configuration information is provisioned to the vault during the provisioning step.

Below is the default Data Agent configuration found at docker_setup/provision_config.json:

```json
{
  "influxdb": {
    "influxdb_port": "8086",
    "retention": "1h30m5s",
    "username": "admin",
    "password": "admin123",
    "dbname": "datain",
    "ssl": "True",
    "verify_ssl": "False"
  },
  "redis": {
    "redis_port": "6379",
    "password": "redis123",
    "retention": "40s"
  },
  "persistent_image_store": {
    "type": "minio"
  },
  "minio": {
    "minio_port": "9000",
    "access_key": "admin",
    "secret_key": "password",
    "retention_time": "1h",
    "retention_poll_interval": "60s",
    "ssl": "false"
  }
}
```
"opcua": {
  "port": "4840"
},

"OutStreams": {
  "classifier_results": "OPCUA",
  "video1": "NATS"
}

**Note:** It is important that default security credentials and settings are provided. It is recommended that these be changed before an EIS based system is deployed into a production environment.

### 2.9.2 Databus Abstraction Configuration

The Data Bus Abstraction is a library provided in the EIS stack that provides publish-subscribe interfaces for transporting data out of the Edge Compute Node. The abstraction layer is provided to support multiple underlying transport protocols such as OPC-UA, DDS, MQTT, AMQP etc. Only OPC-UA is supported in this release.

The only supported data bus protocol in the 1.5 release of EIS is OPC-UA. The only configurable option for the OPC-UA server specified in the “opcua” section is the port number:

- **port**
  - The default port of OPC-UA (4840/tcp) can be changed by updating the "port" parameter.
  - **Note:** If port is changed, the corresponding change must also be made in the `docker_setup/.env` file.

### 2.9.3 Stream Manager Configuration

In the Stream Manager configuration, the data stream that needs to be exported from the EIS stack can be specified. The only feature supported in the 1.02 release of EIS is **OutStream**. **OutStream** indicates the data stream that is sent out of the node via the data bus. The **OutStream** is configured in the "OutStream" section where the parameters are the named stream and the value set to the data bus format of the stream:

- **classifier_results**
  - "classifier_results" is the stream name of the sample results stream which is generated by Data Analytics.
  - The value "OPCUA" specifies the format to use when sending the classifier_results stream.
  - **Note:** The only format supported in the current release is "OPCUA".
- **video1**
  - “video1” is the stream name of the input video frames stream which is consumed by Data Analytics.
  - “NATS” specifies the format of the video1 stream.

The Stream Manager is the module that handles exporting the data stream outside of the Edge Compute Node. The Video Ingestion and Data Analytics modules can generate multiple streams, which get stored into InfluxDB as measurements. Not all of these are to be exported out of the Edge Compute Node according to the use case. In the sample use case of PCB demo, the Data Analytics module consumes the input video stream generated by the Video Ingestion module and, in-turn, produces the classifier_results stream. Only the classifier_results stream is exported to the Edge Server.

![Diagram of data flow](image)

The Stream Manager component makes a subscription to the InfluxDB and receives all measurements / streams from it. Then based on the configuration option, it will use the data bus client Publish(OPCUA) function to export the stream.

### 2.9.4 GetConfigInt Interface

The GetConfigInt interface is exposed by the internal gRPC server of Data Agent. This is an mTLS enabled interface and is used by the infrastructure components like Video Ingestion, Data Analytics and Image Store for getting the updated configuration data after the user has edited the provisioning configuration file. This interface is also used for getting various private keys and certificates needed by the EIS components for its servers and client connections.

### 2.9.5 External Library

In the Edge Compute Node, the classifier results of each part is exposed as a string variable in OPC-UA namespace. The string variable is formatted as JSON.

The data bus abstraction provides publish-subscribe interfaces which get mapped to OPC-UA variable updates. The topic name gets mapped to the variable name in the OPC-UA namespace. Each publish on the topic updates the value of the variable exposed by the server. On the subscription side, an OPC-UA feature to subscribe for data change notifications is used to get the new value whenever the data value is updated for the subscribed variable.

