Containerization using WSL for Integrated Retail Solution

White Paper

June 2023

Document Number: 782229-0.1

Contents

Figures

Tables

Revision History

Date	Revision	Description
June 2023	0.1	Initial release.

§

1.0 Introduction

In the era for multi-workloads consolidations, customers need both agile resource allocation and ensure improved security. These requirements are to protect business critical workloads and adequate compute resources are being allocated for various workloads.

Historically, retail and hospitality industries deployed many different form-factors of Points of Sale (POS) and kiosk devices, such as desktop POS, mobile POS, self-checkout kiosk, Quick Service Restaurant (QSR) kiosk, and digital signage devices that mainly focus on singular workload (for example: Point of Sale application) in the stores. However, demands for other workloads like digital signage, digital surveillance, online in-store kiosk for offline pick-up, and product recognition using computer vision are now increasing.

Workload consolidation unites multiple computerized operations onto fewer systems replacing separate purpose-built hardware machines such as POS, kiosks, digital signages and others with a smaller foundation of general-purpose compute technologies, while maximizing compute utilization.

This paper explains the concept of workload aggregation and consolidation using containerized solution for retail use cases (POS, Surveillance, Digital Signage), including architectural considerations as well as enabling techniques to support common workloads required in the retail and hospitality industries.

1.1 Terminology

Table 1. Terminology

Term	Description
AI	Artificial Intelligence
BKM	Best Known Method
Cmd	Command
CRM	Customer Relationship Management
DP	DisplayPort
EFLOW	Edge for Linux on Windows
ERP	Enterprise Resource Planning
GPU	Graphics Processing Unit

Term	Description
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
iGPU	Integrated Graphics Processing Unit
IT	Information Technology
LXC	Linux Containers
NUC	Next Unit of Computing
OpenVINO	Open Visual Inference and Neural network Optimization
OS	Operating System
POS	Points of Sale
QSR	Quick Service Restaurant
USB	Universal Serial Bus
VM	Virtual Machine
WSL	Windows* Subsystem for Linux*

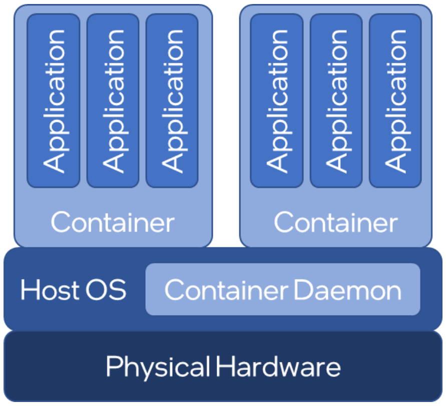
1.2 Introduction to Containerization

Containerization is a modern software approach to deploy various workloads in the current Information Technology (IT) world. Containers share the kernel of their host with other containers running on the system and include just enough of the operating system, the application itself, and its dependencies. Due to the way containers are structured and share a kernel, containers are very light weight and relatively smaller in size compared to other consolidation techniques such as Virtual Machines (VMs).

A high-level example of containerized solution is shown in Figure 1. Typical system includes physical hardware running desired operating system, which is described as host OS. The container daemon runs at same level as host OS to facilitate execution of various containers with desired applications. Some examples of container daemon or software available today are Docker*, Linux Containers (LXC), and Apache Mesos*.

For the scope of this document, we will discuss the concept of integrated retail solution and how containerized solution can support additional workloads like AI analytics and digital signage in the same POS hardware.

