Scaling Edge Inference Deployments on Enterprise IoT Implementations

APRIL | 2020

Chandler Heath
Technical Solutions Architect

Mateo Guzman
IoT Solutions Architect

Nicolas Oliver
IoT Solutions Architect

Marcos Carranza, Intel Corporation
Senior IoT Solutions Architect

Lakshmi Talluru, Intel Corporation
Sr. Director Digital Transformation

Cesar Martinez Spessot, Intel Corporation
Engineering Director, Senior Analytics & AI Architect

World Wide Technology
wwt.com

Table of Contents

Executive Summary

Intel and WWT have teamed up to build an AI and Data Analytics optimized infrastructure to support specific usage models and workflows required by Top 100 enterprises as part of their business demands.

One of the significant challenges that developers face in building scalable IoT solutions for the enterprise is the optimization of their software based on the broad and fragmented IoT components. There are a tremendous implementation effort and time consumption from the development to the production stage. Also, security is usually left as one of the last priorities but is critical in deploying a solution for enterprise customers.

This paper presents the overall Edge Inference Infrastructure based on OpenVINO and Intel's processors and accelerators. The outcome of this joint work is a full end-to-end IoT reference architecture with ingredients, capabilities, and features from hardware/device, Operative System (OS), and containerization that meets customer requirements.

Problem Statement: Creating Production Ready Edge Inference Applications

In a traditional software development process, there are 2 clearly identifiable environments: the development environment where the software product is created, and the production environment where the software product is executed to provide a valuable functionality to a given user or system. These two environments are quite different between each other, and different tools and configurations are used on each of them to achieve value. On development, the focus is on making the software developers productive, giving them different tools like debuggers, compilers, testing frameworks, and validation tools that help them create the software product. On the other side of the spectrum, the production environments are commonly managed by IT administrators and DevOps engineers, with the focus on optimizing the resource utilization of the infrastructure, and protecting the different assets participating in the system from security vulnerabilities and the guarding the entire availability, confidentiality and integrity of the system.

The current trend in software development process is the adoption of tools that will try to normalize how the software is developed and executed, being containers the most relevant technology in this area. Containers allowed developers to specify the production environment with an Infrastructure-as-code technique leveraged by the Dockerfiles and provided the ability to easily execute the software product in the production environment by reducing the hassle of having to reconfigure the infrastructure to meet the software product expectations.

But containers are not a silver bullet that solves the development vs production environment differences. The development process needs to clearly provide the correct baseline and guidelines to effectively normalizing the environments without incurring in higher resource consumption or relaxation of security mechanisms.

Based on this, many applications coming from development teams as production images are really big, and sometimes insecure since many software components or tools were left from development stages. This article describes specifically applications using the Intel Distribution of OpenVINO, which has provided a solution to run computer vision workloads without the need of having the knowledge base of a computer scientist, enabling a much broader software development audience to solve problems can be resolved with computer vision. As previously mentioned, image size and security must be considered otherwise a simple use case commonly incurs in images with Gigabytes of size, and usually required running them with privileged mode to make use of the operative system resources, causing an unnecessary waste of resources and increasing the attack surface for security vulnerabilities. Additionally, by keeping development components as part of a production image also increase the attack surface.

Creating Secure and Efficient Containerized OpenVINO* Based Applications

BASE IMAGE GENERATION

By using publicly available tools and methodologies it is possible to minimize the stated problem, with minimal impact on the software development process.

USING DOCKER MULTI-STAGE BUILDS

Docker multi-stage builds provide a useful path for images and containers optimization. One of the most challenging requirements about building images is keeping the image size down. Each instruction in the Dockerfile adds a layer to the image, and the developer need to remember to clean up any artifacts that is not required before moving on to the next layer. To write an efficient Dockerfile, it is traditionally needed to employ shell tricks and other logic to keep the layers as small as possible and to ensure that each layer has the artifacts it needs from the previous layer and nothing else.

It was actually very common to have a development Dockerfile (which contained everything needed to build your application), and a slimmed-down production Dockerfile, which only contained your application and exactly what was needed to run it. This has been referred to as the “builder pattern”. But, maintaining two Dockerfiles is not ideal.

With multi-stage builds, you use multiple FROM statements in your Dockerfile. Each FROM instruction can use a different base, and each of them begins a new stage of the build. You can selectively copy artifacts from one stage to another, leaving behind everything you don’t want in the final image.