The External library includes these modules:
- Data bus abstraction (DataBus) to subscribe to the classifier results data.
• Image Store (ImageStore) for receiving the images from Image Store.

These modules are available in Python and are tested with Python3.6.
The directory structure of dist_libs is shown below:

```
/opt/intel/iei/dist_libs/
 ├── DataBusAbstraction
 │   └── c
 │       └── common.h
 │       └── DataBus.c
 │       └── DataBus.h
 │   └── doc
 │       └── html
 │           └── annotated.html
 │           └── bc_s.png
 │           └── bdwn.png
 │           └── classes.html
 │           └── closed.png
 │           └── DataBus_8h.html
 │           └── DataBus_8h_source.html
 │           └── dir_361e8465257f7c8b86d659a4a1b9003.html
 │           └── dir_beae23c2702a63294004fd9f6fd117ba.html
 │           └── doc.png
 │           └── doxygen.css
 │           └── doxygen.png
 │           └── dynsections.js
 │           └── files.html
 │           └── folderclosed.png
 │           └── folderopen.png
 │           └── functions.html
 │           └── functions_vars.html
 │           └── globals_defs.html
 │           └── globals_func.html
 │           └── globals.html
 │           └── globals_type.html
 │           └── index.html
 │           └── jquery.js
 │           └── menudata.js
 │           └── menu.js
 │           └── nav_f.png
 │           └── nav_g.png
 │           └── nav_h.png
 │           └── open62541__wrappers_8h.html
 │           └── open62541__wrappers_8h_source.html
 │           └── open.png
 │           └── search
 │               └── all_0.html
 │               └── all_0.js
 │               └── all_1.html
 │               └── all_1.js
 │               └── all_2.html
 │               └── all_2.js
 │               └── all_3.html
 │               └── all_3.js
 │               └── all_4.html
 │               └── all_4.js
 │               └── all_5.html
 │               └── all_5.js
 │               └── all_6.html
 │               └── all_6.js
 │               └── all_7.html
 │               └── all_7.js
 │               └── all_8.html
 │               └── all_8.js
```
2.9.5.1 Modules

DataBus module: Data bus abstraction to subscribe to the classifier results data.
Client module: An Image Store client that provides a wrapper for gRPC and protocol buffers for receiving images from Image Store.

2.9.5.2 Classes

databus Class: Creates an instance of databus

GrpcImageStoreClient Class: Wrapper class for the gRPC Image Store client

2.9.5.3 Class Reference

databus cd
Class

Creates an instance of databus

Inherits from: None


Constructor

import DataBus as dbus
mydatabus = dbus.databus(log)

Parameters:  
  log - logging.logger object for log output

Raises: none

Attributes

• bus - (object) points to the instance of a concrete class of data bus type, default=None
• busType - (string) OPCUA/MQTT/NATS (Only OPCUA supported now), default=None
• direction - (string) PUB/SUB, default="NONE"
• logger - (object) logging.logger object for log output
• mutex - (object) lock object to ensure there are no race conditions for pub/sub, default=Lock()

• pubTopics - (dict) list of topics to be published, default=[]
• subTopics - (dict) list of topics to be subscribed, default=[]
• busType - (string) OPCUA/MQTT/NATS (Only OPCUA supported now), default=None

Methods
ContextCreate(contextConfig)

Create an underlying message bus context for the databus

Parameters:

- `contextConfig` - (dict) Message bus parameters to create the context. The required dictionary object items are:
  - "direction" - one of either "PUB" for a publisher, "SUB" for a subscriber
  - "name" - context namespace (PUB/SUB context namespaces should match)
  - "endpoint" - message bus endpoint address formatted as proto://host:port/, proto://host:port/ ...

Example:

OPCUA -> "opcua://0.0.0.0:4840/"
MQTT -> "MQTT://localhost:1883/"
NATS -> "nats://127.0.0.1:4222/"

Only OPCUA is tested. MQTT is implemented only for reference.

