EWAOL

Arm Ltd.

Oct 11, 2023

CONTENTS

CHAPTER

ONE

CONTENTS

1.1 Introduction

The Edge Workload Abstraction and Orchestration Layer (EWAOL) project provides users with a standards-based framework using containers for the deployment and orchestration of applications on edge platforms. EWAOL is provided as the `meta-ewaol` repository, which includes metadata for building EWAOL distribution images via the Yocto Project.

Under this approach, a full software stack is divided into the following software layers:

- **Application workloads**:

  User-defined container applications that are deployed and executed on the EWAOL software stack. Note that the EWAOL project provides the system infrastructure for user workloads, and not the application workloads themselves. Instead, they should be deployed by end-users according to their individual use-cases.

- **Linux-based filesystem**:

  This is the main component provided by the EWAOL project. The EWAOL filesystem contains tools and services that provide EWAOL core functionalities and facilitate deployment and orchestration of user application workloads. These tools and services include the Docker container engine, the K3s container orchestration framework, and Xen virtualization management software, together with their run-time dependencies. In addition, EWAOL provides supporting packages such as those which enable run-time validation tests or software development capabilities on the target platform.

- **System software**:

  System software specific to the target platform, composed of firmware, bootloader and the operating system, as well as the Xen type-1 hypervisor when building an EWAOL distribution with hardware virtualization support. Note that this system software is not directly developed as part of the EWAOL project, but is instead integrated from the meta-arm, meta-arm-bsp, and meta-virtualization Yocto layers.

EWAOL is the reference implementation for SOAFEE (Scalable Open Architecture For Embedded Edge), the Arm lead industry initiative for extending cloud-native software development to automotive, with a special focus on real-time and functional safety. For more details, please see https://soafee.io.

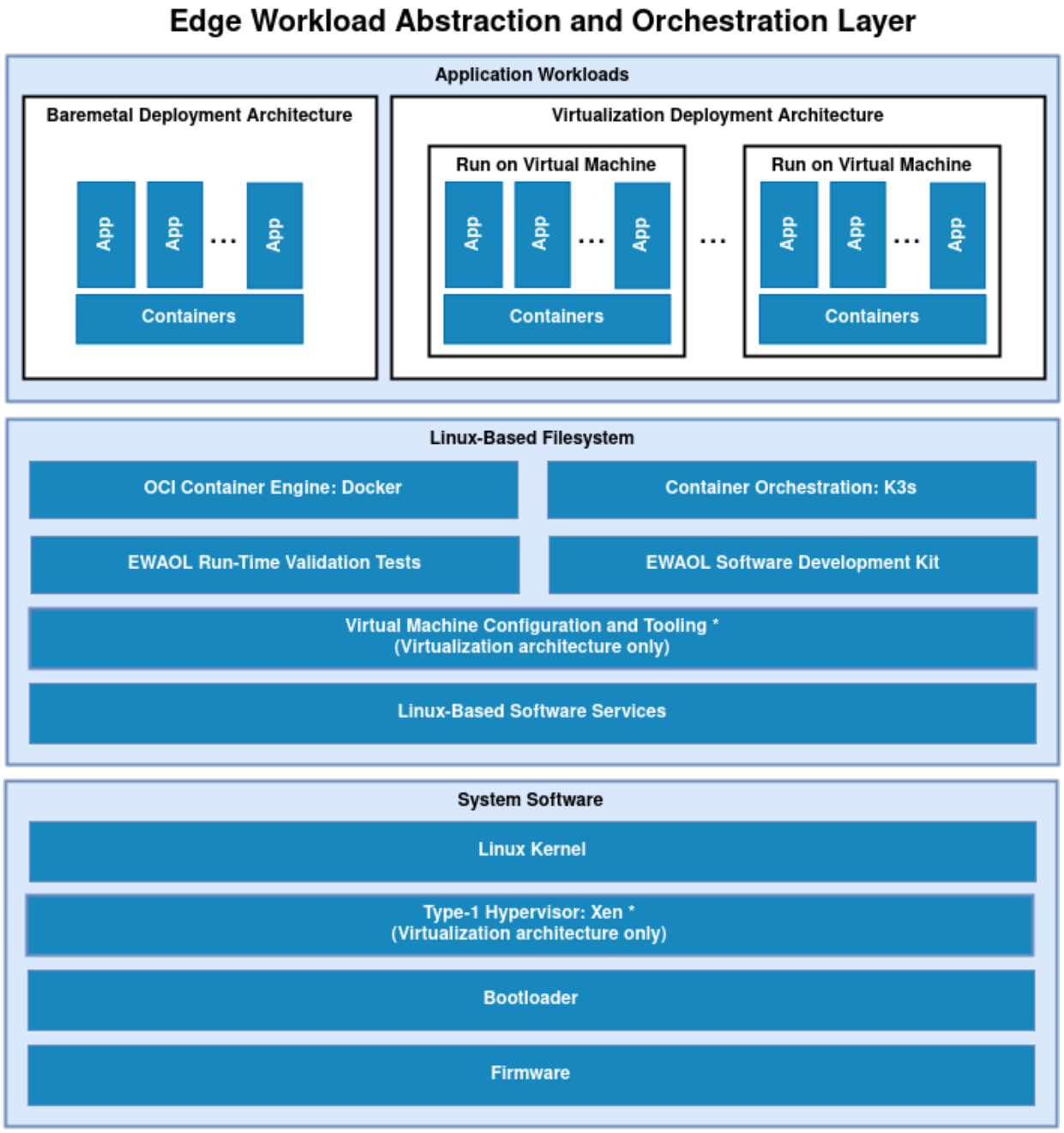
More specifically, the `meta-ewaol` repository contains Yocto layers, configuration files, documentation and tools to support building EWAOL distribution images and validating core functionalities. The project currently supports two hardware target platforms:

- Arm's Neoverse N1 System Development Platform (N1SDP)
- ADLINK's AVA Developer Platform (AVA)

Note: Users of this software stack must consider safety and security implications according to their own usage goals.

1.1.1 High-Level Overview

The following diagram illustrates the EWAOL software stack. EWAOL provides two different system architectures to support application workload deployment and orchestration, as described in *EWAOL System Architectures* later in this introduction.



1.1.2 Use-Cases Overview

EWAOL aims to facilitate the following core use-cases on the supported target platforms:

- Deployment of application workloads via Docker and K3s.
- Orchestration of resource-managed and isolated application workloads via Docker, K3s, and the Xen type-1 hypervisor providing hardware virtualization.

Instructions for achieving these use-cases are given in the *Reproduce* section of the User Guide, subject to relevant assumed technical knowledge as listed later in this introduction at *Documentation Assumptions*.

1.1.3 EWAOL System Architectures

There are two primary system architectures currently provided by EWAOL, differing according to the intended use-case, as follows:

Baremetal Architecture

For this architecture, the EWAOL software stack executes directly on the target hardware. This architecture supports the deployment and orchestration of application workloads running on a target platform without hardware virtualization.

Virtualization Architecture

For this architecture, the EWAOL software stack also includes the Xen type-1 hypervisor to support hardware virtualization in the form of isolated, resource-managed Virtual Machines (VMs). An EWAOL virtualization distribution image will include a Control VM (Dom0) and a single bundled Guest VM (DomU), by default. This architecture enables the deployment and orchestration of application workloads on a set of distinct VMs running on a single target platform.

EWAOL defines two customizable image build targets per target architecture: a standard EWAOL distribution image to support deployment and orchestration of an existing application workload, and an EWAOL distribution image which includes a Software Development Kit (SDK) that supports on-target development and analyses of application workloads and system services.

1.1.4 Features Overview

EWAOL includes the following major features:

- Support for two architectural use-cases (Baremetal, and Virtualization).
- Container engine and runtime with Docker and runc-opencontainers.
- Container workload orchestration with the K3s Kubernetes distribution.
- Hardware virtualization support with the Xen type-1 hypervisor.
- On-target development support with optionally included Software Development Kit.
- Validation support with optionally included run-time integration tests, and build-time kernel configuration checks.
- Tools provided for quality assurance and build support.

Other features of EWAOL include:

- The features provided by the `poky.conf` distribution, which EWAOL extends.
- Systemd used as the init system.
- RPM used as the package management system.

1.1.5 Documentation Overview

The documentation is structured as follows:

- *User Guide*

  Provides guidance for configuring, building, and deploying EWAOL distributions on supported target platforms, running and validating supported EWAOL functionalities, and building the distribution for a custom or unsupported target platform. Also includes migration guidance for how to enable these activities on a later EWAOL release, when upgrading from an older release.

- *Developer Manual*

  Provides more advanced developer-focused details of the EWAOL distribution, its implementation, and dependencies.

- *Codeline Management*

  Describes the branch and release process of EWAOL, and how this process aligns with that of the Yocto Project.

- *Contributing*

  Describes guidance for contributing to the EWAOL project, and describes the tooling provided to support it.

- *License*

  Defines the license under which EWAOL is provided.

- *Changelog & Release Notes*

  Documents new features, bug fixes, limitations, and any other changes provided under each EWAOL release.

Documentation Assumptions

This documentation assumes a base level of knowledge related to two different aspects of achieving the target use-cases via EWAOL:

- Application workload containerization, deployment, and orchestration

  This documentation does not provide detailed guidance on developing application workloads, deploying them, or managing their execution via Docker or the K3s orchestration framework, and instead focuses on EWAOL-specific instructions to support these activities on an EWAOL distribution image.

  For information on how to use these technologies which are provided with the EWAOL distribution, see the Docker documentation and the K3s documentation.

- Xen Type-1 Hypervisor

  EWAOL supports deployment and orchestration of application workloads running on isolated and resource-managed VMs enabled by the Xen type-1 hypervisor. However, this documentation does not provide detailed guidance for booting Xen hardware virtualized systems or managing VMs on an EWAOL distribution image, and provides only basic instructions for logging into a VM as part of the example instruction sequences within the User Guide.

  For detailed guidance on booting Xen hardware virtualized systems as well as managing and connecting to Xen VMs using Xen-specific tools and services, see the public Xen documentation.

- The Yocto Project

  This documentation contains instructions for achieving EWAOL's use-cases using a set of included configuration files that provide standard build features and settings. However, EWAOL forms a distribution layer for integration with the Yocto project and is thus highly configurable and extensible. This documentation supports those activities by detailing the available options for EWAOL-specific customizations and extensions, but assumes knowledge of the Yocto project necessary to prepare an appropriate build environment with these options configured.

  Readers are referred to the Yocto Project Documentation for information on setting up and running non-standard EWAOL distribution builds.

1.1.6 Repository Structure

The `meta-ewaol` repository (https://gitlab.com/soafee/ewaol/meta-ewaol) is structured as follows:

- `meta-ewaol`:
  - `meta-ewaol-distro`

    Yocto distribution layer providing top-level and general policies for the EWAOL distribution images.

  - `meta-ewaol-tests`

    Yocto software layer with recipes that include run-time tests to validate EWAOL functionalities.

  - `meta-ewaol-bsp`

    Yocto BSP layer with target platform specific extensions for particular EWAOL distribution images.

  - `meta-ewaol-config`

    Directory which contains configuration files for running tools on EWAOL, such as files to support use of the kas build tool, or EWAOL-specific configuration for running automated quality-assurance checks.

  - `documentation`

    Directory which contains the documentation sources, defined in ReStructuredText (`.rst`) format for rendering via `sphinx`. See the *Documentation Build Validation* page for guidance on building the documentation.

  - `tools`

    Directory that contains supporting tools for the EWAOL project, from tools to support Bitbake image builds or documentation builds (provided within `tools/build`) to tools for quality assurance (provided within `tools/qa-checks`).

The Yocto layers which are provided by `meta-ewaol` are detailed with their layer dependencies in *Yocto Layers*.

1.1.7 Repository License

The repository's standard license is the MIT license (more details in *License*), under which most of the repository's content is provided. Exceptions to this standard license relate to files that represent modifications to externally licensed works (for example, patch files). These files may therefore be included in the repository under alternative licenses in order to be compliant with the licensing requirements of the associated external works.

Contributions to the project should follow the same licensing arrangement.

1.1.8 Contributions and Issue Reporting

Guidance for contributing to the EWAOL project can be found at *Contributing*.

To report issues with the repository such as potential bugs, security concerns, or feature requests, please submit an Issue via GitLab Issues, following the project's template.

1.1.9 Maintainer(s)

- Javier Tia <javier.tia@linaro.org>

1.2 User Guide

1.2.1 Reproduce

This section of the User Guide describes how to reproduce a standard EWAOL distribution image for a supported *target platform*, configuring and deploying the supported set of distribution image features, and running simple examples of the *EWAOL Use-Cases*.