Figure 1. Generic Containerization Technique for Workloads Consolidation



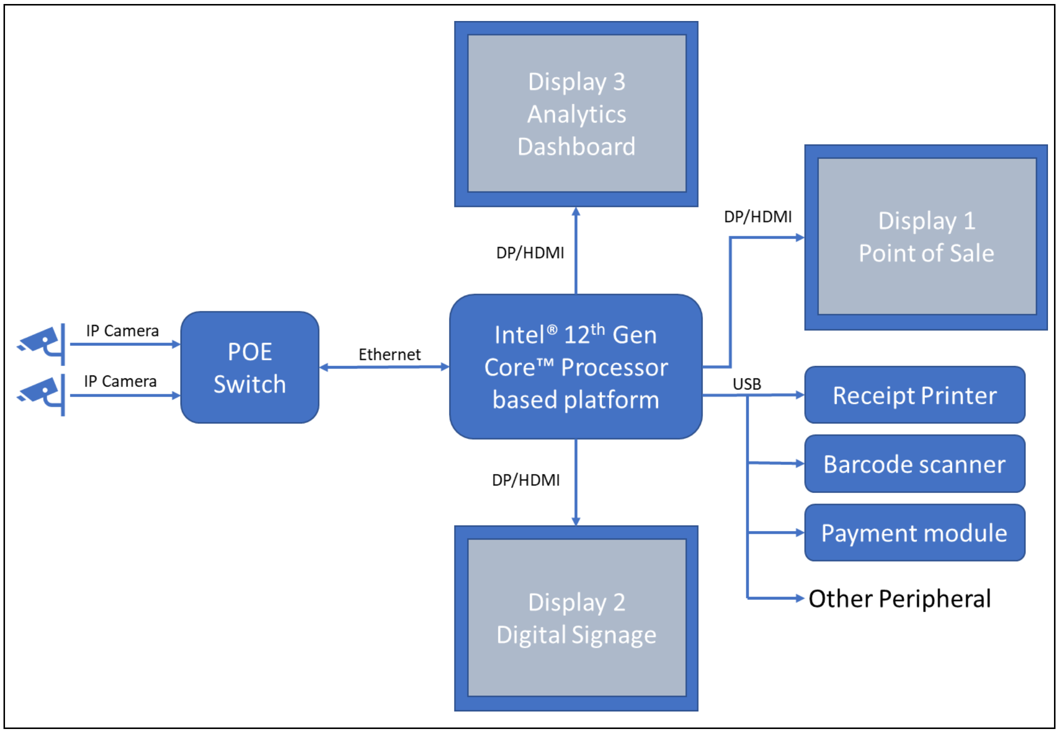
§

2.0 Solution Implementation

In this section of the document, the hardware and software architecture of integrated retails solution with Point of Sale as primary device is described. Figure 2 depicts the typical system architecture of POS, Digital Signage and Analytics included in Integrated Retails solution based on Intel® NUC based 12th Gen Core™ processor. The system is connected with various interfaces such as USB, Serial Port, Ethernet, and HDMI targeting for POS usage model.

Leveraging the latest 12th Gen Intel® Core™ processor architecture which is capable of supporting four displays simultaneously to expand new workload applications, for example, digital signage, analytic dashboard, and digital surveillance.

Figure 2. System Architecture for Integrated Retail Solution



Point of Sale device is essential for all grocery stores, especially for small Mom and Pop stores. For an integrated retail solution in a single box, it is important to prioritize the workload of traditional POS and ERP/CRM related workloads than other types of workloads. The primary OS required for Point of sale is Windows*. However, the Analytics various AI solutions still available as container for Linux operating system. Thus, we need container-based approach to run additional workloads.

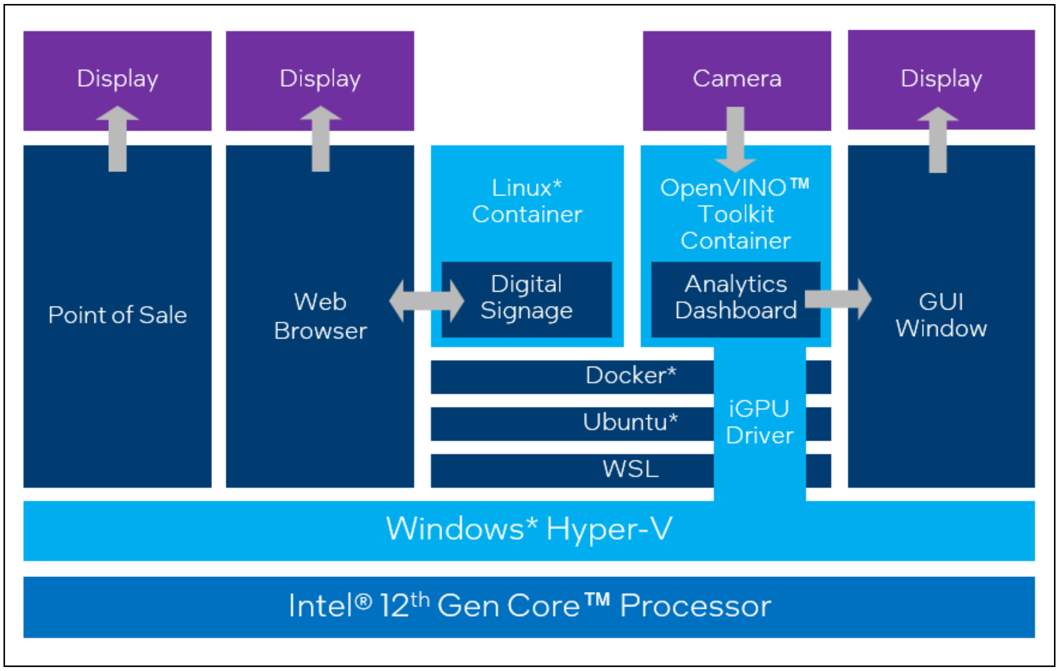
Furthermore, the containerization approach allows the allocation of dedicated resources for POS and assign resources separately for other applications such as AI analytics and digital signage. Figure 3 shows the software architecture of such containerized applications.