- "certFile" - server/client certificate file
- "privateFile" - server/client private key file
- "trustFile" - ca cert used to sign server/client cert

Return value: None

Raises:

- "Not a supported BusType" - endpoint protocol not one of OPCUA, opcua, MQTT, mqtt, NATS, or nats.
- "Not a supported BusDirection" - direction not one of PUB or SUB
- "{...} Failure!!!" - could not create indicated context

Publish(topicConfig, data)

Publish data on the databus

Parameters:

- `topicConfig` - (dict) Publish topic parameters. The required dictionary object items are:
  - "name" - Topic name
  - "type" - Data type associated with the topic (only "string" type currently supported)
“data” - The message whose type should match the topic data type

Return value: None

Raises:

- “Not a supported Message Type” - the “type” was not “string”.
- “Topic name & Type not matching” - the name and type dictionary items were not present
- “Not a supported BusDirection” - the databus.direction is not “PUB”
- “{…} Failure!!” - failed to send the message on the databus

Subscribe(topicConfig, trig, cb=None)

Subscribe data from the databus

Parameters:

- topicConfig - (dict) Subscribe topic parameters. The required dictionary object items are:
  - “name” - Topic name (in hierarchical form with ‘/’ as delimiter)
  - “type” - Data type associated with the topic (only “string” type currently supported)
  - “trig” - “START” or “STOP” the subscription
  - “cb” - A callback function with data as its argument

Return value: None

Raises:

- “Already Subscribed!!!” - the indicated topic has already been subscribed.
- “Topic name & Type not matching” - the name and type dictionary items were not present
- “Unknown Trigger!!” - trig was not “STOP” or “START”
- “{…} Failure!!” - failed to subscribe the topic on the databus

ContextDestroy()

Destroys the underlying message bus context. Also unsubscribes all existing subscriptions.
Return value: None

Raises:

- "{} Failure!!!" - failed to destroy the databus context

**GrpcImageStoreClient Class**

A wrapper class for the gRPC Image Store client

Inherits from: object


**Constructor**

```python
import client as cl
myclient = cl.GrpcImageStoreClient(clientCert, clientKey, caCert, hostname="localhost", port="50055")
```

**Parameters:**

- `clientCert` - (string) filename of the imagestore client certificate
- `clientKey` - (string) filename of the imagestore client key
- `caCert` - (string) filename of the ca certificate
- `hostname` - (string) the hostname/ip address of the m/c where the DataAgent module of EIS is running, default="localhost"
- `port` - the gRPC port, default=50055

**Raises:**

- file open exceptions
- grpc exceptions

**Attributes**

- `hostname` - (string) the hostname/ip address of the m/c where the DataAgent module of EIS is running, default="localhost"
- `port` - the gRPC port, default=50055

**Methods**

**Read(imgHandle)**

Read is a wrapper around the gRPC Python client implementation for Read gRPC interface.

**Parameters:**

- `imgHandle` - config(string)
Return value: (byte[]) the byte stream corresponding to the config value
Raises: None

Store(byteStream, memType)

A wrapper around gRPC Python client implementation for Store gRPC interface
Parameters:
- byteStream - (byte[]) byte stream to be stored
- “memType” - inmemory or persistence
Return value: The imgHandle corresponding to the stored config value
Raises: None

Remove(imgHandle)

Remove is a wrapper around the gRPC Python client implementation for Remove gRPC interface
Parameters:
Return value: None
Raises: None

2.9.6 Security

The primary security objective of the EIS is to prevent the point data and video data generated by the ingestion / analytics components from being accessed by unauthorized entities acting within the system or accessing it via the external network interfaces.

Security is enabled in a two-stage process involving provisioning and execution on an Edge Compute Node. The root user is considered a trusted entity and hence compromising the root password or giving Linux* sudoer permission to an untrusted user on the Edge Compute Node will compromise security.