Introduction

The recommended approach for image build setup and customization is to use the kas build tool. To support this, EWAOL provides configuration files to setup and build different target images, different distribution image features, and set associated parameter configurations.

This page first briefly describes below the kas configuration files provided with EWAOL, before guidance is given on using those kas configuration files to set up the EWAOL distribution on a target platform.

Note: All command examples on this page can be copied by clicking the copy button. Any console prompts at the start of each line, comments, or empty lines will be automatically excluded from the copied text.

The `meta-ewaol-config/kas` directory contains kas configuration files to support building and customizing EWAOL distribution images via kas. These kas configuration files contain default parameter settings for an EWAOL distribution build, and are described in more detail in *Build System*. Here, the files are briefly introduced, classified into three ordered categories:

- **Architecture Configs**: Set the target EWAOL architecture
  - `baremetal.yml` to prepare an image for the baremetal architecture.
  - `virtualization.yml` to prepare an image for the virtualization architecture.
- **Build Modifier Configs**: Set and configure features of the EWAOL distribution
  - `tests.yml` to include run-time validation tests on the image.
  - `baremetal-sdk.yml` to build an SDK image for the baremetal architecture.
  - `virtualization-sdk.yml` to build an SDK image for the virtualization architecture.
  - `security.yml` to build a security-hardened EWAOL distribution image.

  - `xen_pci_passthrough.yml` to include the necessary configuration to enable support for Xen Guest VM PCI passthrough. The configuration provided in this Build Modifier Config will only take effect when building an EWAOL virtualization distribution image, and is currently only supported on the AVA Developer Platform.

- **Target Platform Configs**: Set the target platform

  EWAOL currently supports two hardware target platforms, each with its own Target Platform Config:

  - `n1sdp.yml` to select the Neoverse N1 System Development Platform (N1SDP) as the target platform, corresponding to the `n1sdp` `MACHINE` implemented in meta-arm-bsp.

    See N1SDP Technical Reference Manual for more details about the N1SDP.

  - `ava.yml` to select the AVA Developer Platform (AVA) as the target platform, corresponding to the `ava` `MACHINE` implemented in meta-adlink-ampere.

    See AVA Developer Platform for more details about AVA.

Note: Additional information on EWAOL features such as run-time validation tests and the SDK can be found in the *Developer Manual*

These kas configuration files can be used to build a custom EWAOL distribution by passing one **Architecture Config**, zero or more **Build Modifier Configs**, and one **Target Platform Config** to the kas build tool, chained via a colon (:) character. Examples for this are given later in this document.

In the next section, guidance is provided for configuring, building and deploying EWAOL distributions using these kas configuration files.

Build Host Environment Setup

This documentation assumes an Ubuntu-based Build Host, where the build steps have been validated on the Ubuntu 18.04.6 LTS Linux distribution.

A number of package dependencies must be installed on the Build Host to run build scenarios via the Yocto Project. The Yocto Project documentation provides the list of essential packages together with a command for their installation.

The recommended approach for building EWAOL is to use the kas build tool. To install kas:

```
sudo -H pip3 install --upgrade kas==3.1
```

For more details on kas installation, see kas Dependencies & installation.

To deploy an EWAOL distribution image onto the supported target platform, this User Guide uses `bmap-tools`. This can be installed via:

```
sudo apt install bmap-tools
```

Note: The Build Host should have at least 65 GBytes of free disk space to build an EWAOL baremetal distribution image, or at least 100 GBytes of free disk space to build an EWAOL virtualization distribution image.

Download

The `meta-ewaol` repository can be downloaded using Git, via:

```
# Change the tag or branch to be fetched by replacing the value supplied to
# the --branch parameter option

mkdir -p ~/ewaol
cd ~/ewaol
git clone https://gitlab.com/soafee/ewaol/meta-ewaol.git --branch kirkstone-dev
cd meta-ewaol
```

Build

The provided kas configuration files can be combined to build and EWAOL distribution image for different target platforms, for different EWAOL system architectures, and to apply different sets of customizable parameters. Therefore, the following build guidance is provided as a set of alternatives to target each of the main supported use cases.

Alternatives are distinguished first by EWAOL system architecture as distinct sections, then by hardware target platform or distribution image feature, with each alternative denoted alphabetically (e.g., A, B, ...).

Baremetal Distribution

To build a baremetal distribution image:

A. For the N1SDP hardware target platform:

```
kas build --update meta-ewaol-config/kas/baremetal.yml:meta-ewaol-
↪config/kas/n1sdp.yml
```

The resulting baremetal distribution image will be produced at: `build/tmp_baremetal/deploy/images/n1sdp/ewaol-baremetal-image-n1sdp.*`

B. For the AVA hardware target platform:

```
kas build --update meta-ewaol-config/kas/baremetal.yml:meta-ewaol-
↪config/kas/ava.yml
```

The resulting baremetal distribution image will be produced at: `build/tmp_baremetal/deploy/images/ava/ewaol-baremetal-image-ava.*`

To build a baremetal distribution image with the EWAOL SDK:

C. For the N1SDP hardware target platform:

```
kas build --update meta-ewaol-config/kas/baremetal-sdk.yml:meta-ewaol-
↪config/kas/n1sdp.yml
```

The resulting baremetal distribution image which includes the EWAOL SDK will be produced at: `build/tmp_baremetal/deploy/images/n1sdp/ewaol-baremetal-sdk-image-n1sdp.*`

D. For the AVA hardware target platform:

```
kas build --update meta-ewaol-config/kas/baremetal-sdk.yml:meta-ewaol-
↪config/kas/ava.yml
```

(continues on next page)

(continued from previous page)

```
```

The resulting baremetal distribution image which includes the EWAOL SDK will be produced at: `build/tmp_baremetal/deploy/images/ava/ewaol-baremetal-sdk-image-ava.*`

EWAOL baremetal distribution images can be modified by adding run-time validation tests and security hardening to the distribution. This can be done by including `meta-ewaol-config/kas/tests.yml` and `meta-ewaol-config/kas/security.yml` kas configuration file as a Build Modifier. See *Run-Time Integration Tests* for more details on including run-time validation tests and *Security Hardening* for more details on security hardening.

Virtualization Distribution

To build a virtualization distribution image:

A. For the N1SDP hardware target platform:

```
kas build --update meta-ewaol-config/kas/virtualization.yml:meta-ewaol-
↪config/kas/n1sdp.yml
```

The resulting virtualization distribution image will be produced: `build/tmp_virtualization/deploy/images/n1sdp/ewaol-virtualization-image-n1sdp.*`

B. For the AVA hardware target platform:

```
kas build --update meta-ewaol-config/kas/virtualization.yml:meta-ewaol-
↪config/kas/ava.yml
```

The resulting virtualization distribution image will be produced: `build/tmp_virtualization/deploy/images/ava/ewaol-virtualization-image-ava.*`

To build a virtualization distribution image with the EWAOL SDK:

C. For the N1SDP hardware target platform:

```
kas build --update meta-ewaol-config/kas/virtualization-sdk.yml:meta-
↪ewaol-config/kas/n1sdp.yml
```

The resulting virtualization distribution image which includes the EWAOL SDK will be produced at: `build/tmp_virtualization/deploy/images/n1sdp/ewaol-virtualization-sdk-image-n1sdp.*`

D. For the AVA hardware target platform:

```
kas build --update meta-ewaol-config/kas/virtualization-sdk.yml:meta-
↪ewaol-config/kas/ava.yml
```

The resulting virtualization distribution image which includes the EWAOL SDK will be produced at: `build/tmp_virtualization/deploy/images/ava/ewaol-virtualization-sdk-image-ava.*`

As with the EWAOL baremetal guidance above, EWAOL virtualization distribution images can also be modified to include run-time validation tests and security hardening by adding `meta-ewaol-config/kas/tests.yml` and `meta-ewaol-config/kas/security.yml` kas configuration files respectively. In addition, an EWAOL virtualization distribution image built for the AVA Developer Platform can be customized so that Guest VMs may be assigned an exclusive PCI device via Xen PCI passthrough capability, added via the `meta-ewaol-config/kas/xen_pci_passthrough.yml` kas configuration file. See *Run-Time Integration Tests* for more details on including

run-time validation tests, *Security Hardening* for more details on security hardening, and *Configuring Guest VM PCI Device Passthrough* for more details on PCI passthrough configuration.

Customization

EWAOL defines a set of standard customizable environment variables for configuring the VMs included on a virtualization distribution image. The following list shows the variables and their default values (where `MB` and `KB` refer to Megabytes and Kilobytes, respectively), when including one Guest VM instance:

```
EWAOL_GUEST_VM_INSTANCES: "1"                       # Number of Guest VM␣
↪instances
EWAOL_GUEST_VM1_NUMBER_OF_CPUS: "4"                 # Number of CPUs for Guest␣
↪VM1
EWAOL_GUEST_VM1_MEMORY_SIZE: "6144"                 # Memory size for Guest VM1␣
↪(MB)
EWAOL_GUEST_VM1_ROOTFS_EXTRA_SPACE: ""              # Extra storage space for␣
↪Guest VM1 (KB)
EWAOL_CONTROL_VM_MEMORY_SIZE: "2048"                # Memory size for Control␣
↪VM (MB)
EWAOL_CONTROL_VM_ROOTFS_EXTRA_SPACE: "0"            # Extra storage space for␣
↪Control VM (KB), added as additional space above the storage necessary to␣
↪support all Guest VMs root filesystems
```

To customize these standard variables, set their value in the environment for the kas build. For example, to build a virtualization distribution image for the N1SDP using the above default values, but allocating a non-default value of eight CPUs for its Guest VM, run:

```
EWAOL_GUEST_VM1_NUMBER_OF_CPUS=8 kas build --update meta-ewaol-config/kas/
↪virtualization.yml:meta-ewaol-config/kas/n1sdp.yml
```

An additional non-default environment variable is available for each Guest VM, which can be used to assign the Guest VM exclusive use of a single PCI device. Using this environment variable requires that the Xen PCI passthrough capability is enabled. Details for enabling this capability is provided at *Configuring Guest VM PCI Device Passthrough*. This will provide a corresponding environment variable for each Guest VM, such as the following variable and its default value for the first Guest VM:

```
EWAOL_GUEST_VM1_PCI_PASSTHROUGH_DEVICE: "0000:01:00.0"          # PCI device ID␣
↪to be assigned
```

EWAOL supports adding multiple independently-configurable Guest VMs to a virtualization distribution image. Additional details for this are provided at *Customization*.

Deploy

This section provides instructions for deploying an EWAOL distribution image on the support hardware target platforms:

- *EWAOL distribution image deployment on N1SDP*
- *EWAOL distribution image deployment on AVA*

Note: As the image filenames vary depending on the architecture and the inclusion of the SDK, the precise commands to deploy an EWAOL distribution image vary. The following documentation denotes required instructions with

sequentially numbered indexes (e.g., 1, 2, ...), and distinguishes alternative instructions by denoting the alternatives alphabetically (e.g., A, B, ...).

The deployment guidance requires a physical connection able to be established between the hardware target platform and a PC that can be used to interface with it. For simplicity, this PC is assumed to be the Build Host.

N1SDP

Instructions for deploying an EWAOL distribution image on the N1SDP hardware target platform are divided into two parts:

- *Load the Image onto a USB Storage Device*
- *Update the N1SDP MCC Configuration MicroSD Card*

Load the Image onto a USB Storage Device

EWAOL distribution images are produced as files with the `.wic.bmap` and `.wic.gz` extensions. They must first be loaded to a USB storage device, as follows:

1. Prepare a USB storage device (minimum size of 64 GB).

   Identify the USB storage device using `lsblk` command:

   ```
   lsblk
   ```

   This will output, for example:

   ```
   NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
   sdc      8:0    0    64G  0 disk
   ...
   ```

Warning: In this example, the USB storage device is the `/dev/sdc` device. As this may vary on different machines, care should be taken when copying and pasting the following commands.

2. Prepare for the image copy:

   A. Baremetal

   ```
   sudo umount /dev/sdc*
   cd build/tmp_baremetal/deploy/images/n1sdp/
   ```

   B. Virtualization

   ```
   sudo umount /dev/sdc*
   cd build/tmp_virtualization/deploy/images/n1sdp/
   ```

Warning: The next step will result in all prior partitions and data on the USB storage device being erased. Please backup before continuing.