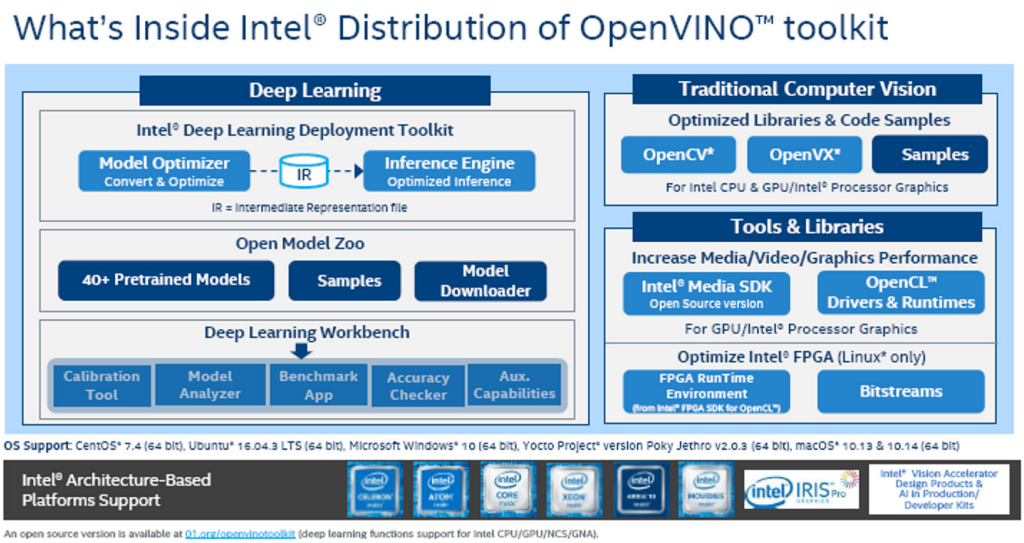
The result is the same optimized production image as before, with a significant reduction in complexity. There is no need to create any intermediate images and extract any artifacts to the local system at all.

```
FROM ubuntu:18.04 AS openvino-dev,
ARG DOWNLOAD_LINK=http://registrationcenter-download.intel.com/akdlm/irc_nas/15944/l_openvino_
toolkit_p_2019.3.334.tgz
ARG INSTALL_DIR=/opt/intel/openvino
ARG TEMP_DIR=/tmp/openvino_installer
# Install dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
build-essential \
cpio \
curl \
```

```
git \
lsb-release \
pciutils \
python3.6 \
python3.6-dev \
python3-pip \
python3-setuptools \
sudo \
wget && \
rm -rf /var/lib/apt/lists/* && \
pip3 install numpy
# Download and install OpenVINO
RUN mkdir -p $TEMP_DIR && cd $TEMP_DIR && \
wget -c $DOWNLOAD_LINK && \
tar xf l_openvino_toolkit_p_2019.3.334.tgz && \
cd l_openvino_toolkit_p_2019.3.334 && \
./install_openvino_dependencies.sh && \
sed -i 's/decline/accept/g' silent.cfg && \
./install.sh --silent silent.cfg && \
rm -rf $TEMP_DIR

# Model Optimizer prerequisites
RUN cd $INSTALL_DIR/deployment_tools/model_optimizer/install_prerequisites && \
./install_prerequisites.sh
# Create minimal deployment package (no OpenCV)
RUN /bin/bash -c "source $INSTALL_DIR/bin/setupvars.sh" && \
$INSTALL_DIR/deployment_tools/tools/deployment_manager/deployment_manager.py \
--targets cpu --output_dir /tmp --archive_name openvino_deploy_package

## Minimal base runtime image
FROM ubuntu:18.04 AS openvino-runtime

COPY --from=openvino-dev /tmp/openvino_deploy_package.tar.gz /tmp
RUN mkdir -p /opt/intel && tar xzf /tmp/openvino_deploy_package.tar.gz -C /opt/intel
RUN echo "source /opt/intel/openvino/bin/setupvars.sh" >> /root/.bashrc
```

OpenVINO Components for Development

OpenVINO is a toolkit to accelerate development of high-performance computer vision & deep learning inference into vision/AI application used from edge to cloud. It enables deep learning on hardware accelerators and easy deployment across multiple types of Intel platforms.



Every tool provided by the OpenVINO frameworks serves a specific purpose to enable software developers to create AI workloads and have an improved developer experience. But in order to run these workloads, only the required libraries and binaries should be included in the production environment.

OpenVINO Components for Runtime

Creating an optimized production workload can be achieved with the help of the OpenVINO Deployment Manager Tool*, which allows to create a software archive including only the required components to execute the workloads. The output of the deployment tool, in combination with the Docker multi-stage build process, can be used to create an optimized and more secure base image for OpenVINO based workloads. Optimized in the sense that it will have the minimal storage and transport overhead in the production environment. Secure in the sense that provides a reduced attack surface for any security exploit that abuses vulnerabilities in the software contained in the production environment.

An example on how to utilize Deployment Manager Tool can be found in Base Image generation section - Item 1. More info on Deployment Manager Tool available at link.