2.9.6.1.1 Vault

The Data Agent container runs the Hashicorp Vault software to store the provisioned secrets along with private keys and certificates into a secure storage.
2.9.6.2 Provisioning

During provisioning, the system administrator provides to the provisioning tool all the required secrets that need to be sealed into the Vault. Provisioning should be performed in a secure environment where the secrets are available in clear text for the administrator. It is recommended to provide the secrets in a separate storage device, such as an USB stick, and then removing the device after provisioning is completed.

2.9.6.2.1 Generating Keys and Certificates

The system administrator will first run the cert-tool available with EIS source code to generate all of the required certificates and keys that are needed by the EIS modules. The cert-tool by default generates the CA certificate and CA private key, which is the root certificate for the ECN deployment. The generated CA key needs to be kept securely with the administrator and is not required in the remaining provisioning steps.

When the administrator wants to provide access to the EIS data to a new third-party client (via gRPC and OPC-UA), the cert-tool is run with the previously generated CA certificate and key as arguments. In this mode, the cert-tool will output a new client certificate and client key, which can be provided to the new EIS client.

If any of the client certificates becomes compromised, then a complete certificate re-provisioning needs to be done for all the EIS deployed nodes by generating a new CA certificate and private key.

2.9.6.2.2 Provisioning Tool

The provisioning tool takes as input a JSON configuration file that defines the path of the certificates generated by cert-tool. All of these details are consumed by the provisioning tool, which adds these secrets into the Hashicorp Vault. The Vault software encrypts these secrets and stores them in the file system.

The Vault is secured using an un-seal key and token, which need to be saved for accessing the secrets stored in the Vault. This un-seal key and token are stored into the TPM 2.0 hardware on the host machine. The Intel® PTT firmware can also be enabled in configurations without dedicated TPM 2.0 hardware.

The vault un-seal key and token are sealed into the TPM using the Policy Authorization method introduced in the TPM 2.0 standard. A new set of TPM authorization public-private keys are generated by the cert-tool for TPM sealing. The secret is sealed to the TPM with the TPM authorization public key. The secret unsealing happens by creating an authorization policy with a specific set of PCR values that need to be checked against for confirming a trusted platform boot. These PCR values are obtained on a trusted test machine with the same configuration and OS as the deployment machine. The test machine is booted completely and the trusted PCR values are read to create the authorization policy. This authorization policy is signed using the TPM authorization private key, which needs to be securely kept with the SI / factory administrator. This key
is required to create a new authorization policy during system updates. If any of the PCR values do not match with the authorization policy, then the unsealing process will fail and Data Agent Service will fail to start.

2.9.6.3 Start up and Run Time

During Edge Compute Node reboot, the secrets stored in the Vault need to be accessible to all of the EIS infrastructure containers. Data Agent plays a crucial role in providing the secrets to other containers.

2.9.6.3.1 Secure Boot

The platform secure boot with PCR extension measurements will verify the BIOS, bootloader, Linux kernel and Initrd and extended corresponding PCRs. The EIS runs as a system application on the Linux distribution. The first container to startup is the Data Agent Service (DAS), which tries to read the secrets out of the TPM. The sane values of PCRs on an untampered system are created as part of an authorization policy, which is signed and is available in the file system. The DAS will pass this signed authorization policy to the TPM to unseal the secret. If there are no changes to the PCRs, proving an untampered platform boot, the TPM sealed secrets (vault un-seal key and token) are then read successfully from the TPM by the DAS.

2.9.6.3.2 Accessing Secrets

After completing a successful secure boot step, the DAS container will get the vault unseal key and token. The DAS will open the Vault and read out all of the secrets provisioned into it during system provisioning.

The DAS provides an internal mutual-TLS (mTLS) enabled gRPC server called GetConfigInt that is available on port 50052 by default. This gRPC server is accessed by all the infrastructure containers (Video Ingestion, Data Analytics, Image Store, etc.). The GetConfigInt server provides the requested secrets to the other containers in JSON form. The infrastructure containers can ask for database credentials, server and client certificates and keys from GetConfigInt.