As shown in Figure 3, the Point-of-Sale application runs directly on the host operating system which is Windows* in this case. Besides, there are two other key containers as follows.

1. Digital signage server application is a containerized application that controls the signage content and its management. Once the server application is started, digital signage content can be played back on the device using web browser.
2. Similarly, OpenVINO™ based analytics for person detection is also a container-based application. This OpenVINO™ container receives images from a camera and performs AI inference, for example, person detection in which can be utilized for various insights to retail business.

Both containers are Linux* containers.

Figure 3. Isolated Workloads Run on Docker* on Top of WSL



Thus, to execute this container it requires Linux* as the host operating system. However, as described before POS is primarily a Windows*-based solution thus to use these containerized applications, first Windows Subsystem for Linux (WSL) is used with Windows* Hyper-V. Then, an Ubuntu* OS is installed on top of WSL. Next ingredient required to execute is container daemon. Docker* is a widely used container daemon, which is also used in this example to run the container described earlier for digital signage content management application and OpenVINO™ AI analytics container. This implementation allows the use of integrated Graphics Processing Unit (iGPU) for

Docker* container hardware acceleration in both digital signage playback as well as AI inference.

Further in later chapter, we will describe the procedure and the best known method (BKM) to install WSL and Ubuntu* along with the example of containerized applications for digital signage server as well as OpenVINO™ based analytics dashboard.

Note that this document is intended for developer communities, thus it is to be expected that consumers of this document are familiar with basic knowledge of Docker*, virtualization, standard Windows* and Linux* commands, as well as installing various dependencies and packages.

§

3.0 Procedure and BKM for WSL-Based Integrated Retail Solution

As described previously, Point of Sale application is installed directly on Windows* host operating system. Thus, this chapter focuses primarily on installing and running two containerized applications of digital signage content management as well as OpenVINO™ AI analytics.

For ease of understanding following are the high-level steps for the system:

1. Install WSL to set up Linux* operating system with access to iGPU of Intel® processors.
2. Install Ubuntu* as guest operating system on top of WSL. Ubuntu* will be the host OS for containerized applications using container daemon.
3. Install Container Daemon using Docker* to run containerized applications.
4. Build and Run OpenVINO™ AI Analytics GUI Application to receive camera feed and run AI inference. On top of that, to visualize the output, we need to install X server application on Windows* side to capture that.
5. Install and Run Digital Signage Server Application to run digital signage management server. For the demo purpose, we have used an open-source application known as Xibo Signage*.

3.1 Prerequisites

The setup needs WSL version 2, and the installation requires Windows* administrator privileges. Check WSL 2 installation page for more detailed requirements.

3.2 Install WSL

Run Cmd 1 as an administrator to install WSL on Windows* PowerShell or Command Prompt.