Considering Workload Consolidation Scenarios and Accelerators

Workload Consolidation concept refers to running several co-existing applications in the same device, and if the device includes accelerators such as FPGA, or VPUs, it is important to consider this in order to generate the most efficient application image.

Including FPGA support or VPU support may increase the size of the image several 10's MBs, but this additional cost allows the utilization of the same container in runtime. With this approach there will be 1 base image that can be used with CPU or accelerators. In case that the additional size is important (network or storage related) then a different approach can be considered, and specific images can be created for each variant:

- Image for CPU
- Image for FPGA
- Image for VPU
- Image for other accelerators

This may become in a problem for QA and DevOps. It is recommended to perform a tradeoff analysis evaluating PROs and CONs, which depends on customer requirements.

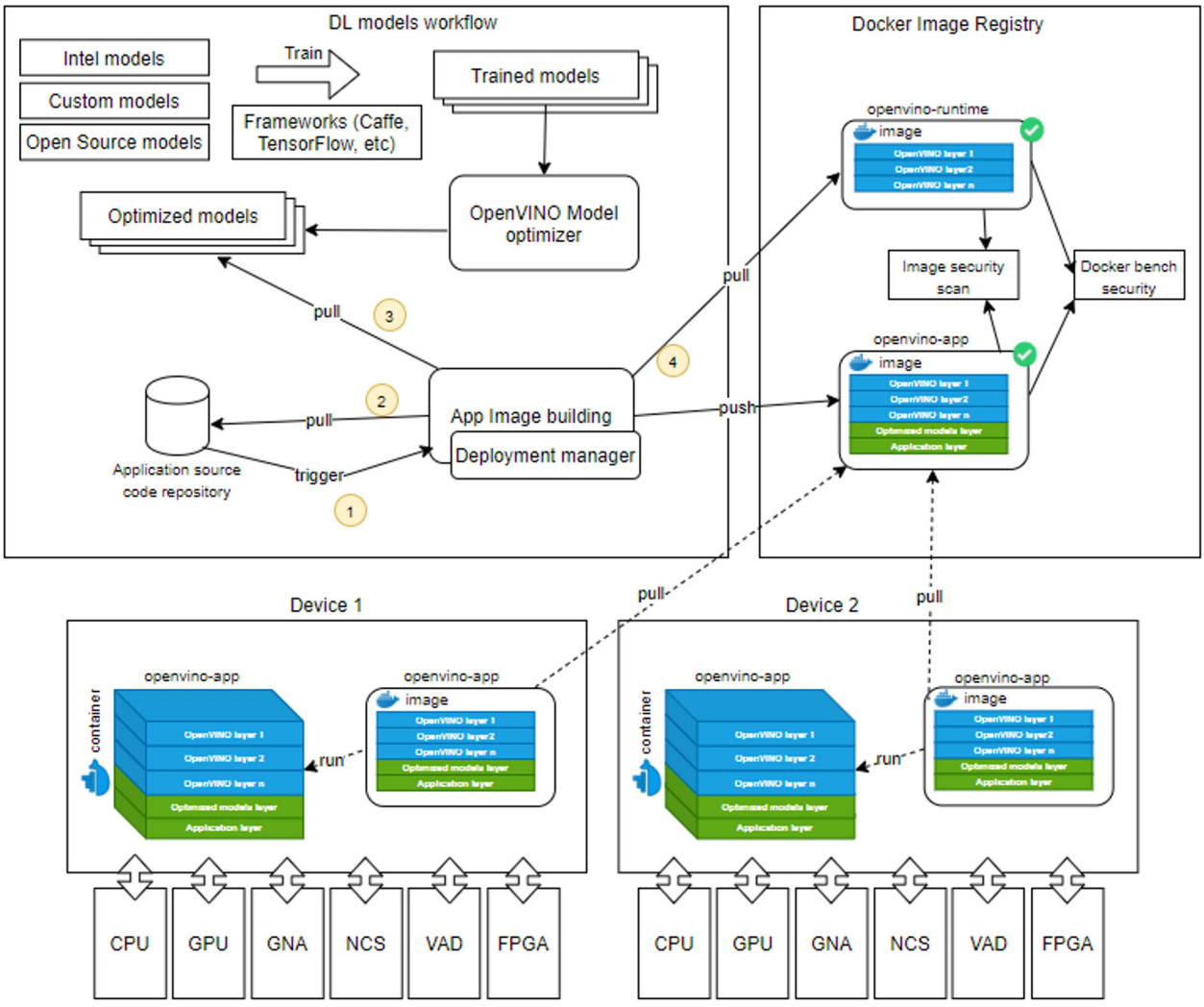
Example on how to utilize Deployment Manager Tool generating an image with support for CPU + VPU:

```
$INSTALL_DIR/deployment_tools/tools/deployment_manager/deployment_manager.py \
--targets cpu vpu --output_dir /tmp --archive_name openvino_deploy_package
```

Application Image Generation

This optimized Docker image with the Intel OpenVINO runtime can now be used as a base image for all the applications that uses the OpenVINO Inference Engine.