The mTLS client key and certificate required for accessing the internal gRPC server is provided to the infrastructure container DAS by following these steps:
1) The EIS startup script (compose_startup.sh) generates a symmetric key on every startup and sets it as an environment variable to the shell running the `docker-compose up` command.

2) DAS starts up and reads the GetConfigInt client key and certificate from the Vault.

3) DAS uses the symmetric key from an environment variable (set by compose_startup.sh) to encrypt the key and certificate and stores the encrypted blobs to the file system in a fixed location. This location is added as a mount point into all other EIS containers.

4) EIS infrastructure containers will decrypt the blob from the symmetric key available in an environment variable and extract the GetConfigInt client certificate and key.

2.9.6.4 Retiring

2.9.6.4.1 Certificate revocation

The current version of EIS does not support certificate revocation using the CRL mechanism. In case of stolen or lost certificates, the System Integrator must manually re-provision all nodes by following the steps mentioned in the Provisioning section. This will generate a new set of certificates and keys that will replace any lost or stolen certificates/private keys, including the CA certificate, and provisions them into the Vault.

**Note:** This sequence mentioned above must be carried out on all EIS nodes installed in the network. Failing to do this might expose a node to attack using a stolen certificate and private key.
3.0  **DEV MODE and Profiling**

3.1  **DEV MODE**

3.1.1  **Starting the EIS in dev mode**
Dev mode eases development phase for SI. This mode can be enabled by setting the environment variable named “dev_mode” to “true” in the docker_setup/.env file. Default value of this variable is “false”.

3.1.2  **Security features disabled in dev mode**
In dev mode the following security features are disabled.
- All the components are communicating over the non-encrypted channels.
- Credential provisioning step is skipped. The ‘provision_config.json’ file is used as is, instead of provisioning to the vault.

Note: User should not use this mode in production environment, since the above mentioned security features are disabled in this mode.

3.1.3  **Faster development cycle while developing classifier algorithm**

**Without the dev mode,** SI has to go through the below steps
1. Modify-algorithm
2. Dockerize algorithm (docker image build process)
3. Restart-EIS
4. Watch for expected result.

**In dev mode,** SI can modify the algorithm and run the algorithm like any other python program on bare metal. So algorithm development cycle goes through only two steps.
1. Modify algorithm
2. Run and Watch for expected result.

The ‘IEdgeInsights/DataAnalytics/sample_analytics_app’ directory also has sample template (‘sample_analytics.py’) to immediately start algorithm development. The file ‘sample_analytics.py’ has the function named ‘analyzeFrame’ where algorithm writer should start developing the algorithm. SI can even create their own analytics container too. Please refer the file ‘IEdgeInsights/DataAnalytics/sample_analytics_app/README.md’.
3.1.4 Faster development cycle while developing classifier and trigger algorithm

In dev mode the "algo" directory is mounted inside the "ia_video_analytics" container. So if SI can make the modification into the algorithm and restart EIS without rebuilding an image. So algorithm development cycle goes through only two steps.

1. Modify algorithm
2. Run and Watch for expected result.

3.2 PROFILING MODE

Profiling mode is used to find out performance statistics. In this mode, every EIS component records the processing time taken for any single frame. In the visualizer, all the statistics are gathered and SI can see the end to end processing time, taken for individual frame and end to end average time taken. This mode can be enabled in EIS by setting the environment variable named "PROFILING" in docker_setup/.env to value "true". To start visualizer in profiling mode, requires minor changes in Makefile. In the Makefile there are two makefile variables named "SECURE_VISUALIZE_ARGS" and "NON_SECURE_VISUALIZE_ARGS". These variable values, represents the arguments to visualizer. One of the argument is –P, with default value false. Change this value to true (Replace “–P false” with “–P true”). Please refer the README of visualizer app, to understand more about the statistics and their meaning.
4.0 Code Modification Recommendations

Below is the list of modules, which need to be modified for implementing various factory use cases. The module name corresponds to the directory name in the IEdgeInsights directory.