Note: **Linux* will have difficulty navigate through directories and file names with spaces. Create username as well as directories/filenames that are friendly for both Windows* and Linux*.**

Cmd 1. Command to Install WSL

```
> wsl --install

> wsl --version
WSL version: 1.2.5.0
Kernel version: 5.15.90.1
WSLg version: 1.0.51
```

```
MSRDC version: 1.2.3770
Direct3D version: 1.608.2-61064218
DXCore version: 10.0.25131.1002-220531-1700.rs-onecore-base2-hyp
Windows version: 10.0.22000.1817
```

3.3 Install and Run Linux*

Install Ubuntu* 22.04 LTS from Microsoft* Store. Run the installed Linux* from the Start menu and update the packages using Cmd 2.

Cmd 2. Command to Update Ubuntu*

```
$ sudo apt update
$ sudo apt upgrade
```

3.4 Install Container Daemon

Use Cmd 3 to activate Linux* `systemd` on WSL.

Cmd 3. Command to Activate Linux* `systemd` on WSL

```
$ sudo vim /etc/wsl.conf
[boot]
systemd=true
```

Then use Cmd 4 to install Docker*.

Cmd 4. Command to Install Docker*

```
$ sudo apt update
$ sudo apt install docker.io -y
$ sudo usermod -aG docker $USER
$ docker --version
Docker version 20.10.21, build 20.10.21-0ubuntu1~22.04.3
$ exit
```

Restart WSL and install Docker* Compose using Cmd 5.

Cmd 5. Command to Install Docker* Compose

```
> wsl --shutdown

$ sudo apt install docker-compose
```

3.5 Install X Server for Windows*

Install X Server (VcXsrv) on Windows* using the following link. Make sure to disable access control option. This application is required to play back output of OpenVINO™ AI analytics as GUI.

- Install X Server for Windows*: https://medium.com/@potatowagon/how-to-usegui-apps-in-linux-docker-container-from-windows-host-485d3e1c64a3

3.6 Build OpenVINO™ Analytics Containerized Application

Use person detection demo from OpenVINO™ Model Zoo. Build the image using `Dockerfile` in Cmd 6.

Cmd 6. Person Detection Demo Dockerfile

```
$ cat Dockerfile
FROM openvino/ubuntu20_dev:2022.3.0

# Install basics
USER root
RUN apt-get update -y \
&& apt-get install -y libgtk2.0-dev libcanberra-gtk-module
libcanberra-gtk3-module \
&& apt-get install -y ffmpeg

# Create new user
RUN groupadd -g 1001 tami
RUN useradd -rm -d /home/tami -s /bin/bash -g 1001 -u 1001 tami
USER tami

# Get video files
RUN mkdir -p /home/tami/videos
WORKDIR /home/tami/videos
RUN curl -O
https://storage.openvinotoolkit.org/data/test_data/videos/store-
aisle-detection.mp4

# Install Open Model Zoo
WORKDIR /home/tami
RUN git clone --recurse-submodules
https://github.com/openvinotoolkit/open_model_zoo.git
WORKDIR /home/tami/open_model_zoo
RUN git checkout 2022.3.0
WORKDIR /home/tami/open_model_zoo/demos/
RUN ./build_demos.sh

# Setup Object Detection Demo
WORKDIR
/home/tami/open_model_zoo/demos/object_detection_demo/python
RUN omz_downloader --name person-detection-retail-0013
```

```
# Finalizing
WORKDIR /home/tami
```

Build the image using Cmd 7.

Cmd 7. Command to Build Person Detection Demo

```
$ sudo docker build \
  -f Dockerfile \
  -t open-model-zoo:2022.3.0 .

$ sudo docker images
REPOSITORY               TAG          IMAGE ID    CREATED    SIZE
open-model-zoo           2022.3.0     xxx         xxx        5.24GB
openvino/ubuntu20_dev    2022.3.0     xxx         xxx        4.35GB
```

3.7 Run OpenVINO™ Analytics Containerized Application

Run X Server (VcXsrv) on Windows* with access control option disabled. Then, get Windows* host IP using Cmd 8.

