# Getting Started with the Industrial Edge Insights Software

The Industrial Edge Insights Software is a scalable reference stack for computer vision and time series data acquisition, analysis and actuation. Follow these steps to begin working with the stack.

You’ll need a system with the following:

* An Intel Core Or Intel Xeon E3 Processor (6th gen or newer)
* At least 4GB Ram
* At least 64GB hard drive (Note: the hard drive will be erased as part of the ISO image flashing process)
* An internet connection

You’ll also need:

* Flash drive (USB 3.0 is preferred) with at least 32GB

We’ve tested this on the following systems:

* iEi TANK\* AIoT Developer Kit
* JWIPC AI-Ready Vision Kit for Smart Manufacturing

By the end of this guide you will understand the following:

* How to flash the operating system to your hardware
* How to run a containerized computer vision demo application and visualize your results
* How to structure and create a Trigger and Classifier for computer vision applications

Let’s get started.

## Step 1: Flash the Operating System

We have created a pre-configured ISO image with the required dependencies and containers to run the Intel Edge Insights Software.

What you’ll learn:

* Flash ISO image to system
* System username and password

By the end of this step you will have a flashed system and be logged into the Ubuntu Desktop.

1. Make sure you have requested access and downloaded the software from the main overview page.

2. Create a bootable USB drive using an imaging application, such as [Etcher](https://www.balena.io/etcher/).

3. If you are using Etcher, follow these steps:

a. Insert your USB drive

b. Select the download file

c. Select your flash drive

d. Flash

**Note:** Since the image is large, the flashing can take over 30 minutes.

4. Insert your USB flash drive into your system. Most systems default to boot from a USB drive. If you see a yellow and orange Clonezilla screen come up, then you know it is booting from the USB. If you are using the iEi TANK 870-Q170 system, and it is not booting from the USB, try the following:

a. Power down (**Note:** to turn the system on hold the power button for a few seconds)

b. Press DEL or F2 as soon as it’s turned on

c. In BIOS go to Boot make the USB drive Boot Option #1

d. Press F4 to save and exit

5. When the system boots from the USB drive you should see a Clonezilla screen. Proceed with the default selections with all the options you will be given. The flashing process can take over 30 minutes (if you have a USB 3.0 drive) and over 5 hours (if you have a USB 2.0 drive).

**Note:** You’ll need to check on the flashing about a quarter of the way through to enter ‘yes’.

5. Power down, then remove your USB drive.

6. Power up, then you should get to the Ubuntu Desktop. The username and password are both ‘ieisw’, though it should login automatically.

7. Before continuing on, connect your system to the internet.

## Run Prerequisites:

Steps to run

* Open the terminal
* Enter following commands

$ ﻿cd ~/IEdgeInsights/cert-tool/

$ python3 main.py ﻿

$ cp -r Certificates/ ../

$ cd ~/IEdgeInsights/docker\_setup/

$ sudo ./provision\_startup.sh ../Certificates/

Generate the certificates again with python file. Copy the certificates to IEdgeInsights folder and run the provision\_startup.sh file to provision the machine with new host IP

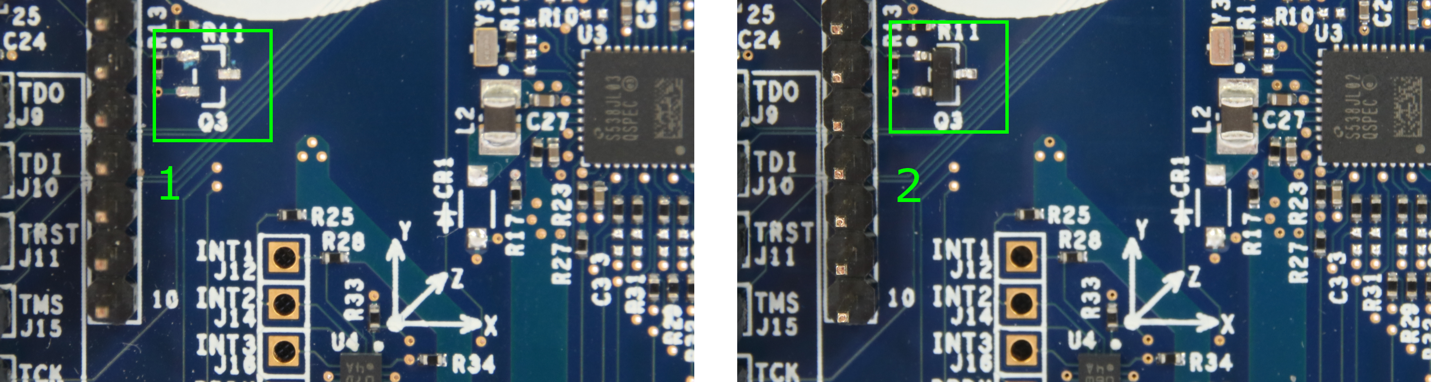
Copy dummy.py from the repository to:

* ~/IEdgeInsights/algos/dpm/triggers/

## Step 2: Run a containerized demo application

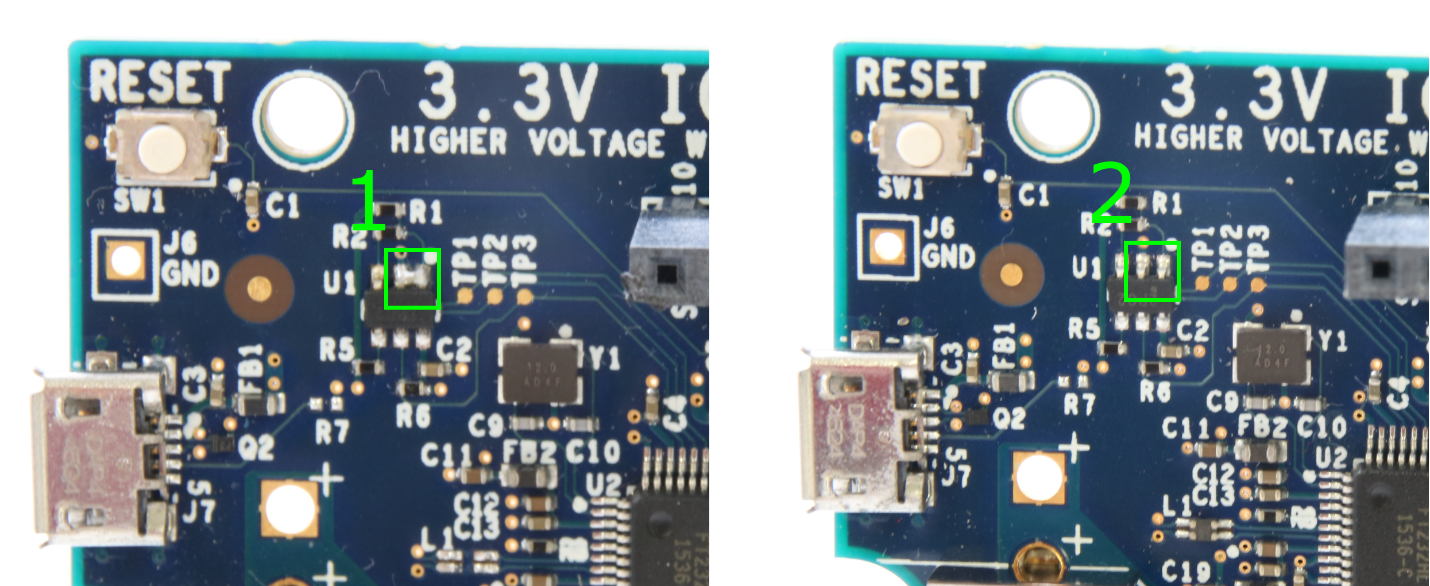
To get familiar with the Industrial Edge Insights Software, you’ll be using a demo application that uses a video of Printed Circuit Boards (PCBs). In this demo scenario, the PCBs are being inspected for quality control. There are two types of defects being detected: a missing component and a short. Below are examples of how the application classifies. The video used in this application is located at **~/IEdgeInsights/test\_videos/pcb\_d2000.avi** . This video shows 3 PCBs rotating through the screen.

Missing component



In the image above, #1 is missing a component, and #2 has the component, so it is normal.

Short



In the image above, #1 has two solder joints connected together that should not be (a short), and #2 is normal.

What you’ll learn:

* How to start the Edge Insights Software
* How to visualize the results of the demo application
* How the application works at a high level

At the end of this section you’ll have a working version of the PCB defect detection application running on your system.

1. Connect your system to the internet. The software requires an IP address for the container network configuration and keys.

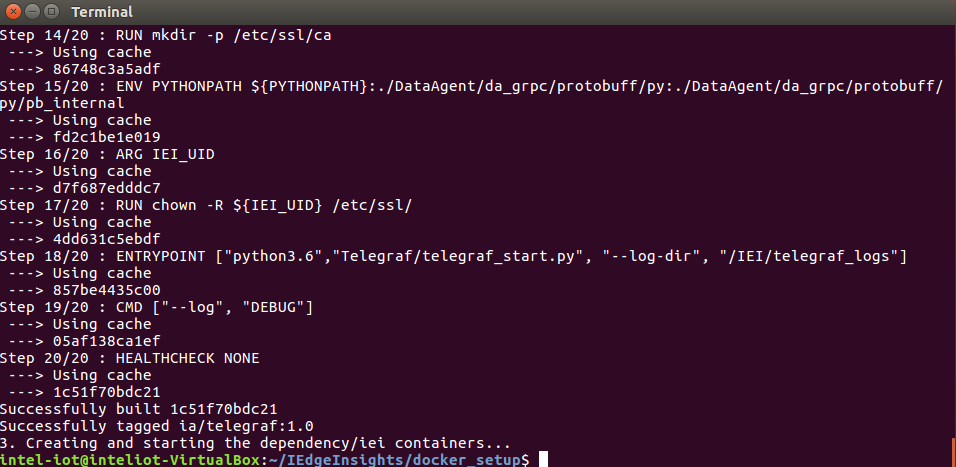
2. Open a terminal and run the following commands:

$ cd ~/IEdgeInsights/docker\_setup

$ sudo ./compose\_startup.sh

**Note:** The root password for this ISO image is the same as the username: devkit

The compose\_startup.sh script will set up the environment variables, generate the security keys and start all the docker containers. You should see something like below



3. To verify if the data pipeline within Edge Insight is working correctly you can check the log files by entering following command,

$ docker logs -f ia\_data\_agent

You should see messages like “Publishing topic: [classifier\_results]”, similar to what is below

A screenshot of a computer

Description automatically generated

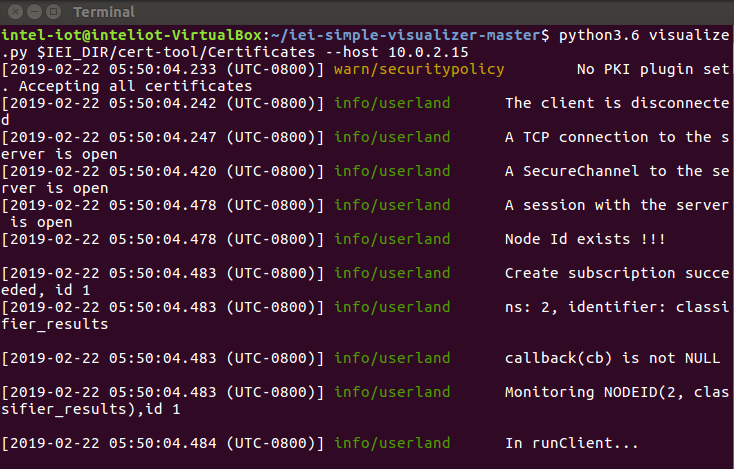
To terminate the running log file, press Ctrl + Z

4. Run the visualizer by entering the following commands:

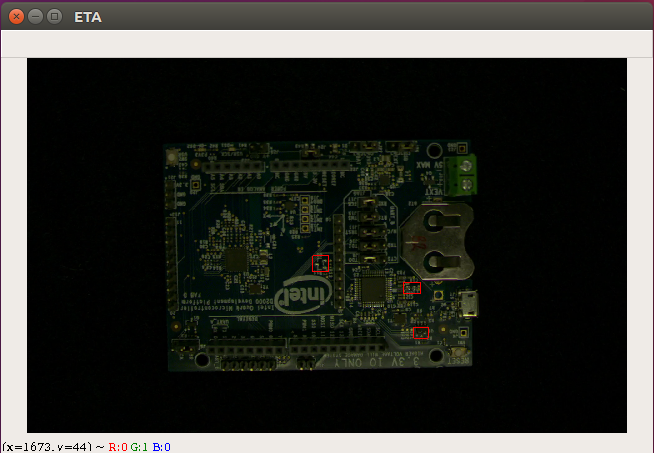
$ **cd ~/iei-simple-visualizer/**

**$ ./run\_simple\_visualizer.sh**

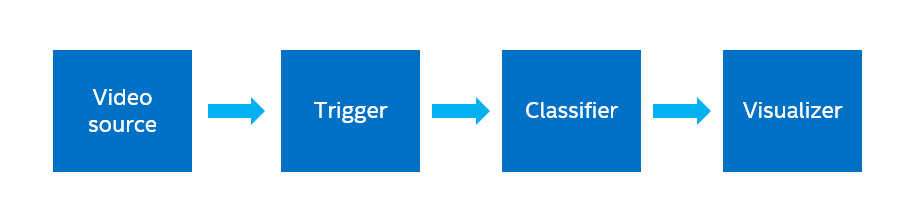
The visualizer source is a Python script which subscribes to data over OPCUA (an industrial protocol) using the databus abstraction library.



The application will open another output window to show the PCB image with defects outlined in red.



To terminate the running python file, press Ctrl + Z in the terminal

So how does the PCB defect detection demo work?

1. The frames in the sample video are sent to the trigger (it’s ok if you don’t know what the trigger is right now we’ll explain it in the next section).
2. The trigger filters out un-needed frames. In this case it is checking that the PCB board is near the center of the frame.
3. Frames of interest are passed on to the classifier, where a deep learning model is used for computer vision inference to determine defects. Defect locations are sent to the database.
4. The visualizer subscribes to the database and displays images with the defects overlaid.

This general flow can be followed for almost any computer vision application. Since the Edge Insights Software comes with the framework of data ingestion, analytics, and database infrastructure, it allows you to focus more on writing really good computer vision algorithms (triggers and classifiers) specific to your needs.

Now you’re ready to get hands-on with the Edge Insights Software.