3. Flash the image onto the USB storage device using `bmap-tools`:

A. Baremetal distribution image:

```
sudo bmaptool copy --bmap ewaol-baremetal-image-n1sdp.wic.bmap ewaol-
↪baremetal-image-n1sdp.wic.gz /dev/sdc
```

B. Baremetal-SDK distribution image:

```
sudo bmaptool copy --bmap ewaol-baremetal-sdk-image-n1sdp.wic.bmap␣
↪ewaol-baremetal-sdk-image-n1sdp.wic.gz /dev/sdc
```

C. Virtualization distribution image:

```
sudo bmaptool copy --bmap ewaol-virtualization-image-n1sdp.wic.bmap␣
↪ewaol-virtualization-image-n1sdp.wic.gz /dev/sdc
```

D. Virtualization-SDK distribution image:

```
sudo bmaptool copy --bmap ewaol-virtualization-sdk-image-n1sdp.wic.
↪bmap ewaol-virtualization-sdk-image-n1sdp.wic.gz /dev/sdc
```

The USB storage device can then be safely ejected from the Build Host, and plugged into one of the USB 3.0 ports on the N1SDP.

Update the N1SDP MCC Configuration MicroSD Card

Note: This process doesn't need to be performed every time the USB Storage Device gets updated. It is only necessary to update the MCC configuration microSD card when the EWAOL major version changes.

The instructions are as follows:

1. Connect a USB-B cable between the Build Host and the DBG USB port of the N1SDP back panel.
2. Find four TTY USB devices in the `/dev` directory of the Build Host, via:

   ```
   ls /dev/ttyUSB*
   ```

   This will output, for example:

   ```
   /dev/ttyUSB0
   /dev/ttyUSB1
   /dev/ttyUSB2
   /dev/ttyUSB3
   ```

   By default the four ports are connected to the following devices:

   - ttyUSB<n> Motherboard Configuration Controller (MCC)
   - ttyUSB<n+1> Application processor (AP)
   - ttyUSB<n+2> System Control Processor (SCP)
   - ttyUSB<n+3> Manageability Control Processor (MCP)

   In this guide the ports are:

   - ttyUSB0: MCC

- ttyUSB1: AP
- ttyUSB2: SCP
- ttyUSB3: MCP

The ports are configured with the following settings:

- 115200 Baud
- 8N1
- No hardware or software flow support

3. Connect to the N1SDP's MCC console. Any terminal applications such as `putty`, `screen` or `minicom` will work. The `screen` utility is used in the following command:

```
sudo screen /dev/ttyUSB0 115200
```

4. Power-on the N1SDP via the power supply switch on the N1SDP tower. The MCC window will be shown. Type the following command at the `Cmd>` prompt to see MCC firmware version and a list of commands:

```
?
```

This will output, for example:

```
Arm N1SDP MCC Firmware v1.0.1
Build Date: Sep  5 2019
Build Time: 14:18:16
+ command ------------------+ function ----------------------------------+
| CAP "fname" [/A]          | captures serial data to a file             |
|                           |  [/A option appends data to a file]        |
| FILL "fname" [nnnn]       | create a file filled with text             |
|                           |  [nnnn - number of lines, default=1000]    |
| TYPE "fname"              | displays the content of a text file        |
| REN "fname1" "fname2"     | renames a file 'fname1' to 'fname2'        |
| COPY "fin" ["fin2"] "fout"| copies a file 'fin' to 'fout' file         |
|                           |  ['fin2' option merges 'fin' and 'fin2']   |
| DEL "fname"               | deletes a file                             |
| DIR "[mask]"              | displays a list of files in the directory  |
| FORMAT [label]            | formats Flash Memory Card                  |
| USB_ON                    | Enable usb                                 |
| USB_OFF                   | Disable usb                                |
| SHUTDOWN                  | Shutdown PSU (leave micro running)         |
| REBOOT                    | Power cycle system and reboot              |
| RESET                     | Reset Board using CB_nRST                  |
| DEBUG                     | Enters debug menu                          |
| EEPROM                    | Enters eeprom menu                         |
| HELP  or  ?               | displays this help                         |
|                                                                        |
| THE FOLLOWING COMMANDS ARE ONLY AVAILABLE IN RUN MODE                  |
|                                                                        |
| CASE_FAN_SPEED "SPEED"    | Choose from SLOW, MEDIUM, FAST             |
| READ_AXI "fname"          | Read system memory to file 'fname'         |
|          "address"        | from address to end address                |
|          "end_address"    |                                            |
| WRITE_AXI "fname"         | Write file 'fname' to system memory        |
```

(continues on next page)

(continued from previous page)

```
|          "address"       | at address                                  |
+--------------------------+---------------------------------------------+
```

5. In the MCC window at the `Cmd>` prompt, enable USB via:

```
USB_ON
```

6. Mount the N1SDP's internal microSD card over the DBG USB connection to the Build Host and copy the required files.

   The microSD card is visible on the Build Host as a disk device after issuing the `USB_ON` command in the MCC console, as performed in the previous step. This can be found using the `lsblk` command:

```
lsblk
```

   This will output, for example:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sdb      8:0    0     2G  0 disk
└─sdb1   8:1    0     2G  0 part
```

   Warning: In this example, the `/dev/sdb1` partition is being mounted. As this may vary on different machines, care should be taken when copying and pasting the following commands.

   Mount the device and check its contents:

```
sudo umount /dev/sdb1
sudo mkdir -p /tmp/sdcard
sudo mount /dev/sdb1 /tmp/sdcard
ls /tmp/sdcard
```

   This should output, for example:

```
config.txt   ee0316a.txt   LICENSES   LOG.TXT   MB   SOFTWARE
```

7. Wipe the mounted microSD card, then extract the contents of `n1sdp-board-firmware_primary.tar.gz` onto it:

   A. Baremetal

```
sudo rm -rf /tmp/sdcard/*
sudo tar --no-same-owner -xf \
  build/tmp_baremetal/deploy/images/n1sdp/n1sdp-board-firmware_primary.
↪tar.gz -C \
  /tmp/sdcard/ && sync
sudo umount /tmp/sdcard
sudo rmdir /tmp/sdcard
```

   B. Virtualization

```
sudo rm -rf /tmp/sdcard/*
sudo tar --no-same-owner -xf \
```

(continues on next page)

(continued from previous page)

```
  build/tmp_virtualization/deploy/images/n1sdp/n1sdp-board-firmware_
↪primary.tar.gz -C \
  /tmp/sdcard/ && sync
sudo umount /tmp/sdcard
sudo rmdir /tmp/sdcard
```

Note: If the N1SDP board was manufactured after November 2019 (Serial Number greater than `36253xxx`), a different PMIC firmware image must be used to prevent potential damage to the board. More details can be found in Potential firmware damage notice. The `MB/HBI0316A/io_v123f.txt` file located in the microSD needs to be updated. To update it, set the PMIC image (`300k_8c2.bin`) to be used in the newer models by running the following commands on the Build Host:

```
sudo umount /dev/sdb1
sudo mkdir -p /tmp/sdcard
sudo mount /dev/sdb1 /tmp/sdcard
sudo sed -i '/^MBPMIC: pms_0V85.bin/s/^/;/g' /tmp/sdcard/MB/HBI0316A/io_v123f.
↪txt
sudo sed -i '/^;MBPMIC: 300k_8c2.bin/s/^;//g' /tmp/sdcard/MB/HBI0316A/io_v123f.
↪txt
sudo umount /tmp/sdcard
sudo rmdir /tmp/sdcard
```

To run the deployed EWAOL distribution image, simply reboot the target platform by running the following command on the MCC console:

```
REBOOT
```

Once the reboot has occurred, the EWAOL distribution boot process should then be output to the MCC console. After the boot process has completed, the EWAOL log-in prompt should appear and the distribution has been successfully deployed.

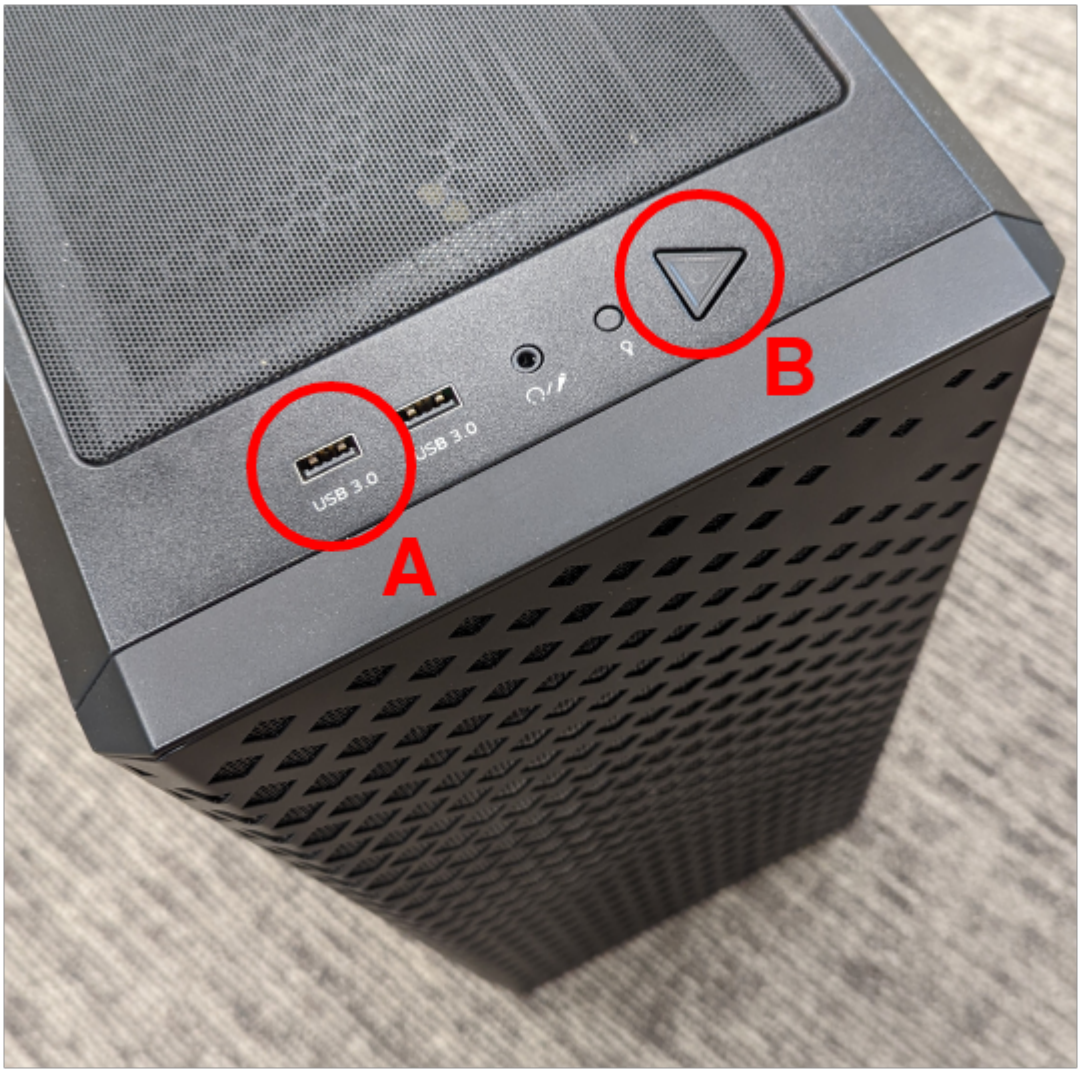
AVA

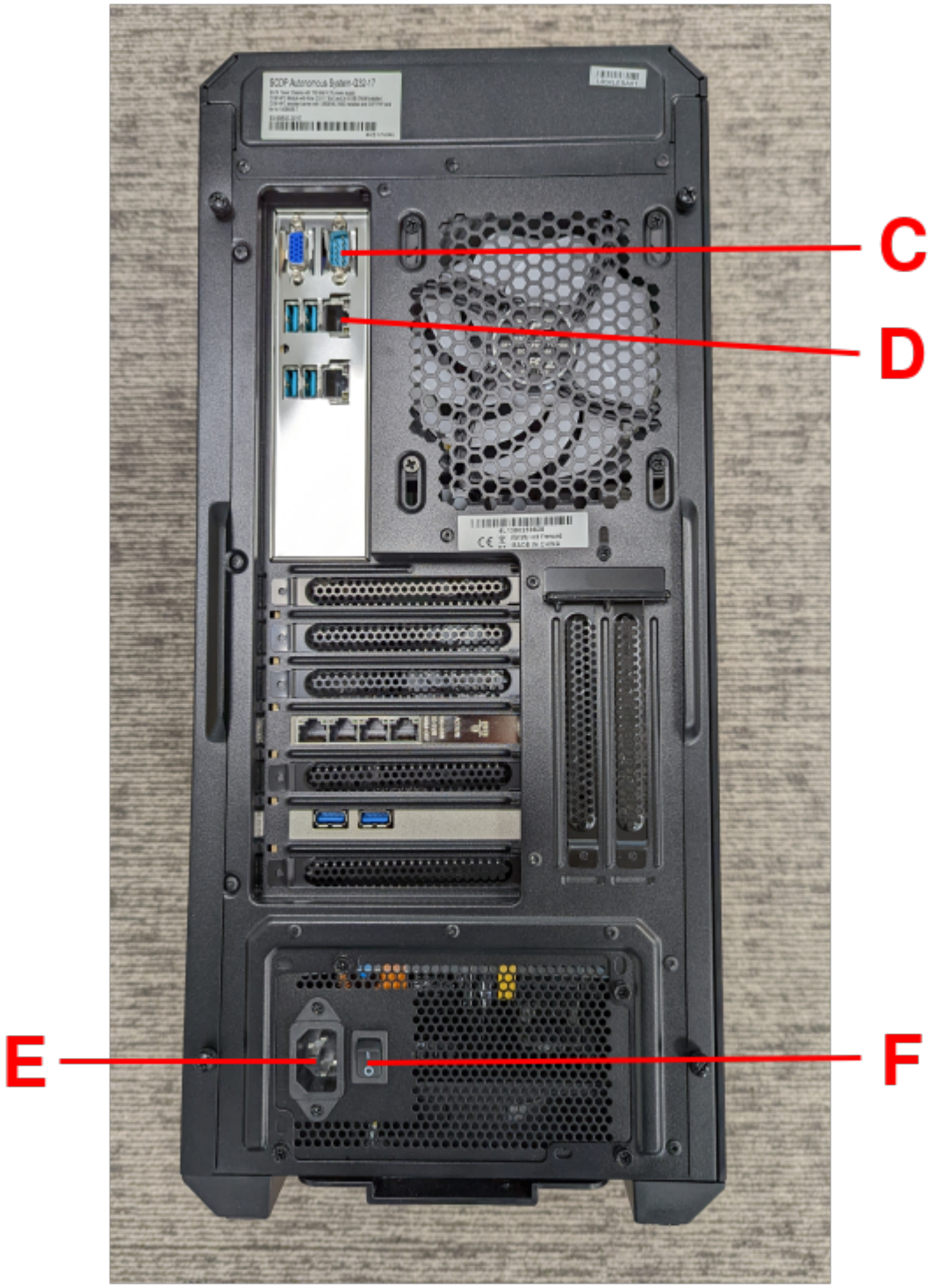
Note: To use the AVA Developer Platform, please make sure the latest available firmware is installed. See the ADLINK's AVA Developer Platform documentation for guidance and support on installing the latest firmware. The following instructions and supporting images were created using Tianocore/EDK2 version `1.07.300.02b Build 20220302`.

Instructions for deploying an EWAOL distribution image on the AVA hardware target platform are divided into three parts:

1. *Load the AVA Flasher Image onto a USB Storage Device*
2. *Boot AVA into the Flasher Image Loaded on the USB Storage Device*
3. *Flash the EWAOL Distribution Image onto the AVA NVMe M.2 Storage Device*

The following two images, with reference labels given in red, are provided to support these instructions:


USB 3.0
USB 3.0
A
B


C
D
E
F

1. Load the AVA Flasher Image onto a USB Storage Device

First, it is necessary to use the Build Host to load AVA's bootable 'Flasher Image' onto a USB storage device. This will later be connected to the AVA Developer Platform and used to boot the machine. The steps to do this are as follows:

1. **On the Build Host:** run the following commands to download and unpack the AVA Flasher Image from ADLINK's AVA Developer Platform Downloads Page into an appropriate storage directory, here created as `~/ava_flasher_image`:

```
mkdir -p ~/ava_flasher_image && cd ~/ava_flasher_image
wget https://hq0epm0west0us0storage.blob.core.windows.net/%24web/public/COMe/Ampere/
↪AVA/Images/Yocto/adlink-flasher-image-ava.tar.xz
tar -xJf adlink-flasher-image-ava.tar.xz && cd adlink-flasher-image-ava
```

2. **On the Build Host:** connect a USB storage device (minimum size of 64 GB) and identify it using the `lsblk` command:

```
lsblk
```

   This will output, for example:

```
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sdc      8:0    0    64G  0 disk
...
```

Warning: In this example, the USB storage device is the `/dev/sdc` device. As this may vary on different machines, care should be taken when copying and pasting the following commands.

3. **On the Build Host:** prepare for the Flasher Image copy:

```
sudo umount /dev/sdc*
cd ~/ava_flasher_image
```

Warning: The next step will result in all prior partitions and data on the USB storage device being erased. Please backup before continuing.

3. **On the Build Host:** transfer the Flasher Image onto the USB storage device using the `bmaptool` utility:

```
sudo bmaptool copy --bmap adlink-flasher-image-ava.wic.bmap adlink-flasher-image-
↪ava.wic.gz /dev/sdc
```

4. Safely eject the USB storage device from the Build Host.

2. Boot AVA into the Flasher Image Loaded on the USB Storage Device

Next, prepare the AVA Developer Platform as follows.

5. Connect a USB to RS232 female DB9 serial converter cable between the Build Host and the `Serial Console` port on the AVA back-panel, marked `C` in the *reference images*.
6. Connect the AVA Developer Platform to the network via the `GbE System (In Band)` ethernet port, marked `D` in the *reference images*.
7. Provide power to the AVA Developer Platform via a C13 mains power cable connected to the `Power Input` port, marked `E` in the *reference images*.
8. Switch the AVA Developer Platform's `Power Main Switch` on, marked `F` in the *reference images*.
9. Connect the USB storage device containing the AVA Flasher Image to the AVA Developer Platform using a USB 3.0 port, marked `A` in the *reference images*.

Then, set up the Build Host to access the AVA Developer Platform via a serial console:

10. **On the Build Host:** find the TTY USB device in the `/dev` directory that corresponds to the serial connection from the Build Host to the AVA Developer Platform that was set up in step 5, via:

    ```
    ls /dev/ttyUSB*
    ```

    In this example, the corresponding TTY USB device is assumed to be `/dev/ttyUSB0`.

    The port should be configured with the following settings:

    - 115200 Baud
    - 8N1
    - No hardware or software flow support

11. **On the Build Host:** set up a terminal to interface with the AVA Developer Platform's serial console. This terminal will be referred to as the 'Serial Console Terminal'. Any terminal applications such as `putty`, `screen` or `minicom` will work. The `screen` utility is used in the following command:

    ```
    sudo screen /dev/ttyUSB0 115200
    ```

12. Power-on the AVA Developer Platform via the power button, marked `B` in the *reference images*.

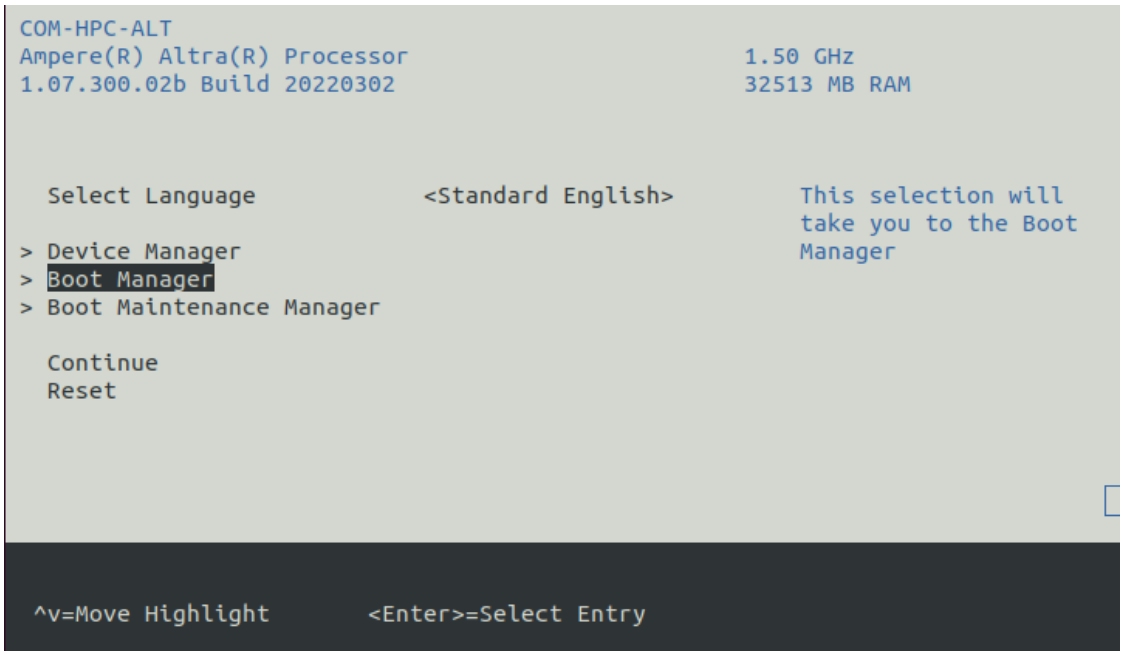
    The Serial Console Terminal should then start receiving output from the AVA boot process.

13. **On the Serial Console Terminal:** interrupt the boot process to access the boot options menu, by entering `ESCAPE` at the prompt (by pressing the ESC key once on the keyboard) shown in the following image:

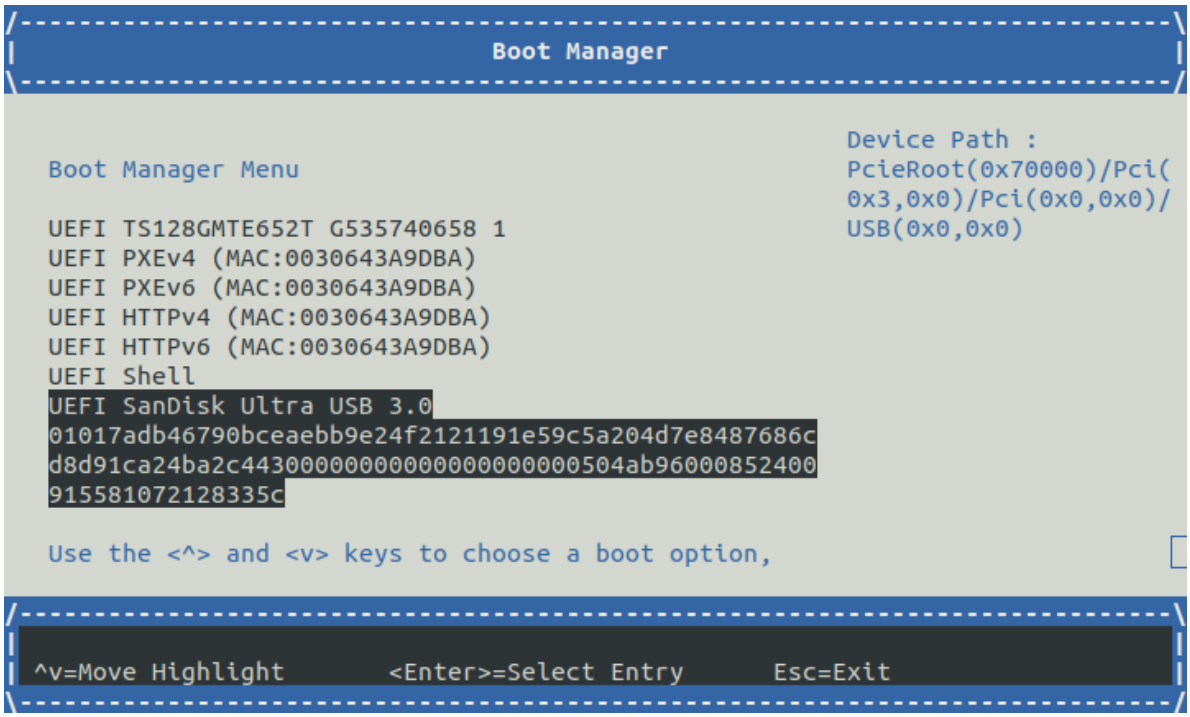
    ```
    PROGRESS CODE: V01040001 I0
    PROGRESS CODE: V02020000 I0
    PROGRESS CODE: V02020004 I0
    PROGRESS CODE: V02020003 I0
    PROGRESS CODE: V02020000 I0
    PROGRESS CODE: V02020004 I0
    PROGRESS CODE: V02020003 I0
    Tianocore/EDK2 firmware version 1.07.300.02b Build 20220302
    Press ESCAPE for boot options PROGRESS CODE: V02020006 I0
    PROGRESS CODE: V02020006 I0
    ```

This should provide access to the EDK2 interface shown in the following image:

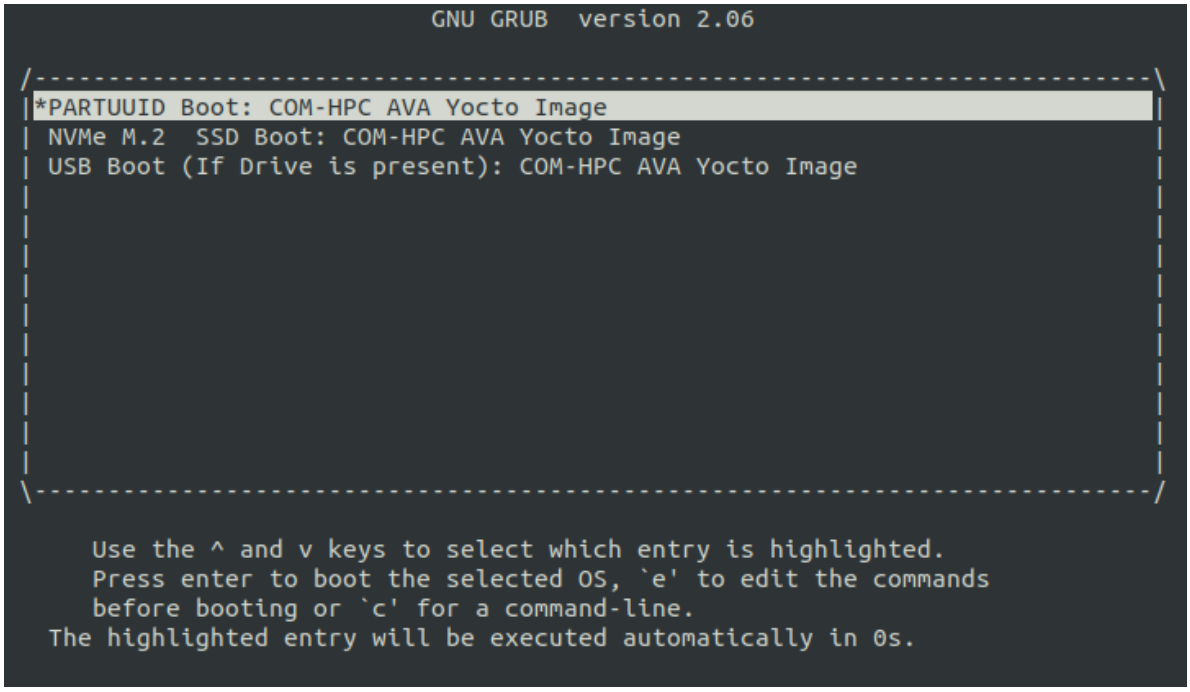
14. **On the Serial Console Terminal:** move to the `Boot Manager` entry using the arrow keys, and select it by pressing the `ENTER` key:



15. **On the Serial Console Terminal:** the connected USB storage device containing the AVA Flasher Image should then be visible in the `Boot Manager Menu`. Highlight that USB storage device entry using the arrow keys and select it by pressing the `ENTER` key. An example `Boot Manager Menu` showing a connected USB storage device is shown in the following image:



16. **On the Serial Console Terminal:** a GRUB2 boot menu will appear as shown in the following image:



Either select the highlighted entry, or wait for it to be selected automatically.

Output from the AVA Flasher Image boot process should then appear on the Serial Console Terminal, and this process should result in a Linux console, with no manual account log-in required, such as the following:

```
Poky (Yocto Project Reference Distro) 4.0.1 ava ttyAMA0

ava login: root (automatic login)

root@ava:~#
```

3. Flash the EWAOL Distribution Image onto the AVA NVMe M.2 Storage Device

To flash the EWAOL distribution image onto the AVA's persistent storage, it must first be transferred to the USB storage device which is running the AVA Flasher Image on the AVA Developer Platform. The steps for doing this are as follows:

17. **On the Build Host:** create or swap to a **different** terminal from that used for the Serial Console Terminal, such as the one that was used to execute the `kas build` commands during the *build instructions* described previously. This terminal will be referred to as the 'Build Host Terminal'.

18. **On the Build Host Terminal:** change the working directory to the directory which contains the Yocto build folder (here assumed to be the root directory of the cloned `meta-ewaol` repository), and prepare for the EWAOL distribution image copy:

    A. Baremetal

    ```
    cd build/tmp_baremetal/deploy/images/ava/
    ```

    B. Virtualization

```
cd build/tmp_virtualization/deploy/images/ava/
```

19. **On the Serial Console Terminal:** determine the IP address associated with the AVA Flasher Image running on the AVA Developer Platform, by running the following command:

```
ifconfig eth0 | grep "inet addr"
```

Running this command will output, for example:

```
inet addr:[IP]  Bcast:10.1.195.255  Mask:255.255.254.0
```

The relevant IP address to extract is denoted `[IP]` in this example output, which is also used to the reference the IP address in the next step.

20. **On the Serial Console Terminal:** define an environment variable to hold the IP address and allow copy-pasting of the following commands, by running:

```
export TARGET_IP=[IP]
```

Be sure to replace `[IP]` in this command with the IP address determined in the previous step.

21. **On the Build Host Terminal:** transfer the EWAOL distribution image to the AVA Developer Platform using the `scp` utility. The command to run depends on the target EWAOL distribution image:

    A. Baremetal distribution image:

```
scp ewaol-baremetal-image-ava.wic.* root${TARGET_IP}:/tmp/
```

    B. Baremetal-SDK distribution image:

```
scp ewaol-baremetal-sdk-image-ava.wic.* root@${TARGET_IP}:/tmp/
```

    C. Virtualization distribution image:

```
scp ewaol-virtualization-image-ava.wic.* root@${TARGET_IP}:/tmp/
```

    D. Virtualization-SDK distribution image:

```
scp ewaol-virtualization-sdk-image-ava.wic.* root@${TARGET_IP}:/tmp/
```

22. **On the Serial Console Terminal:** once the file transfer has completed, flash the EWAOL distribution image to the AVA NVMe M.2 storage device using the `bmaptool` utility.

Note: This guidance assumes that the AVA Developer Platform storage drives and partitions have not been modified, and no additional storage devices have been connected other than those described in these instructions. The AVA NVMe M.2 storage device therefore corresponds to the `/dev/nvme0n1` device.

Warning: The next step will result in all prior partitions and data on the AVA NVMe M.2 storage device to be erased.

    A. Baremetal distribution image:

```
bmaptool copy --bmap /tmp/ewaol-baremetal-image-ava.wic.bmap /tmp/ewaol-
↪baremetal-image-ava.wic.gz /dev/nvme0n1
```

B. Baremetal-SDK distribution image:

```
bmaptool copy --bmap /tmp/ewaol-baremetal-sdk-image-ava.wic.bmap /tmp/ewaol-
↪baremetal-sdk-image-ava.wic.gz /dev/nvme0n1
```

C. Virtualization distribution image:

```
bmaptool copy --bmap /tmp/ewaol-virtualization-image-ava.wic.bmap /tmp/
↪ewaol-virtualization-image-ava.wic.gz /dev/nvme0n1
```

D. Virtualization-SDK distribution image:

```
bmaptool copy --bmap /tmp/ewaol-virtualization-sdk-image-ava.wic.bmap /tmp/
↪ewaol-virtualization-sdk-image-ava.wic.gz /dev/nvme0n1
```

23. On the Serial Console Terminal: once the `bmaptool` process has complete, power-off the AVA Developer Platform by running:

```
poweroff
```

24. Remove the USB storage device containing the AVA Flasher Image from the AVA Developer Platform.
25. Power-on the AVA Developer Platform via the power button, marked `B` in the *reference images*.

The EWAOL distribution boot process should then be output to the Serial Console Terminal. After the boot process has completed, the EWAOL log-in prompt should appear and the distribution has been successfully deployed.

Run

The EWAOL distribution image can be logged into as `ewaol` user. See *User Accounts* for more information about user accounts and groups.

On an EWAOL virtualization distribution image, this will access the Control VM. To log into a Guest VM, the `xl` tool can be used. For example, on a default EWAOL virtualization distribution image:

```
sudo xl console ewaol-guest-vm1
```

This command will provide a console on the Guest VM, which can be exited by entering `Ctrl+]`. See the xl documentation for further details.

The distribution can then be used for deployment and orchestration of application workloads in order to achieve the desired use-cases.

Validate

As an initial validation step, check that the appropriate Systemd services are running successfully, depending on the target architecture:

- Baremetal Architecture:
  - `docker.service`
  - `k3s.service`

  These services can be checked by running the command:

```
systemctl status --no-pager --lines=0 docker.service k3s.service
```

And ensuring the command output lists them as active and running.

- Virtualization Architecture:
  - `docker.service`
  - `k3s.service`
  - `xendomains.service`

  These services can be checked by running the command:

  ```
  systemctl status --no-pager --lines=0 docker.service k3s.service xendomains.
  ↪service
  ```

  And ensuring the command output lists them as active and running.

More thorough run-time validation of EWAOL components are provided as a series of integration tests, available if the `meta-ewaol-config/kas/tests.yml` kas configuration file was included in the image build. These are detailed at *Run-Time Integration Tests*.

The integration tests that this command will execute are detailed in *Validation*, along with the expected format of the test output and additional details for running and customizing the validation.

Reproducing the EWAOL Use-Cases

With the EWAOL distribution running and validated, it can be used to achieve the target *EWAOL Use-Cases*.

This section briefly demonstrates simplified use-case examples, where detailed instructions for developing, deploying, and orchestrating application workloads are left to the external documentation of the relevant technology, as stated in the *Documentation Assumptions*.

Note: The following example instructions form similar but simplified versions of the activities carried out by the run-time validation tests that can be included on the EWAOL distribution. See *Validation* and the test implementations for further demonstrations of EWAOL use-cases.

Deploying Application Workloads via Docker and K3s

This example use-case is performed on the:

- Baremetal distribution image
- Virtualization distribution image

This example deploys the Nginx webserver as an application workload, using the `nginx` container image available from Docker's default image repository. The deployment can be achieved either via Docker or via K3s, as follows:

1. Reboot the image and log-in as the `ewaol` user.

   On a virtualization distribution image, this will produce a console on the Control VM.

2. Deploy the example application workload:

   - **Deploy via Docker**

     2.1. Run the following example command to deploy via Docker:

     ```
     sudo docker run --name nginx_docker_example -p 8082:80 -d nginx
     ```

2.2. Confirm the Docker container is running by checking its `STATUS` in the container list:

```
sudo docker container list
```

The container should appear in the list of running containers, with the associated name `nginx_docker_example`. For example:

```
CONTAINER ID   IMAGE     COMMAND                  CREATED         ␣
↪STATUS          PORTS                                    NAMES
cb7f67053556   nginx     "/docker-entrypoint...."   14 seconds ago   Up␣
↪13 seconds   0.0.0.0:8082->80/tcp, :::8082->80/tcp   nginx_docker_
↪example
```

- **Deploy via K3s**

2.1. Run the following example command to deploy via K3s:

```
cat << EOT > nginx-example.yml && sudo kubectl apply -f nginx-example.yml
apiVersion: v1
kind: Pod
metadata:
  name: k3s-nginx-example
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
      hostPort: 8082
EOT
```

2.2. Confirm that the K3s Pod hosting the container is running by checking that its `STATUS` is `running`, using:

```
sudo kubectl get pods -o wide
```

The output should be similar to the following example output, which was captured on the N1SDP:

```
NAME                 READY   STATUS    RESTARTS   AGE   IP          NODE ␣
↪  NOMINATED NODE   READINESS GATES
k3s-nginx-example   1/1     Running   0          28s   [IP]   n1sdp
↪<none>           <none>
```

3. After the Nginx application workload has been successfully deployed, it can be interacted with on the network, via for example:

```
wget localhost:8082
```

This should download the webserver's default `index.html` page and return a successful exit status, similar to the following example output:

```
--YYYY-MM-DD HH:mm:ss--  http://localhost:8082/
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:8082... connected.
HTTP request sent, awaiting response... 200 OK
```

(continues on next page)

(continued from previous page)