Cmd 8. Command to Get Windows* Host IP

```
> ipconfig
...
IPv4 Address. . . . . . . . . . . : 192.168.x.x
...
```

Open Ubuntu* and execute Cmd 9 to run the person detection demo. Details for the argument's selection are as follows:

- `-itu root:root` : To run as a super user. To access the GPU, the user needs to be a super user. For CPU run, this is optional.
- `--net host` : Set network to follow the host for the container to access host's display server application.
- `-e DISPLAY=192.168.x.x:0.0` : This is for display setting as well as the access to display server. Use the host IP address from Cmd 8.
- `--device /dev/dxg` : To access GPU driver. This is optional for CPU run.
- `--volume /usr/lib/wsl:/usr/lib/wsl` : Grant the container access to the WSL 2 libraries.
- `/bin/bash -c` : Command to execute the demo application.

Cmd 9. Command to Run Person Detection Demo

```
$ docker run \
  -itu root:root \
  --net host \
  -e DISPLAY=192.168.x.x:0.0 \
  --device /dev/dxg \
  --volume /usr/lib/wsl:/usr/lib/wsl \
  --rm open-model-zoo:2022.3.0 \
  /bin/bash -c "cd /home/tami && sudo python3
./open_model_zoo/demos/object_detection_demo/python/object_detect
ion_demo.py -at ssd -t 0.95 -d GPU --loop \
  -i ./videos/store-aisle-detection.mp4 \
  -m
./open_model_zoo/demos/object_detection_demo/python/intel/person-
detection-retail-0013/FP16-INT8/person-detection-retail-0013.xml"
```

The GUI application will be shown in the opened X Server application.

3.8 Install Digital Signage Server Containerized Application

This example is explained using open source Xibo Signage* application. Open Ubuntu*, download Xibo Signage* for Linux*, and extract it using Cmd 10. Rename the directory to avoid path issue.

Cmd 10. Command to Extract Xibo Signage*

```
$ tar -xzvf xibo-docker.tar.gz
$ mv xibo-docker-x.x.x xibo
$ cd xibo
```

Copy `config.env.template` file to `config.env` and edit it as needed.

3.9 Run Digital Signage Server Containerized Application

Go to `xibo` directory and run Xibo Signage* using Cmd 11.

Cmd 11. Command to Run Xibo Signage*

```
$ docker-compose up -d
```

When running it for the first time, Docker* Compose will download all the required images as shown in Cmd 12.

Cmd 12. Command to List Xibo Signage* Images

```
$ docker images
REPOSITORY             TAG          IMAGE ID   CREATED   SIZE
mysql                  5.7          xxx        xxx       455MB
memcached              alpine       xxx        xxx       9.42MB
xibosignage/xibo-cms   release-x    xxx        xxx       575MB
ianw/quickchart        latest       xxx        xxx       964MB
xibosignage/xibo-xmr   0.9          xxx        xxx       29.5MB
```

Open a web browser to access the server containerized application from `http://127.0.0.1:80` IP and port as well as the default `xibo_admin` username and `password` password.

The commands in Cmd 13 may be useful to stop, start, and uninstall the application.

Cmd 13. Optional Commands to Stop, Start, and Uninstall Xibo Signage*

```
$ docker-compose stop

$ docker-compose start

$ docker-compose down
```

§

4.0 Resource Allocation for Containerized Applications on WSL

As mentioned earlier, it is important to control the resources allocated to POS and any other containerized applications in the system. The following subsections show the way to allocate resources for the containerized applications that use either Docker* or Docker* Compose.

4.1 Resource Allocation When Using Docker*

For applications that use Docker* command to run, `--cpus` and `--memory` options can be used to limit the number of CPU and memory utilized by the application. For example, Cmd 14 will be run with only two CPUs.

Cmd 14. Command to Run Docker* Application with Limited Number of CPUs

```
$ docker run \
  --cpus="2.0" \
  -itu root:root \
  --net host \
  -e DISPLAY=192.168.100.7:0.0 \
  --device /dev/dxg \
  --volume /usr/lib/wsl:/usr/lib/wsl \
  --rm open-model-zoo:2022.3.0 \
  /bin/bash -c "..."
```

4.2 Resource Allocation When Using Docker* Compose

For applications that uses Docker* Compose, add `cpus` as well as `mem_limit` keys in the `docker-compose.yml` file. Cmd 15 shows the way to limit the number of CPUs utilized by `cms-web` services to 1. Note that the limit for the other services will need to be set individually when needed.