## Step 3: Create a Simple Trigger and Classifier

The configuration for the Edge Insight Software is located at **~/IEdgeInsights/docker\_setup/config/factory.json**. The factory.json contains the algorithm related configurations for ingestion, triggers and classifiers.

What you’ll learn:

* How to configure the factory.json file to run a new trigger and classifier
* What is inside a simple trigger and classifier

1. Before editing the factory.json we will create the copy of the file,

**$ cd ~/IEdgeInsights/docker\_setup/**

**$ cp config/factory.json config/factory.json.original**

1. Modify the factory.json file by using nano editor

**$ gedit config/factory.json**

|  |
| --- |
| {  "machine\_id": "tool\_2",  "trigger\_threads": 1,  "data\_ingestion\_manager": {  "ingestors": {  "video\_file": {  "video\_file": "./test\_videos/pcb\_d2000.avi",  "loop\_video": true  }  }  },  "triggers": {  "**dummy\_trigger**": {  }  },  "classification": {  "max\_workers": 1,  "classifiers": {  "**dummy**": {  "trigger": ["**dummy\_trigger**"],  "config":{  }  }  }  }  } |

You’ll notice the ./test\_videos/pcb\_d2000.avi file listed in the “data\_ingestion\_manager”. This section is where camera or video feeds are configured.

The “triggers” and “classifiers” sections can include user defined parameters, though for the simple trigger and classifier those aren’t needed.

It’s important to note that the trigger and classifier names need to match the names of the Python files, but without the .py extension. For example, dummy\_trigger > dummy\_trigger.py .

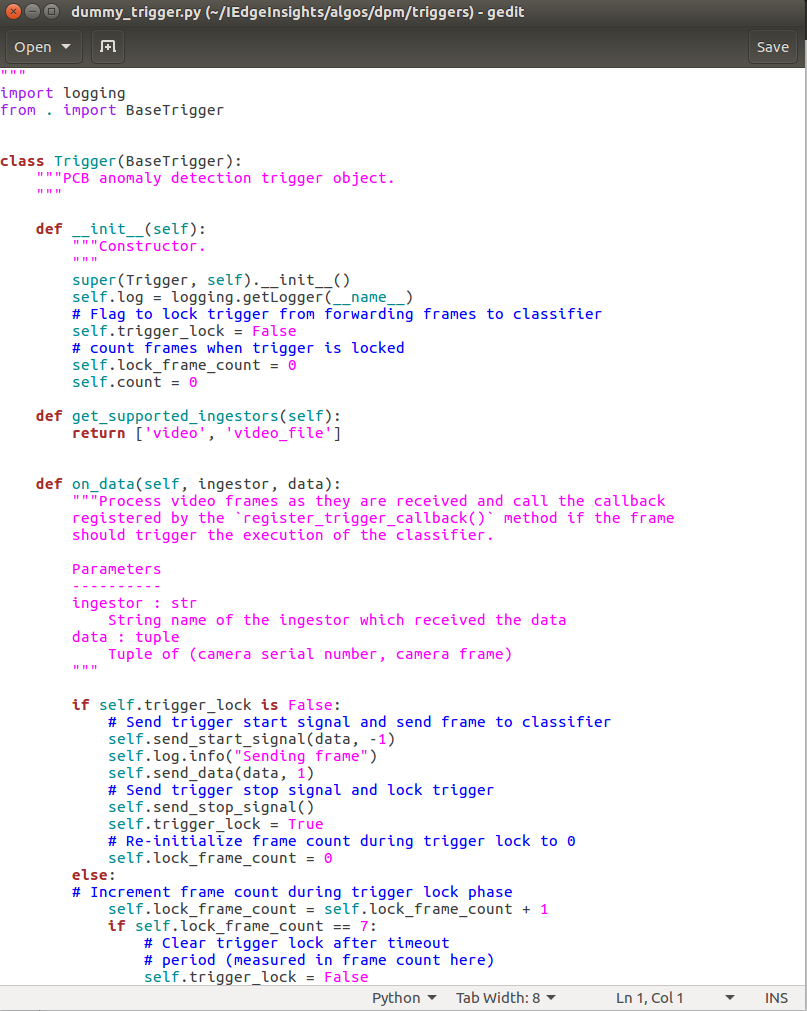
Save the file with Ctrl + S and then close the gedit editor.

1. All the algorithm files are located at ~/IEdgeInsights/algos/dpm/ folder. Open the dummy\_trigger file from ~/IEdgeInsights/algos/dpm/triggers/dummy\_trigger.py

**$ ﻿cd ~/IEdgeInsights/algos/dpm/triggers**

**$ ls**

**$ gedit dummy\_trigger.py**



The goal for a trigger is to process the incoming data from the camera and determine whether or not a classification period should be started by Edge Insight. Once a classification cycle is started, it is the triggers job to pass on the incoming video frames

to the classification, and then to signal software stack when the classification is finished.