- **Algos** – This directory contains all the user-defined algorithms that are executed as a trigger (algos/dpm/triggers) or as a classifier (algos/dpm/classification/classifiers) for the input video stream.

- **VideoIngestion** – Contains the logic of ingesting frames into the InfluxDB and Image Store using the DataIngestionLib. This module needs to be modified if the use case demands ingestion in a different way than provided as default. E.g. ingesting frames only to in-memory store, adding special meta-data along with the ingested frames, etc.

- **DataAnalytics** – This module runs the UDF, which is registered with Kapacitor within a default TICK script. Modify this for scenarios such as when changing the TICK script, for changing the integration with classifier algorithm or for changing the classifier_results format going into the InfluxDB.

- **FactoryControlApp** – For closed loop control specific to a factory use case.

Below is the list of modules that are not recommended to be modified. Modifying these modules can compromise the security architecture of the EIS stack.

- DataAgent
- StreamManager
- ImageStore
- DataBusAbstraction
- DataIngestionLib
- StreamSubLib
5.0 **Visualizer (IEI Simple Visualizer)**

EIS stack provide the default visualizer with the release package. This will show the images exported by the EIS stack on a UI window with the defect bounding boxes drawn on the images. This can be referred for creating any kind of visualization with the exported data.

This visualizer can subscribe to the streams published by DataAgent service of EIS. The streams to be captured can be mentioned in the 'config.json' file. This visualizer can be run with different argument to enable following features.


[-d DISPLAY] [-i IMAGE_DIR] [-D DEV_MODE]

[-P PROFILING_MODE]

Optional arguments:

-h, --help show this help message and exit

-c CERT_DIR, --cert_dir CERT_DIR

IEI certificates directory (default: None)

-host HOST, --host HOST

Hosts IP address (default: localhost)

-p PORT, --port PORT ETA databus port (default: 4840)

-f, --fullscreen Start visualizer in fullscreen mode (default: False)

-l LABELS, --labels LABELS

JSON file mapping the defect type to labels (default: None)

-d DISPLAY, --display DISPLAY

live preview of classified images(true/false)

(default: true)

-i IMAGE_DIR, --image_dir IMAGE_DIR
directory name to save the images (default: None)

-D DEV_MODE, --dev_mode DEV_MODE

dev_mode can be true or false (default: false)

-P PROFILING_MODE, --profiling_mode PROFILING_MODE

profiling_mode can be true or false (default: false)

To build the visualizer and start it please refer the README.MD of iei-simple-visualizer.
6.0 Working with EIS Docker Containers

EIS sets up the Docker containers using the docker-compose tool. If only minor changes are made in the user-defined functions without adding any new files, the existing Docker files and compose setup will be sufficient to rebuild the EIS. The user can run ‘sudo make build run’ in the docker_setup directory for rebuilding the EIS containers as detailed in the Setup Guide document.

If new files are added or UDF entry points are changed, then the Docker files need to be updated to reflect the same. In the case of new Python dependencies, the corresponding <module>_requirements.txt file needs to be updated.

Other useful commands using the make tool are:
sudo make build : Will build the images for EIS components, however will not start the EIS
sudo make build run : Will build the images for EIS components and starts EIS
sudo make run : Will starts EIS.
sudo make down : Will stop the EIS.
Sudo make distlibs : Dist_Libs provides external client sdk for DataBusAbstraction and ImageStore along with the sample client files.