Cmd 15. docker-compose.yml Example to Limit Number of CPUs for Each Service

```
$ cat docker-compose.yml
version: "2.1"
services:
    cms-db:
    cms-xmr:
    cms-web:
        cpus: 1.0
```

§

5.0 Test Setup, Results, and Summary

5.1 Test Setup

This chapter is intended to provide the real setup implemented with the BKM mentioned in the previous sections. Figure 4 is the picture of realistic setup for such Integrated retail solution. Table 2 shows the major system blocks and ingredients for the setup.

Table 2. System Blocks and Ingredients

Component	Details
Processor	Intel® Core™ i7 12800HL, 45 W (ADL PS) with Intel vPro® 6P+8E Cores, 20 Threads 24 MB Intel® Smart Cache 2.4 GHz Base Frequency with up to 4.8 GHz Turbo Frequency
Memory	64 GB SODIMM DDR5-4800, Dual Channel x 64-bit (Total 128-bit Memory Interface)
Storage	128 GB M.2 PCIe NVMe
External Cameras	POE Camera: HIKVISION* DS-S2CD1653G0-IZ Wi-Fi Camera: Sricam* SH034
External POE Switch	5-Port Gigabit Desktop Switch with 4-Port POE+ TL-SG1005LP
Host Operating System	Microsoft* Windows* 10 version 10.0.22000.1817
Graphics Driver	Intel® Iris® Xe Graphics 31.0.101.3959
WSL	WSL 2 version 1.2.5.0
Linux* OS	Ubuntu* version 22.04.1 LTS Kernel version 5.15.90.1-microsoft-standard-WSL2
Docker*	Version 20.10.21
Docker* Compose	Version 2.15.1
OpenVINO™ Toolkit	Version 2022.3.0
Xibo Signage*	Version 3.3.3

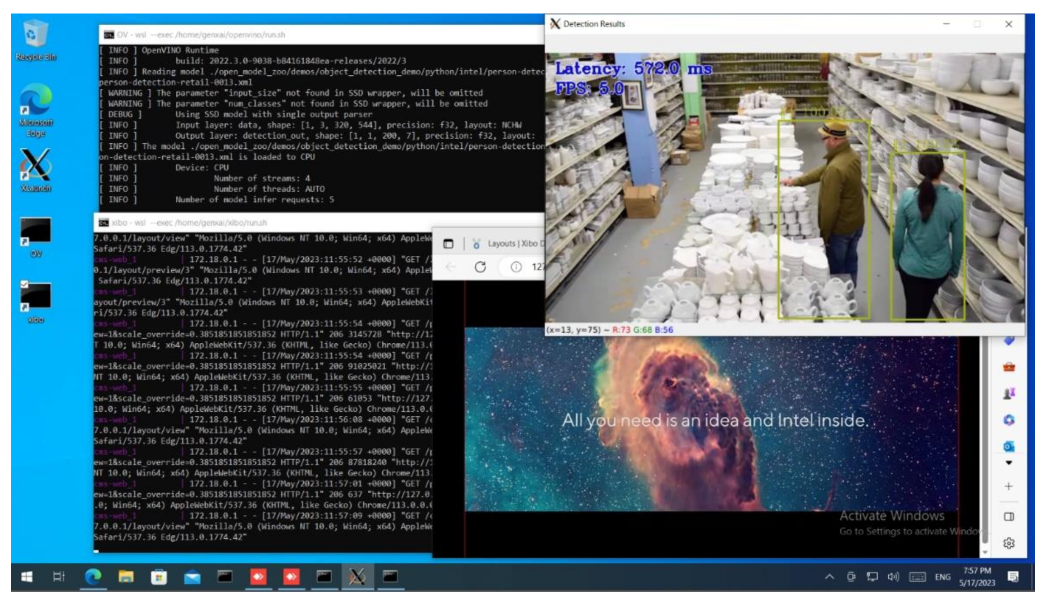
5.2 Results

After installing various dependencies, software system was successfully booted, and the following applications was executed:

1. Point of Sale application runs directly on Windows* OS.
2. OpenVINO™ AI analytics application runs in a container on Linux* with GUI displaying analytics result of person detection.
3. A playback of 1080p HDR digital signage on web browser based on Xibo Signage* server application running in the other Linux* container.