In this file, the algorithm is taking the image from video ingestion container and sending it to data agent to be received by classifier without doing any filtering.

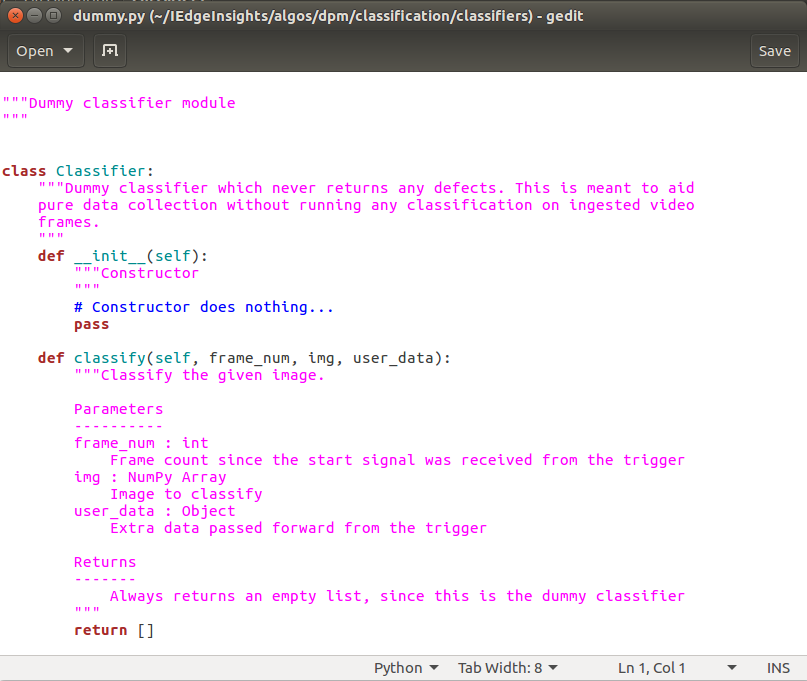
After understanding the trigger class, you can close the file without any modification.

1. Open the dummy classifier file at location ~/IEdgeInsights/algos/ dpm/classification/classifiers/dummy.py

**$ cd ~/IEdgeInsights/algos/dpm/classification/classifiers/**

**$ ls**

**$ ﻿gedit dummy.py**



Classifiers contain the different algorithms to use on the video frames that are ingested into the agent. The “classification” key in the factory.json determines which classifier is used in association with which trigger.

The dummy.py classifier is an empty classifier which will take the image and just return an empty list without any analytics.

After understanding the classiifer class, you can close the file without any modification.

You can learn more about the inputs, outputs, and structure of the class in the Developer Guide.

1. Build the containers again with the new triggers and classifiers.

$ cd ~/IEdgeInsights/docker\_setup

$ sudo ./compose\_startup.sh

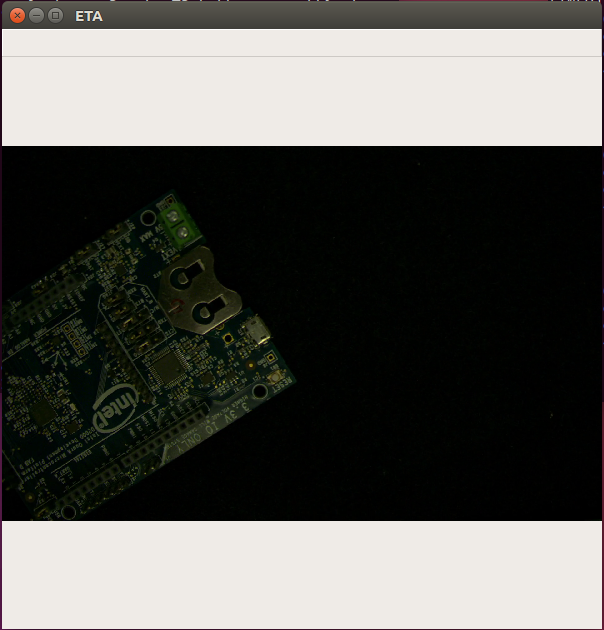
This should take 2-3 minutes while the video analytics and video ingestion containers are re-built.

1. Start the visualizer

$ cd ~/iei-simple-visualizer/

$ ./run\_simple\_visualizer.sh

The result shows the dummy trigger and dummy classifier implementation where the trigger passes through every frame, and the empty classifier returns zero defects. You should see the PCB boards pass through the screen at odd angles with no defects.



To terminate the running python file, press Ctrl + Z in the terminal

## Next Steps

Now that you have an understanding of how to create a simple trigger and classifier, explore additional tutorials and guides to gain a deeper understanding of the software.