6.1 Docker setup directory structure:

```
docker_setup/
  └── build_logs.log
  └── build_success_logs.log
  └── clear_linux_setup_guide.md
  └── client-tests-compose.yml
  └── client_tests_startup.sh
  └── compose_startup.sh
  └── config
      └── algo_config
          └── factory_classification.json
          └── factory_multi_cam.json
          └── factory_pcbdemo_hddl.json
          └── factory_pcbdemo.json
          └── factory_pcbdemo_myriad.json
          └── factory_profileting.json
          └── factory_rtsp_cvlc.json
          └── factory_rtsp_hikvision_ds2.json
          └── factory_usb.json
          └── ref_classification
              └── squeezenet1.1_FP16.bin
              └── squeezenet1.1_FP16.xml
              └── squeezenet1.1_FP32.bin
              └── squeezenet1.1_FP32.xml
              └── squeezenet1.1.labels
          └── ref_pcbdemo
              └── model_1.bin
              └── model_1.xml
```

Config files volume mounted inside the container during their startup
6.2 Compose Files

Docker compose is a tool used to define and run multiple-container applications using Docker. It is a wrapper around Docker that is responsible for creating a network and adding the containers to that network. By default, the “bridge” network is used, unless specified otherwise in the `docker-compose.yml` or `docker-compose.yaml` file.
More details about docker-compose and docker-compose.yml file is available at
- Docker compose
- Docker compose CLI
- Docker compose file reference
- Dockerfile reference, Best practices to write Dockerfiles, Docker commands
6.3 Useful Docker* commands

Below is a list of the most used docker-compose commands, which come in handy for development and debugging of the EIS stack:

1. **docker-compose build** - builds all of the service containers. To build a single service container, use `docker-compose build [serv_cont_name]`

   This command generates the following EIS images ("docker images | grep ia | sort" command can be used to see a list of these images)
   
   - ia/data_agent: 1.5
   - ia/data_analytics: 1.5
   - ia/factoryctrl_app: 1.5
   - ia/gobase: 1.5
   - ia/gopybase: 1.5
   - ia/ia_dist_libs: latest
   - ia/imagestore: 1.5
   - ia/logrotate: 1.5
   - ia/provision: 1.5
   - ia/pybase: 1.5
   - ia/telegraf: 1.5
   - ia/video_ingestion: 1.5

   **Note:** If any source file is modified, the `docker-compose build` command must be re-run to see the changes reflected within the container.

2. **docker-compose down** - stops and removes the service containers. Also, removes the Docker network.

3. **docker-compose up -d** - brings up the service containers in detached mode (-d) by picking up the changes made in the `docker-compose.yml` file. It creates a Docker network and joins these containers to it.

   **Note:** To see the list of EIS containers running, use the command: `docker ps`:
   
   - ia/data_agent: 1.0
   - ia/data_analytics: 1.0
   - ia/factoryctrl_app: 1.0
   - ia/imagestore: 1.0
   - ia/logrotate: 1.0
   - ia/telegraf: 1.0
   - ia/video_ingestion: 1.0
4. `docker-compose logs -f` – blocking process to see the Docker logs of all the containers. To see the logs of a single container, you can use `docker logs -f [serv_cont_name]`.

5. To run the Docker images separately, use the command: `docker-compose run --no-deps [service_cont_name].` 
Example: `docker-compose run --name ia_video_ingestion --no-deps ia_video_ingestion`
Will run just the Video Ingestion container (the switch `--no-deps` will not bring up its dependencies configured in the docker-compose file).

If the container is not launching correctly, there could be an issue with the container's entrypoint program which can be overridden by providing the extra switch `--entrypoint /bin/bash` before the service container name in the `docker-compose run` command above. This gives a shell inside the container to run the actual entrypoint program from the container's terminal to determine the root cause of the issue.

If the container is already running, to get a shell inside the container to perform additional operations, use the command: `docker exec [service_cont_name] /bin/bash` or `docker exec -it [cont_name] /bin/bash` 
Example: `docker-exec -it ia_data_analytics /bin/bash`

Note: Now that the gRPC internal secrets are being encrypted, please do not forget to run the command source `/set_shared_key_nonce_env_vars.sh` as this sets the key and nonce environment variables needed for the `docker-compose.yml` file.
7.0 Debugging EIS

7.1 Debugging through the logs

The logs generated by every container is kept at '/opt/intel/iei/logs'. There will be directories named 'classifier_logs', 'consolidatedLogs', 'DataAgent', 'factoryctrl_app_logs', 'telegraf_logs', 'video_analytics_logs', 'video_ingestion_logs'.