Figure 4 shows all applications running concurrently.

Figure 4. Linux* GUI and Server Containerized Applications on WSL



OV - wsl --exec /home/genxai/openvino/run.sh

```
[ INFO ] OpenVINO Runtime
[ INFO ]        build: 2022.3.0-9038-b84161848ea-releases/2022/3
[ INFO ] Reading model ./open_model_zoo/demos/object_detection_demo/python/intel/person-detec
person-detection-retail-0013.xml
[ WARNING ] The parameter "input_size" not found in SSD wrapper, will be omitted
[ WARNING ] The parameter "num_classes" not found in SSD wrapper, will be omitted
[ DEBUG ]       Using SSD model with single output parser
[ INFO ]        Input layer: data, shape: [1, 3, 320, 544], precision: f32, layout: NCHW
[ INFO ]        Output layer: detection_out, shape: [1, 1, 200, 7], precision: f32, layout:
[ INFO ] The model ./open_model_zoo/demos/object_detection_demo/python/intel/person-detection
on-detection-retail-0013.xml is loaded to CPU
[ INFO ]        Device: CPU
[ INFO ]                Number of streams: 4
[ INFO ]                Number of threads: AUTO
[ INFO ]        Number of model infer requests: 5
```

xibo - wsl --exec /home/genxai/xibo/run.sh

```
7.0.0.1/layout/view" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
Safari/537.36 Edg/113.0.1774.42"
cms-web_1          | 172.18.0.1 - - [17/May/2023:11:55:52 +0000] "GET /1
0.1/layout/preview/3" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleW
 Safari/537.36 Edg/113.0.1774.42"
cms-web_1          | 172.18.0.1 - - [17/May/2023:11:55:53 +0000] "GET /1
ayout/preview/3" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit
ri/537.36 Edg/113.0.1774.42"
cms-web_1          | 172.18.0.1 - - [17/May/2023:11:55:54 +0000] "GET /p
ew=1&scale_override=0.3851851851851852 HTTP/1.1" 206 3145728 "http://12
T 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0
cms-web_1          | 172.18.0.1 - - [17/May/2023:11:55:54 +0000] "GET /p
ew=1&scale_override=0.3851851851851852 HTTP/1.1" 206 91025021 "http://1
NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.
cms-web_1          | 172.18.0.1 - - [17/May/2023:11:55:55 +0000] "GET /p
ew=1&scale_override=0.3851851851851852 HTTP/1.1" 206 61053 "http://127.
10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0
cms-web_1          | 172.18.0.1 - - [17/May/2023:11:56:08 +0000] "GET /c
7.0.0.1/layout/view" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
Safari/537.36 Edg/113.0.1774.42"
cms-web_1          | 172.18.0.1 - - [17/May/2023:11:55:57 +0000] "GET /p
ew=1&scale_override=0.3851851851851852 HTTP/1.1" 206 87818240 "http://1
NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.
cms-web_1          | 172.18.0.1 - - [17/May/2023:11:57:01 +0000] "GET /p
ew=1&scale_override=0.3851851851851852 HTTP/1.1" 206 637 "http://127.0.
.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/113.0.0.0
cms-web_1          | 172.18.0.1 - - [17/May/2023:11:57:09 +0000] "GET /c
7.0.0.1/layout/view" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
Safari/537.36 Edg/113.0.1774.42"
```

NOTE: Top row shows the GUI application whereas the bottom row shows the server application.

5.3 Summary

Running containerized applications is useful to realize workload consolidation at the edge. Using containers with Docker* or similar container daemon can help in assigning resources based on priority and isolating workloads for better security. Additional layer or security can be enabled using privileged access management of the containers.

WSL is one of the mechanisms to run Linux* on Windows* environment, but it is recommended to run containers on additional OS such as Ubuntu*. WSL is targeted for development environment, but for productization, it is recommended to use other solutions such Azure* IoT Edge for Linux on Windows* (EFLOW). There could be different dependencies, prerequisites, and procedures to realize the similar system as described in this paper using EFLOW. EFLOW uses CBL-Mariner Linux* instead of Ubuntu*, however the fundamental architecture and concept will remain same.

§