Tutorials:

How to bring your computer vision application into the Edge Insights Software

How to use Intel Vision Accelerator Products in Edge Insights Software

How to port your application running in Linux into a Docker container

Reference Implementations:

Object Flaw Detector – Product quality inspection using computer vision

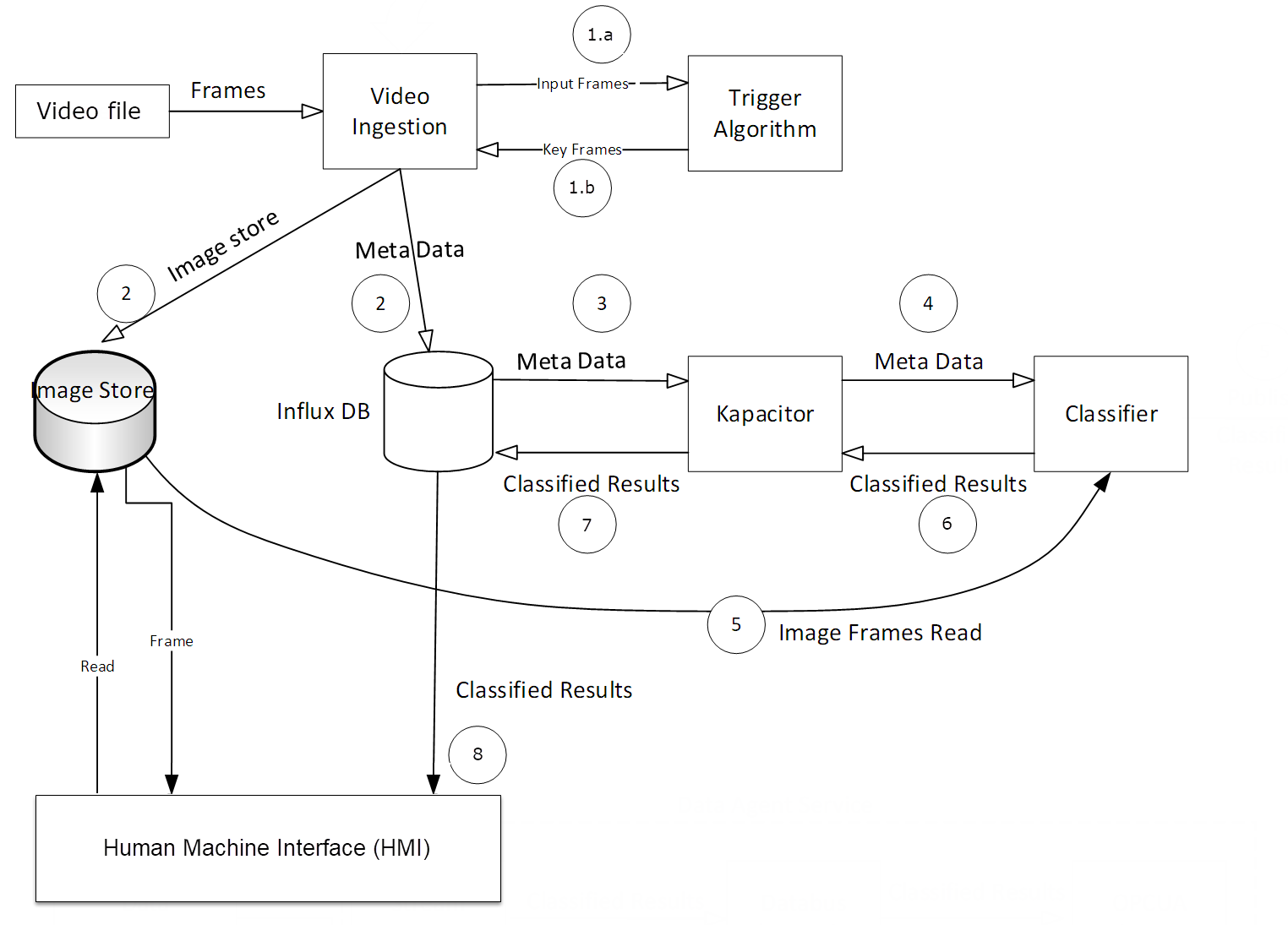
Motor Fault Detector – Condition monitoring of equipment to reduce downtime

Developer Guide

## Appendix: Reference Material

### Intel Edge Insights - Architecture Introduction

Here is how the Intel Edge Insights framework is structured for an application in general.



The configuration of an application is controlled by a json file (factory.json below). It is not shown the diagram above.

**factory.json**

Location: ~/IEdgeInsights/docker\_setup/config/factory.json

Purpose: This file is the main configuration file for the entire work stream. In it, a user can define the data ingestion, storage, triggers, and classifiers. It uses a standard JSON format.

**Trigger**

Location: ~/IEdgeInsights /algos/dpm/triggers/

Purpose: The trigger Python script is a filter for the incoming data stream, mainly to reduce the storage and computation requirements by only passing on frames of interest. All input frames are passed to the Trigger (**1.a** in the diagram above). When it detects a frame of interest based on user defined functions, it activates a signal (**1.b**) which causes that frame to be saved in the Image Store database (**2**), and the metadata for that frame in the InfluxDB database (**2**).

