# Hackathon Code

## Introduction

This document describes the foundational code provided to the Sportsnet Hockey hackathon participants to get them started. The code consists of 4 folders, which are described below.

## Message Broker

This section contains the python files to show how to send and receive messages from a message broker. The chosen message broker is RabbitMQ and the tutorial for this code can be found [here](https://www.rabbitmq.com/tutorials/tutorial-one-python.html). The send.py and receive.py are like the ones in the tutorial. We recommend reviewing this [Publish and Subscribe tutorial](https://www.rabbitmq.com/tutorials/tutorial-three-python.html).

receive\_video.py connects to the message broker and consumes from the ‘video-feed’ queue. The ‘video-feed’ queue gets filled by running the VideoCapture/server.py file.   
receive\_video.py does the decoding of the message in the message broker and displays one frame at a time using [opencv](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_image_display/py_image_display.html).

Docker-compose file is consolidated into the main one that is used for the whole project, but this one can help you understand better the RabbitMQ instance of the application. The files under etc folder help configure RabbitMQ. In our case the relevant part is in the queues section of the definitions.json file. There we list 3 queues:

* video-feed: contains each frame of the video feed.
* video-openvino: contains each processed frame of the video feed. More on this in the Openvino section.
* puck-tracker: contains each data point from the json file for the puck tracker data. More on this in the Ppt section.

## Openvino

* + The openvino folder contains all the files necessary to run an Openvino inference (image analysis) on an image to detect and frame bodies in the picture. Inside of the data folder there are files that contain pre-trained models of machine learning algorithms that can detect bodies. Inside of the openvino\_server.py we use the flask framework to create a server that uses the pre-trained models on an “images” endpoint on port 5000 (http://<computer-ip-address>:5000/images). Once an image is posted in base64 format (without the encoding: data:image/jpeg;base64,) it will then be returned to the user with the bodies identified as well as a copy to be stored in the Rabbitmq Message Broker.

## Ppt

This section shows how to get the puck tracker data contained in the json files into the message broker and how to retrieve the data. Emit\_log.py sends 10 rows of MetaStats.JSON to the message broker puck-tracker queue. receive\_logs.py receives all messages in the puck-tracker queue and prints its content to the console.

## VideoCapture

* + In the video capture folder you will find a file called server.py in this file we are using a plugin called opencv which reads the video file frame by frame. In this application we are defining the video file and inserting that into the video capture method of opencv which reads the file so we can loop over each frame to read it, resize it, convert it to a jpeg in base 64 format, and send its information via another plugin called pika to the rabbitmq message broker for later access.