The following diagram shows how this optimized image can be reused between different applications.



A computer vision application may use one or more Deep Learning models, which are typically trained and generated using a deep learning framework (this process is not covered in this document). These models are transformed to an Intermediate Representation using the Intel® OpenVINO Model Optimizer in order to be able to use them with the OpenVINO Inference Engine.

Image Publishing Process

Once the image is ready for being used, it has to be uploaded to the available enterprise pre-production image registry. This will allow to perform a quality and security analysis mentioned in the next section.

Once the image is confirmed as secure, it can be promoted to the available enterprise production image registry, which will hold all the different images available to use in production environment.

Image Security and Trust

As mentioned in previous sections, security is a key area to consider during each phase of the software development lifecycle. Once the images are generated, they have to be stored in a secure location and also there are tools to help improve their quality from the security perspective, such as image scanners which will consider up to date CVE database.

This is a must have in the current challenging world in which new vulnerabilities are identified every day.

In the Appendix section there is a mention to options to perform this using open source or commercial technologies.

BENEFITS

The benefits of using the strategies and recommendation mentioned in this whitepaper are:

- Storage utilization: by having smaller images, it is simpler to deploy new applications, or to update already deployed applications. This will reduce storage of the device holding the image and will also reduce the use of networking.
  Also related with storage, by having a container cache infrastructure, that is by using docker cache, it is possible to reduce the size of images, not replicating commonalities for applications using the same base components.
- Security:
  - By having images including only required functionality, the attack surface is reduced considerably, leaving in the image only the components that are known, and not just a black box.

- Image signature allows to check the authenticity/integrity of the images, avoiding cases in which attackers can modify the functionality.
- Image scan verifies that they are free from known security vulnerabilities (CVE database) or exposures.
- By reusing the same base image (OpenVINO image) the attack surface is reduced.

REQUIREMENTS

- Intel Distribution of OpenVINO (Latest)
- Docker > 17.05 (support of multi-stage builds)
- Harbor (Latest),
- Clair (Latest),
- Notary (Latest).

Conclusion

This whitepaper provides a clear recipe and BKMs on how to generate and manage optimized applications dependent on the Intel Distribution of OpenVINO to be used on Edge Production environments, with the main objective of keeping systems secure and having optimized resource utilization.

Appendix

Infrastructure required for increasing security of image generation and promotion.

In order to securely store docker images, assuring integrity and authenticity, a trusted private image registry is needed. One example of a private registry is the Harbor open source project. An example of a commercial registry is explained in the “Secure the IoT Edge with Trusted Docker Containers” whitepaper.

Harbor is an open source trusted cloud native registry project that stores, signs, and scans content. Harbor extends the open source Docker distribution by adding the functionalities usually required by users such as security, identity and management. Having a local centralized registry, closer to the build and run environment, can improve the image transfer efficiency vs. using remote registries such as hub.docker.com. Harbor supports replication of images between registries, and offers advanced security features such as user management, access control and activity auditing.

Harbor can also be integrated with a Notary server to allow signing and verification of images. Notary is a tool for publishing and managing trusted collections of content. Publishers can digitally sign collections and consumers can verify integrity and origin of content. This ability is built on a straightforward key management and signing interface to create signed collections and configure trusted publishers.

With Notary anyone can provide trust over arbitrary collections of data. Using The Update Framework (TUF) as the underlying security framework, Notary takes care of the operations necessary to create, manage, and distribute the metadata necessary to ensure the integrity and freshness of your content.

Additionally, images can be scanned to identify vulnerabilities. An example of this is Harbor scanning images on a regular basis and warning users of vulnerabilities by using the Clair scanner. Clair is an open source project for the static analysis of vulnerabilities in application containers like Docker. In regular intervals, Clair ingests vulnerability metadata from common vulnerabilities databases like the National Vulnerability Database (NVD) from NIST and MITRE CVE database and stores it in its local database. It scans each layer and aggregates the results to give a complete picture of what is being shipped as a part of the software stack. Most importantly, it correlates this information with a vulnerability database that is kept up to date through periodic updates. This gives unprecedented insight into your exposure to known security threats. Security Scanning is done for each pushed tag and can be configured to happen automatically after every push. An example of a scanned image tagged and pushed to a Harbor registry using the Clair scanner is presented in the image. The scanner detects each of the components present in the different layers that composes the image, and report vulnerabilities based on the version of the detected component.