**Classifier**

Location: ~/IEdgeInsights /algos/dpm/classification/classifiers/

Purpose: The classifier is a user defined algorithm that is run on each frame of interest. Kapacitor , an open source data processing engine, subscribes to the meta-data stream (**3**), and the classifier receives the meta-data from Kapacitor (**4**). The classifier pulls the frame from the Image Store (**5**), and saves the analysis results as metadata back into the InfluxDB database (**6 and 7**).

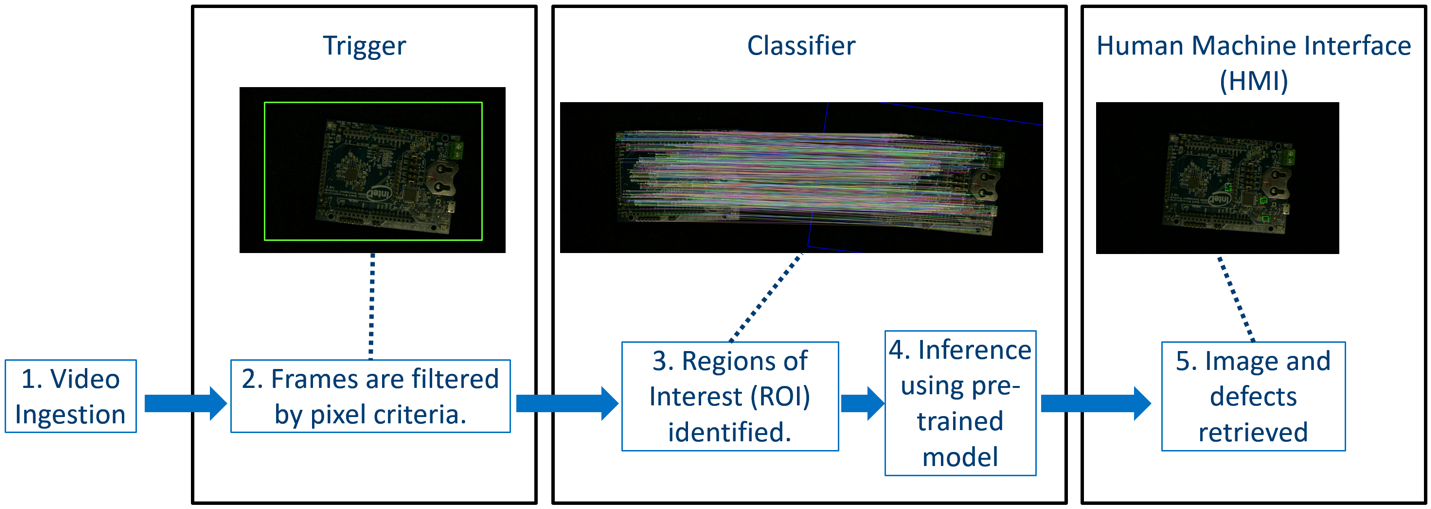
It’s important to note that as an application developer, you do not need to worry about handling the data flow described above from data ingestion to classification. The included software stack using InfluxDB and Kapacitor handle the data movement and storage. In fact, you do not need to specifically include read and write functions to the databases in the Trigger and Classifier scripts, since that is already included in the base class for you.

In this demo, the Human Machine Interface (HMI) that you see in the web browser is pulling the results from the Image Store, and matching them to the InfluxDB database with the classified results (**8**), and overlaying the bounding boxes on top of the detected defects.

**PCB Demo Application Details**

This application uses a pre-trained deep learning model that was trained with MobileNet. Before doing classification of the image, the image is aligned and regions of interest are identified using an overlay. The regions of interest are used by the deep learning model to classify the defect as either a missing component, short, or normal. This sample utilizes a video file of 3 PCBs placed on a rotary table running at 30rpm.

Below is a more detailed description of what is going on in this demo application.



**1. Video Ingestion –** All frames are passed to the trigger. In this case it is a sample video that is located in ~/ETA/IEdgeInsights /test\_videos/ folder.

**2. Frames are filtered by pixel criteria –** The PCB trigger **(~/IEdgeInsights/algos/dpm/triggers/pcb\_trigger.py**) checks for a rectangle shape within specified pixel area, as defined in factory.json. If the frame meets the criteria, it is passed on by a signal function, which causes that particular frame to be saved in the Image Store and the metadata to be stored in the Data Store (InfluxDB).

**3. Regions of Interest (ROI) identified –** The PCB classifier (**~/IEdgeInsights/algos/dpm/classification/classifiers/pcbdemo/\_\_init\_\_.py**) is activated by a Kapacitor container running in the background. When new data is entered into InfluxDB, the Kapacitor container activates the classifier script and passes it the data for that frame. At that point a feature match between the reference and test image is done to obtain the defect Regions of Interest (ROI). The blurry image you see in the diagram is a pictorial representation of the feature matching being done in OpenCV.