Below is the brief about every directory.

classifier_logs: logs of point data analytics.

consolidatedLogs: consolidated logs of all the services.

DataAgent: logs of data agent.

factoryctrl_app_logs: logs of factory control app.

telegraf_logs: logs of telegraph.

video_analytics_logs: logs of video analytics.

video_ingestion_logs: logs of video ingestion.

7.2 Debugging video ingestion through gst-launch

7.2.1 GStreamer valid plugins / element.

In the factory_multi_cam.json file there is attribute named 'video_src' to mention the source of the video frame. For example:

"video_src": "rtsplocatọ́n=rtsp://localhost:8554/ latency=100 ! rtp264depay ! h264parse ! mfxdecode ! videoconvert ! appsink".

In the above example, the highlighted are the gstreamer element/plugins.

The list of valid gstreamer plugin or elements can be found at https://gstreamer.freedesktop.org/documentation/plugins.html?gi-language=c

To know which all plugins are installed, please use the command 'gst-inspect-1.0'. This command will show all the installed plugins.
7.2.2 Debugging the Video ingestion container, in case of ‘video_src’ getting modified by SI.

In case, if SI modifies the value of attribute “video_src” (an attribute from factory.json file), and video ingestion stops working, then use the command ‘gst-launch-1.0’ from inside the video ingestion container. To get the granular logs from this command line utility please set the environment variable named “GST_DEBUG”. The value can range from 1 to 5.

For example. Think of, a video_str attribute is modified to the new value which is having the invalid gstreamer element or plugin, say

```
rtspsrc location="rtsp://localhost:8554/" latency=100 ! rtph264depay ! h264parse123 ! avdec_h264 ! videoconvert ! appsink
```

and the video ingestion starts showing

```
```

```
```

So in this example, ‘h264parse123’ is not valid gstreamer plugin or element.

**So to debug this issue further**, go inside the container (Command to go inside the container: docker exec -it -u root ia_video_ingestion bash)

And set the GST_DEBUG variable ( export GST_DEBUG=3).

Use the below command and look at the logs too.

```
gst-launch-1.0 rtspsrc location="rtsp://localhost:8554/" latency=100 ! rtph264depay ! h264parse123 ! avdec_h264 ! videoconvert ! appsink
```

The highlighted part in the above command is exactly the value of “video_src” attribute. Since, ‘h264parse123’ is not valid gstreamer plugin or element, the above command will give logs with ERROR messages.

```
0:00:00.018350108 85 0xe31750 ERROR GST_PIPELINE grammar.y:716:priv_gst_parse_yyparse: no element "h264parse123"
```

```
0:00:00.018434694 85 0xe31750 ERROR GST_PIPELINE grammar.y:801:priv_gst_parse_yyparse: link has no sink [source=@0xe44080]
```

```
0:00:00.018473162 85 0xe31750 ERROR GST_PIPELINE grammar.y:716:priv_gst_parse_yyparse: no element "avdec_h264"
```

```
0:00:00.018497100 85 0xe31750 ERROR GST_PIPELINE grammar.y:801:priv_gst_parse_yyparse: link has no source [sink=@(nil)]
```
0:00:00.019408613  85  0xe31750 ERROR GST_PIPELINE
grammar.y:801:priv_gst_parse_yyparse: link has no source [sink=@0xe50f30]

WARNING: erroneous pipeline: no element "h264parse123"

So this gst-launch-1.0 command line utility pointed out that there is no element 'h264parse123'. Now it is time to check, is that plugin valid one and is that installed ('gst-inspect-1.0' command can be used to check whether that plugin is installed)

7.3  **Debugging through DEV MODE.**

For more about the developer mode, please the [link](#)
8.0 Known Issues

The current version of the EIS does not support certificate revocation in case of lost or stolen certificates. Full node re-provisioning needs to be done in this scenario as explained in the provisioning section.

High CPU usage is observed with the sample PCB demo trigger algorithm.