**4. Inference using pre-trained model –** The model (**~/IEdgeInsights/docker\_setup/config/ref/model\_1.xml**) used in this application is a MobileNet model that has been trained for this dataset. The inference is being done using the OpenVINO™ Toolkit Inference Engine.

**5. Image and defects retrieved –** The HMI is notified via MQTT of new data, and it overlays the bounding boxes of the defects on top of the original image, which it obtained from the Image Store.

**Visualizer - Explanation**

The Visualizer is a data sync application which uses python binding of ImageStore gRPC module to fetch the classified images from the ia\_image\_store container and python binding of DataBusAbstraction module to get the metadata of the classified messages from the DataAgent module via OPC-UA.

1. gRPC Module – Google Remote Procedure Call Module

gRPC is a high-performance, open-source RPC framework which uses Protocol Buffer. This protocol helps the containers to talk with each other via network.

1. OPC-UA – Open Platform Communications – Unified Architecture

OPC is the interoperability standard for secure and reliable data exchange which offers service oriented architecture with security and scalability.

### Docker Technology

The Edge Insight software stack uses the Docker technology to package the application and used Docker Compose tool to execute multi container applications.

According to the Docker website, “Docker allows you to package an application with all of its dependencies into a standardized unit for software development.” Docker is a platform for developers, system admins to develop, ship and run applications. Docker Engine manages the images and the containers.

Some terms to be aware of include:

* A *container* is a runtime instance of docker image. It consists of docker image, execution environment and standard set of instructions.
* An *image* is ordered collection of execution parameters, dependencies, filesystem changes to execute the application.
* *Docker\* Compose* is a tool for defining and running multi-container Docker applications

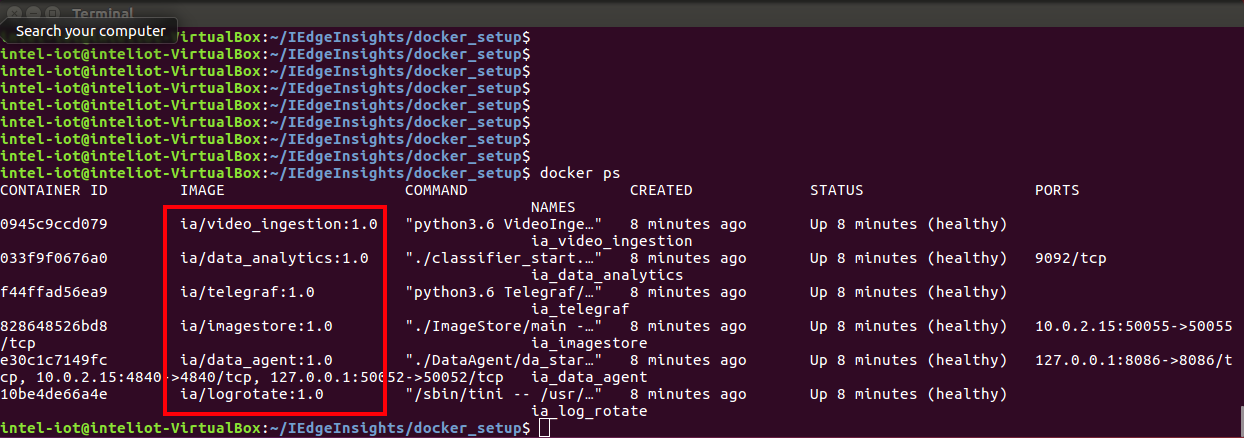
**Useful docker-compose and docker commands**

* docker-compose build - builds all the service containers. To build a single service container, use docker-compose build [serv\_cont\_name]
* docker-compose down - stops and removes the service containers
* docker-compose up -d - brings up the service containers by picking the changes done in docker-compose.yml
* docker ps - check running containers
* docker ps -a - check running and stopped containers
* docker stop $(docker ps -a -q) - stops all the containers
* docker rm $(docker ps -a -q) - removes all the containers. Useful when you run into issue of already container is in use.
* [docker compose cli](https://docs.docker.com/compose/reference/overview/)
* [docker compose reference](https://docs.docker.com/compose/compose-file/)
* [docker cli](https://docs.docker.com/engine/reference/commandline/cli/)

**About the containers within the Industrial Edge Insights Software**

Enter following command to verify all the running docker containers.

$ docker ps



* Dependency containers:
  + log rotate (ia/log\_rotate) – Container to automatically rotate, compress and periodically prune log files.
* IEI core containers:
  + DataAgent (ia/data\_agent) – Container responsible for initializing the stream manager to listen to the data and starting gRPC servers.
  + Image Store (ia/imagestore) – Container responsible for image storage, retrieval, and removal with respect to in-memory and file system.
  + Video Ingestion (ia/video\_ingestion) – Container takes the video (continuous images) and send it to database.
  + Data Analytics (ia/data\_analytics) – Container runs the classifier algorithm to detect defects in PCB.
  + Telegraf (ia/telegraf) – Container used for data ingestion