```
Length: 615 [text/html]
Saving to: ‘index.html’

index.html                                  100
↪%[=========================================================================>]  ␣
↪  615  --.-KB/s    in 0s

YYYY-MM-DD HH:mm:ss (189 MB/s) - ‘index.html’ saved [615/615]
```

Note: As both methods deploy a webserver listening on port 8082, the two methods cannot be run simultaneously and one deployment must be stopped before the other can start.

To stop the application workload deployed via Docker, use the command:

```
sudo docker stop nginx_docker_example
```

The container should then no longer appear in the list of running containers given by `sudo docker container list`.

To stop the application workload deployed via K3s, use the command:

```
sudo kubectl delete pod k3s-nginx-example
```

The K3s Pod which was running the container should no longer appear in the list of K3s Pods given by `sudo kubectl get pods -o wide`.

Orchestrating Resource-Managed and Isolated Application Workloads via K3s and Xen VMs

This example use-case is performed on the:

- Virtualization distribution image

This example uses the K3s orchestration framework to use the Control VM to schedule an Nginx webserver application workload for execution on the Guest VM.

To do this, it is first necessary for a K3s agent to be initialized on the Guest VM and connected with the K3s server running on the Control VM, to form a cluster. The application workload can then be scheduled for deployment to the Guest VM via K3s orchestration. This example process is as follows:

1. **Log-in to the Control VM**

   Reboot the virtualization distribution image, then log-in as the `ewaol` user.

2. **Connect Guest VM K3s Agent**

   2.1. On the **Control VM**, determine its IP address via:

   ```
   ifconfig xenbr0
   ```

   2.2. On the **Control VM**, determine the node-token for the K3s server via:

   ```
   sudo cat /var/lib/rancher/k3s/server/node-token
   ```

   2.3. On the **Control VM**, log in to the **Guest VM** as the `ewaol` user, via:

```
sudo xl console ewaol-guest-vm1
```

2.4. On the **Guest VM**, and denoting the IP address and node-token as `[IP]` and `[TOKEN]` respectively, change the `ExecStart=` line in `/lib/systemd/system/k3s-agent.service` to:

```
ExecStart=/usr/local/bin/k3s agent --server=https://[IP]:6443 --
↪token=[TOKEN] --node-label=ewaol.node-type=guest-vm
```

2.5. On the **Guest VM**, start the K3s Agent with these values via:

```
sudo systemctl daemon-reload && sudo systemctl start k3s-agent
```

2.6. On the **Guest VM**, disconnect from it and return to the Control VM via:

```
Ctrl+]
```

2.7. On the **Control VM**, ensure that the K3s server and the Guest VM's K3s agent are connected, by running:

```
sudo kubectl get nodes
```

The hostname of the Guest VM should appear as a node in the list, with a `STATUS` of `ready`. The output should be similar to the following example, produced when running this step on the N1SDP:

```
NAME              STATUS   ROLES                  AGE    ␣
↪VERSION
ewaol-guest-vm1   Ready    <none>                 22s    v1.22.
↪6-k3s1
n1sdp             Ready    control-plane,master   6m40s  v1.22.
↪6-k3s1
```

3. **Schedule Application Workload**

3.1. On the **Control VM**, schedule the Nginx application workload to be deployed on the Guest VM, by running the following example command:

```
cat << EOT > nginx-example.yml && sudo kubectl apply -f nginx-
↪example.yml
apiVersion: v1
kind: Pod
metadata:
  name: k3s-nginx-example
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
      hostPort: 8082
  nodeSelector:
    ewaol.node-type: guest-vm
EOT
```

3.2. On the **Control VM**, confirm that the K3s Pod (which hosts the container) was deployed to the Guest VM by checking its `STATUS` is `running` and its `NODE` is the Guest VM's hostname, by running the following command:

```
sudo kubectl get pods -o wide
```

The output should be similar to the following example output:

```
NAME                READY   STATUS    RESTARTS   AGE   IP          ␣
↪   NODE              NOMINATED NODE   READINESS GATES
k3s-nginx-example   1/1     Running   0          33s   [IP]  ␣
↪ewaol-guest-vm1   <none>           <none>
```

4. **Access the Application Workload**

The webserver will then be running on the Guest VM. To access the webserver:

4.1. On the **Control VM**, log in to the **Guest VM** as the `ewaol` user, via:

```
sudo xl console ewaol-guest-vm1
```

4.2. On the **Guest VM**, access the webserver by running the following example command:

```
wget localhost:8082
```

This should download the webserver's default `index.html` page and return a successful exit status, similar to the following example output:

```
--YYYY-MM-DD HH:mm:ss--  http://localhost:8082/
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:8082... connected.
HTTP request sent, awaiting response... 200 OK
Length: 615 [text/html]
Saving to: ‘index.html’

index.html                          100
↪%[=================================================================================>
↪]     615  --.-KB/s    in 0s

YYYY-MM-DD HH:mm:ss (189 MB/s) - ‘index.html’ saved [615/615]
```

While the Guest VM is running this application workload, other deployments may be carried out (for example) on the Control VM, thus enabling isolation between application workloads running on resource-managed virtualized hardware.

Describes how to reproduce an EWAOL distribution image for a supported target platform, configuring and deploying the supported set of distribution features, and running example EWAOL use-cases.

1.2.2 Extend

This section of the User Guide describes how to extend EWAOL in order to configure and build it for deployment on custom or unsupported target platforms, via kas.

Porting EWAOL to a Custom or Unsupported Target Platform via kas

To build an EWAOL distribution image that targets an externally defined target platform using `meta-ewaol`, a kas configuration file must be defined and added to the external Yocto BSP or application layer. For example, `my-machine.yml` (where `my-machine` is the `MACHINE` name of the custom or unsupported target platform) defined in a custom BSP layer `meta-my-bsp-layer` should have the following structure to build a baremetal distribution image:

```
header:
  version: 11
  includes:
    - repo: meta-ewaol
      file: meta-ewaol-config/kas/baremetal.yml
    - repo: meta-ewaol
      file: meta-ewaol-config/kas/tests.yml

repos:
  meta-my-bsp-layer:

  meta-ewaol:
    url: https://gitlab.com/soafee/ewaol/meta-ewaol.git
    refspec: kirkstone-dev

machine: my-machine
```

This example kas configuration file for the `my-machine` target platform defines the Yocto project configuration build via the kas configuration files that are added in the `includes` section. These inclusions can be customized in order to build a different image. For example, to build a virtualization distribution image with run-time validation tests, include `meta-ewaol-config/kas/virtualization.yml` instead of `meta-ewaol-config/kas/baremetal.yml` in the above example.

Images for `my-machine` can then be built using this example kas configuration file by running the following kas command:

```
kas build --update meta-my-bsp-layer/my-machine.yml
```

Describes how to extend the EWAOL project to build for a custom or unsupported target platform.

1.2.3 Migrating to Later Releases

This page describes guidance for updating a user's build environment and processes from those required to use a previous EWAOL release, to instead setup and use a later EWAOL release, as part of a migration process.

Details of the EWAOL release process can be found at *Codeline Management*, and a summary of each release can be found at *Changelog & Release Notes*.

The following migration guidance is described such that the required changes are with respect to the previous EWAOL release. The processes are categorized according to the associated section of the user-guide documentation.

To v1.0

After following the below guidance to transition to EWAOL `v1.0`, boot the resulting image to run and validate the release.

EWAOL Reproduce Migration

Build Host Setup

- The list of essential packages and package versions for the associated release of the Yocto Project was updated. Refer to the list of essential packages documentation to ensure the necessary packages are installed and upgraded to support the migration.
- The supported version of the `kas` build tool was updated to `3.1`. See *Build Host Environment Setup* for guidance on installing this version of `kas`.

Download

- To migrate to `v1.0`, it is necessary to download that version of the `meta-ewaol` repository source. To do this, either clone the repository into a new directory by following the instructions given in *Download*, or update the existing local repository by switching to the `v1.0` tag using Git.

Build

- The kas configuration files provided to build and customize EWAOL distribution images have been updated, and it is necessary to supply them to the kas build tool in a particular order. Therefore, to build the later version of EWAOL, refer to the *Build Documentation*.
- If working from an existing local `meta-ewaol` repository that was switched to the `v1.0` release but has artifacts remaining from previous builds, ensure that there are no locally staged changes to the dependent layers so that the kas build tool can successfully update them.

Deploy

- `v1.0` introduces new types of EWAOL distribution images, and the resulting filenames and paths are therefore different than previous releases. Check the *Build* and *Deploy* sections of the release documentation for the new filenames produced during the build, and which should be deployed to the target platform.

EWAOL Extend Migration

Porting

- If migrating to EWAOL `v1.0` as part of porting to a custom or unsupported target platform, it is necessary to change the existing custom kas configuration file for the target platform to use the correct kas configuration files, and supply them with the correct ordering in the `includes` YAML section. In addition, the `meta-ewaol` repository definition must be set to use the `v1.0` release tag in `refspec:`. See *Porting* for details of the necessary configuration.

Guidance for migrating an existing EWAOL build environment to a later EWAOL release.

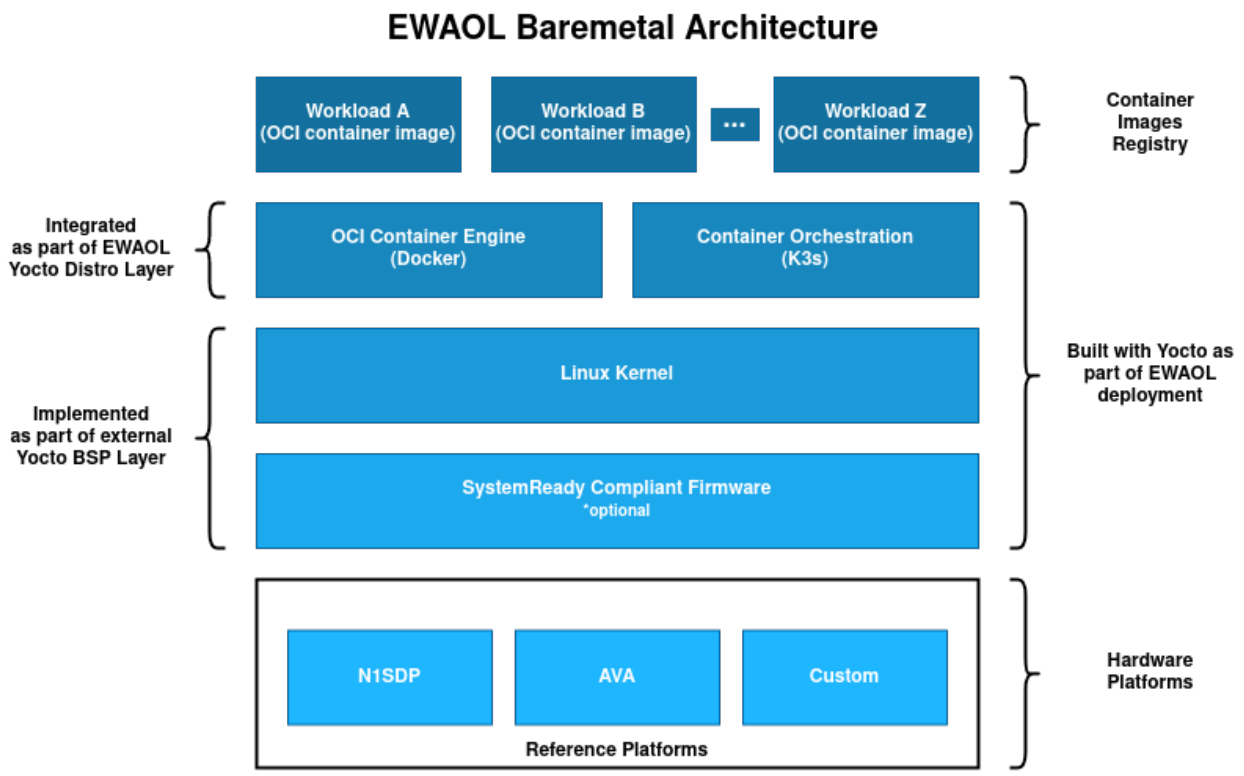
1.3 Developer Manual

1.3.1 System Architectures

Introduction

This page describes the two target architectures supported by the EWAOL project.

Baremetal Architecture



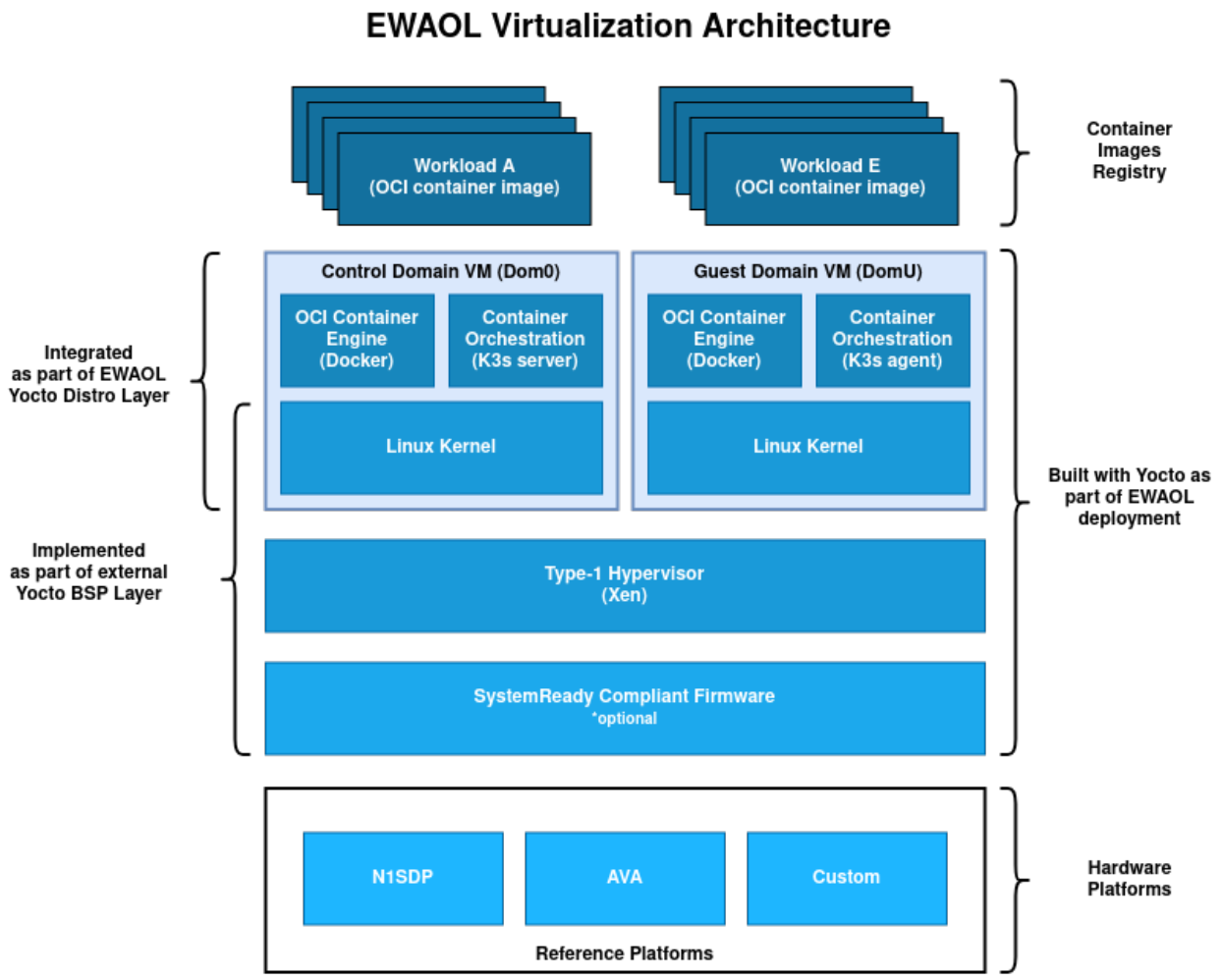
An EWAOL baremetal distribution image (`ewaol-baremetal-image`) is enabled if `ewaol-baremetal` is included in `DISTRO_FEATURES`. The image includes the following image features by default:

- Container engine and runtime with Docker and runc-opencontainers
- Container workload orchestration with the K3s Kubernetes distribution

On a baremetal distribution image system boot, a K3s Systemd service (`k3s.service`) provides a container orchestration environment consisting of a single K3s control plane and a single built-in K3s agent, communicating via the local loopback network interface. This enables the orchestration and execution of containerized application workloads on the baremetal distribution image, operating as a single-node K3s cluster.

See *Baremetal Architecture* for details on building an EWAOL baremetal distribution image.

Virtualization Architecture



An EWAOL virtualization distribution image (`ewaol-virtualization-image`) is enabled if `ewaol-virtualization` is included in `DISTRO_FEATURES`. The image includes the following image features by default:

- Hardware virtualization support with the Xen type-1 hypervisor
- Container engine and runtime with Docker and runc-opencontainers
- Container workload orchestration with the K3s Kubernetes distribution

On an EWAOL virtualization distribution image, the software stack includes the Xen type-1 hypervisor and provides a Control VM (Dom0) and a single bundled Guest VM (DomU), by default. Virtualization support also includes Xen-related configurations and necessary Xen-management packages into the Control VM root filesystem.

The Control VM includes the `xen-tools` package along with network configuration for the `xenbr0` bridge, to allow the Guest VM to access the external network. By default, the bundled Guest VM image is based on the `generic-arm64` `MACHINE`.

A Guest VM is included into the EWAOL virtualization distribution image via the `ewaol-guest-vm-package` recipe, with the Guest VM's rootfs stored as a raw image file in `*.qcow2` format. In addition, this package includes a sample Xen domain configuration file, which holds Xen-specific Guest VM settings as detailed in xl domain configuration. By default one Guest VM (with hostname `ewaol-guest-vm1`) is built and included on the virtualization distribution image.

See *Virtualization Architecture* for details on building an EWAOL virtualization distribution image.

An EWAOL virtualization distribution image can be customized, including setting the number of included Guest VMs. The supported virtualization-specific customization parameters and how to set them are detailed at *Customization*.

1.3.2 User Accounts

EWAOL distribution images contain the following user accounts:

- `root` with administrative privileges enabled by default. The login is disabled if `ewaol-security` is included in `DISTRO_FEATURES`.
- `ewaol` with administrative privileges enabled with `sudo`.
- `user` without administrative privileges.
- `test` with administrative privileges enabled with `sudo`. This account is created only if `ewaol-test` is included in `DISTRO_FEATURES`.

By default, each users account has disabled password. The default administrative group name is `sudo`. Other sudoers configuration is included in `meta-ewaol-distro/recipes-extended/sudo/files/ewaol_admin_group.in`. For virtualization images, above user accounts are created for Control VM and Guest VM domains.

If `ewaol-security` is included in `DISTRO_FEATURES`, each user is prompted to a set new password on first login. For more information about security see: *Security Hardening*.

All *Run-Time Integration Tests* are executed as the `test` user.

An EWAOL distribution image can be configured to include run-time integration tests that validate successful configuration of the EWAOL user accounts. Details of the user accounts validation tests can be found in the *User Accounts Tests* section of the *Validation* documentation.

1.3.3 Build System

An EWAOL build can be configured by setting the target platform via the `MACHINE` BitBake variable, the desired distribution image features via the `DISTRO_FEATURES` BitBake variable, and customizing those features via feature-specific modifiable variables.

This page first overviews EWAOL's support for the kas build tool. Each available distribution image feature and each supported target platform is then defined together with their associated kas configuration files, followed by any other additional customization options. The process for building without kas is then briefly described.

kas Build Tool Support

The kas build tool enables automatic fetch and inclusion of layer sources, as well as parameter and feature specification for building target images. To enable this, kas configuration files in the YAML format are passed to the tool to provide the necessary definitions.

These kas configuration files are modular, where passing multiple files will result in an image produced with their combined configuration. Further, kas configuration files can extend other kas configuration files, thereby enabling specialized configurations that inherit common configurations.

The `meta-ewaol-config/kas` directory contains kas configuration files that support building images via kas for the EWAOL project, and fall into three ordered categories:

- **Architecture Configs**
- **Build Modifier Configs**
- **Target Platform Configs**

To build an EWAOL distribution image via kas, it is required to provide one **Architecture Config** and one **Target Platform Config**, unless otherwise stated in their descriptions below. Additional Build Modifiers are optional, and depend on the target use-case. Currently, it is necessary that kas configuration files are provided in order: the **Architecture Config** is defined first, then additional build features via zero or more **Build Modifier Configs**, and finally the **Target Platform Config**.

The kas configuration files to enable builds for a supported target platform or to configure each EWAOL distribution image feature, are described in their relevant sections below: *Target Platforms* and *Distribution Image Features*, respectively. Example usage of these kas configuration files can be found in the *Build* section of the User Guide.

Note: If a kas configuration file does not set a particular build parameter, the parameter will take its default value.

Target Platforms

There are currently two supported target platforms (corresponding to the `MACHINE` BitBake variable), each with an associated kas configuration file, as follows.

N1SDP

- **Corresponding value for `MACHINE` variable**: `n1sdp`.
- **Target Platform Config**: `meta-ewaol-config/kas/n1sdp.yml`.

This supported target platform is the Neoverse N1 System Development Platform (N1SDP), implemented in the meta-arm-bsp Yocto BSP layer.

To enable this, the `n1sdp.yml` Target Platform Config includes common configuration from the `meta-ewaol-config/kas/include/arm-machines.yml` kas configuration file, which defines the BSPs, layers, and dependencies required when building for the `n1sdp`.

AVA

- **Corresponding value for `MACHINE` variable**: `ava`.
- **Target Platform Config**: `meta-ewaol-config/kas/ava.yml`.

This supported target platform is the AVA Developer Platform (AVA), implemented in the meta-adlink-ampere Yocto BSP layer.

Distribution Image Features

For a particular target platform, the available EWAOL distribution image features (corresponding to the contents of the `DISTRO_FEATURES` BitBake variable) are detailed in this section, along with any associated kas configuration files, and any associated customization options relevant for that feature.

EWAOL Architectures

EWAOL distribution images can be configured via kas using Architecture Configs. These include a set of common configuration from a base EWAOL kas configuration file:

- `meta-ewaol-config/kas/include/ewaol-base.yml`

This kas configuration file defines the base EWAOL layer dependencies and their software sources, as well as additional build configuration variables. It also includes the `meta-ewaol-config/kas/include/ewaol-release.yml` kas configuration file, where the layers dependencies are pinned for any corresponding EWAOL release.

Baremetal Architecture

- **Corresponding value in `DISTRO_FEATURES` variable**: `ewaol-baremetal`.
- **Architecture Config**: `meta-ewaol-config/kas/baremetal.yml`.

  This EWAOL distribution image feature:

  - Enables the `ewaol-baremetal-image` build target, to build an EWAOL baremetal distribution image.
  - Is incompatible with the `ewaol-virtualization` distribution image feature.

  The Architecture Config for this distribution image feature automatically appends `ewaol-baremetal` to `DISTRO_FEATURES` and sets the sets the build target to `ewaol-baremetal-image`.

  To build EWAOL baremetal distribution image, provide the Baremetal Architecture Config to the kas build command. For example, to build an EWAOL baremetal distribution image for the N1SDP hardware target platform, run the following command:

```
kas build meta-ewaol-config/kas/baremetal.yml:meta-ewaol-config/kas/n1sdp.
↪yml
```

Virtualization Architecture

- **Corresponding value in** `DISTRO_FEATURES` **variable**: `ewaol-virtualization`.
- **Architecture Config**: `meta-ewaol-config/kas/virtualization.yml`.

  This EWAOL distribution image feature:

  - Enables the `ewaol-virtualization-image` build target, to build an EWAOL virtualization distribution image.
  - Includes the Xen hypervisor into the software stack.
  - Enables Xen specific configs required by kernel.
  - Includes all necessary packages and adjustments to the Control VM's root filesystem to support management of Xen Guest VMs.
  - Uses BitBake Multiple Configuration Build.
  - Includes a single Guest VM based on the `generic-arm64` `MACHINE`, by default.
  - Is incompatible with the `ewaol-baremetal` distribution image feature.

  The Architecture Config for this distribution image feature automatically appends `ewaol-virtualization` to `DISTRO_FEATURES` and sets the sets the build target to `ewaol-virtualization-image`.

  To build EWAOL virtualization distribution image, provide the Virtualization Architecture Config to the kas build command. For example, to build an EWAOL virtualization distribution image for the N1SDP hardware target platform, run the following command:

```
kas build meta-ewaol-config/kas/virtualization.yml:meta-ewaol-config/kas/
↪n1sdp.yml
```

Customization

Configurable build-time variables for the Guest VM are defined within the `meta-ewaol-distro/conf/multiconfig/ewaol-guest-vm.conf` file and the `meta-ewaol-distro/conf/distro/include/ewaol-guest-vm.inc` which it includes.

The following list shows the standard set of available variables for the Control VM and the single default Guest VM, together with the default values (where `MB` and `KB` refer to Megabytes and Kilobytes, respectively):

```
EWAOL_GUEST_VM_INSTANCES: "1"                       # Number of Guest VM␣
↪instances
EWAOL_GUEST_VM1_NUMBER_OF_CPUS: "4"                 # Number of CPUs for Guest␣
↪VM1
EWAOL_GUEST_VM1_MEMORY_SIZE: "6144"                 # Memory size for Guest VM1␣
↪(MB)
EWAOL_GUEST_VM1_ROOTFS_EXTRA_SPACE: ""              # Extra storage space for␣
↪Guest VM1 (KB)
EWAOL_CONTROL_VM_MEMORY_SIZE: "2048"                # Memory size for Control␣
↪VM (MB)
EWAOL_CONTROL_VM_ROOTFS_EXTRA_SPACE: "0"            # Extra storage space for␣
↪Control VM (KB), added as additional space above the storage necessary to␣
↪support all Guest VMs root filesystems
```

The variables may be set either within an included kas configuration file (see `meta-ewaol-config/kas/virtualization.yml` for example usage), the environment, or manually via, for example, `local.conf`. The `EWAOL_*_ROOTFS_EXTRA_SPACE` variables apply their values to the relevant `IMAGE_ROOTFS_EXTRA_SPACE` BitBake variable.

Adding Extra EWAOL Guest VM Instances

It is possible to deploy multiple EWAOL Guest VM instances on the virtualization distribution image, each one based on the same kernel and image recipe. The number of Guest VM instances built for and included on the virtualization distribution image can be set via the `EWAOL_GUEST_VM_INSTANCES` variable.

Guest VM instances can be independently configured via BitBake variables which reference the Guest VM's integer instance index, from 1 to the value of `EWAOL_GUEST_VM_INSTANCES`, inclusive. For example, variables with a prefix `EWAOL_GUEST_VM1_` apply to the first Guest VM, variables with a prefix `EWAOL_GUEST_VM2_` apply to the second Guest VM, and so on. All Guest VM instances use the same default configuration, apart from the hostname, which is generated for each Guest VM by appending the instance index to the `EWAOL_GUEST_VM_HOSTNAME` BitBake variable. By default, the first Guest VM will have a hostname `ewaol-guest-vm1`, the second will have a hostname `ewaol-guest-vm2`, and so on. An example of configuring a second Guest VM instance using the kas tool is given in `meta-ewaol-config/kas/include/second-guest-vm-parameters.yml`, although these variables will only be used if `EWAOL_GUEST_VM_INSTANCES` is set to build two or more Guest VMs.

Configuring Guest VM PCI Device Passthrough

An EWAOL virtualization distribution image running on the AVA Developer Platform is capable of supporting Xen PCI passthrough, allowing Guest VMs to be assigned exclusive use of a single PCI device. This capability is not enabled by default, and requires the following Build Modifier Config:

- **Build Modifier Config**: `meta-ewaol-config/kas/xen_pci_passthrough.yml`.

  This Build Modifier Config appends `xen-pci-passthrough` to `MACHINE_FEATURES`.

Note: Xen PCI device passthrough is currently only supported on the AVA Developer Platform.

With the capability enabled, it is then possible to assign a single PCI device to a Guest VM by configuring an additional environment variable, provided for the corresponding Guest VM. This environment variable and its default value when the Build Modifier Config is provided to enable Xen PCI passthrough support is as follows:

```
EWAOL_GUEST_VM1_PCI_PASSTHROUGH_DEVICE: "0000:01:00.0"          # PCI device ID␣
↪to be assigned
```

As described in the previous section, this example environment variable customizes the first Guest VM only, but other Guest VMs may be configured similarly (if they have been defined).

By default, the Build Modifier Config assigns the first PCI ethernet network device (which has device ID `0000:01:00.0`) for exclusive use by the first Guest VM.

Warning: The PCI device IDs configured for PCI passthrough are not validated as part of the EWAOL build system, and it is therefore the responsibility of the user to ensure that the device IDs are valid, and that multiple Guest VMs have not been assigned exclusive use of the same PCI device.

Other EWAOL Features

Developer Support

- **Corresponding value in** `DISTRO_FEATURES` **variable**: `ewaol-devel`.

  This EWAOL distribution image feature:

  - Is default if not set with any other EWAOL-specific `DISTRO_FEATURES`.
  - Includes packages appropriate for development image builds, such as the `debug-tweaks` package, which sets an empty root password for simplified development access.

Run-Time Integration Tests

- **Corresponding value in** `DISTRO_FEATURES` **variable**: `ewaol-test`.
- **Build Modifier Config**: `meta-ewaol-config/kas/tests.yml`.

  This EWAOL distribution image feature:

  - Includes the EWAOL test suites provided to validate the image is running successfully with the expected EWAOL functionalities.

  The Build Modifier for this distribution image feature automatically includes the Yocto Package Test (ptest) framework in the image, configures the inclusion of `meta-ewaol-tests` as a Yocto layer source for the build, and appends the `ewaol-test` feature to `DISTRO_FEATURES` for the build.

  To include run-time integration tests on an EWAOL distribution image, provide the Build Modifier Config to the kas build command.

  For example, to include the tests on an EWAOL distribution image for the N1SDP hardware target platform, run the following commands depending on the target architecture:

  - Baremetal architecture for N1SDP:

    ```
    kas build meta-ewaol-config/kas/baremetal.yml:meta-ewaol-config/kas/tests.
    ↪yml:meta-ewaol-config/kas/n1sdp.yml
    ```

  - Virtualization architecture for N1SDP:

    ```
    kas build meta-ewaol-config/kas/virtualization.yml:meta-ewaol-config/kas/tests.
    ↪yml:meta-ewaol-config/kas/n1sdp.yml
    ```

  Each suite of run-time integration tests and specific customizable variables associated with each suite are detailed separately, at *Run-Time Integration Tests*.

Security Hardening

- **Corresponding value in** `DISTRO_FEATURES` **variable**: `ewaol-security`.
- **Build Modifier Config**: `meta-ewaol-config/kas/security.yml`.

  This EWAOL distribution image feature:

  - Configures user accounts, packages, remote access controls and other image features to provide extra security hardening for the EWAOL distribution image.

To include extra security hardening on an EWAOL distribution image, provide the Build Modifier Config to the kas build command, which appends the `ewaol-security` feature to `DISTRO_FEATURES` for the build.

For example, to include it on the EWAOL distribution image for the N1SDP hardware target platform, run the following commands depending on the target architecture:

- Baremetal architecture for N1SDP:

```
kas build meta-ewaol-config/kas/baremetal.yml:meta-ewaol-config/kas/security.
↪yml:meta-ewaol-config/kas/n1sdp.yml
```

- Virtualization architecture for N1SDP:

```
kas build meta-ewaol-config/kas/virtualization.yml:meta-ewaol-config/kas/
↪security.yml:meta-ewaol-config/kas/n1sdp.yml
```

The security hardening is described in more detail at *Security Hardening*.

Software Development Kit (SDK)

- **Corresponding value in `DISTRO_FEATURES` variable**: `ewaol-sdk`.
- **Build Modifier Config**: `meta-ewaol-config/kas/baremetal-sdk.yml` and `meta-ewaol-config/kas/virtualization-sdk.yml`, for the baremetal architecture and virtualization architecture, respectively.

  This EWAOL distribution image feature:

  - Adds the EWAOL Software Development Kit (SDK) which includes packages and image features to support on-target software development activities.
  - Enables two additional SDK build targets, `ewaol-baremetal-sdk-image` and `ewaol-virtualization-sdk-image`, each only compatible with the corresponding architecture's distribution image feature.

  The Build Modifier for this distribution image feature automatically appends `ewaol-sdk` to `DISTRO_FEATURES`, and sets the appropriate build target with the necessary configuration from the relevant Architecture Config included by default, meaning it is not necessary to explicitly supply an Architecture Config to the kas build tool if passing an SDK Build Modifier Config.

  To include the SDK on an EWAOL distribution image, provide the appropriate SDK Build Modifier Config to the kas build command.

  For example, to include the SDK on an EWAOL distribution image for the N1SDP hardware target platform, run the following commands depending on the target architecture:

  - Baremetal architecture for N1SDP:

```
kas build meta-ewaol-config/kas/baremetal-sdk.yml:meta-ewaol-config/kas/n1sdp.
↪yml
```

  - Virtualization architecture for N1SDP:

```
kas build meta-ewaol-config/kas/virtualization-sdk.yml:meta-ewaol-config/kas/
↪n1sdp.yml
```

  The SDK itself is described in more detail at *Software Development Kit (SDK)*.

Additional Distribution Image Customizations

An additional set of customization options are available for EWAOL distribution images, which don't fall under a distinct distribution image feature. These customizations are listed below, grouped by the customization target.

Filesystem Customization

Adding Extra Rootfs Space

The size of the root filesystem can be extended via the `EWAOL_ROOTFS_EXTRA_SPACE` BitBake variable, which defaults to `2000000` Kilobytes. The value of this variable is appended to the `IMAGE_ROOTFS_EXTRA_SPACE` BitBake variable.

For an EWAOL virtualization distribution image, additional variables are provided which apply to the different root filesystems as described in *Virtualization Architecture Customization*. By default, the Control VM's rootfs size is increased at build-time to support the sum of all Guest VM rootfs sizes. The customizable `EWAOL_CONTROL_VM_ROOTFS_EXTRA_SPACE` variable therefore corresponds just to the space allocated for the Control VM rootfs, in addition to the size necessary to support the Guest VM(s). Increasing `EWAOL_ROOTFS_EXTRA_SPACE` when building an EWAOL virtualization distribution image increases both the Guest VM(s) and Control VM rootfs size, which means that increasing this variable will result in the Control VM rootfs size expanding by more than the set value.

Filesystem Compilation Tuning

The EWAOL filesystem by default uses the generic `armv8a-crc` tune for `aarch64` based target platforms. This reduces build times by increasing the sstate-cache reused between different image types and target platforms. This optimization can be disabled by setting `EWAOL_GENERIC_ARM64_FILESYSTEM` to `"0"`. The tune used when `EWAOL_GENERIC_ARM64_FILESYSTEM` is enabled can be changed by setting `EWAOL_GENERIC_ARM64_DEFAULTTUNE`, which configures the `DEFAULTTUNE` BitBake variable for the `aarch64` based target platforms builds. See DEFAULTTUNE for more information.

In summary, the relevant variables and their default values are:

```
EWAOL_GENERIC_ARM64_FILESYSTEM: "1"              # Enable generic file system␣
↪(1 or 0).
EWAOL_GENERIC_ARM64_DEFAULTTUNE: "armv8a-crc"    # Value of DEFAULTTUNE if␣
↪generic file system enabled.
```

Their values can be set by passing them as environmental variables. For example, the optimization can be disabled using:

```
EWAOL_GENERIC_ARM64_FILESYSTEM="0" kas build meta-ewaol-config/kas/baremetal.
↪yml:meta-ewaol-config/kas/n1sdp.yml
```

Manual BitBake Build Setup

In order to build an EWAOL distribution image without the kas build tool directly via BitBake, it is necessary to prepare a BitBake project as follows:

- Configure *dependent Yocto layers* in `bblayers.conf`.
- Configure the `DISTRO` as `ewaol` in `local.conf`.
- Configure the image `DISTRO_FEATURES`, including the EWAOL Architecture (`ewaol-baremetal` or `ewaol-virtualization`), in `local.conf`.

Assuming correct environment configuration, the BitBake build can then be run for the desired image target corresponding to one of the following:

- `ewaol-baremetal-image`
- `ewaol-baremetal-sdk-image`
- `ewaol-virtualization-image`
- `ewaol-virtualization-sdk-image`

As the kas build configuration files within the `meta-ewaol-config/kas/` directory define the recommended build settings for each feature. Any additional functionalities may therefore be enabled by reading these configuration files and manually inserting their changes into the BitBake build environment.

1.3.4 Yocto Layers

The `meta-ewaol` repository provides three layers compatible with the Yocto Project, in the following sub-directories:

- `meta-ewaol-distro`

  Yocto distribution layer providing top-level and general policies for the EWAOL distribution images.

- `meta-ewaol-tests`

  Yocto software layer with recipes that include run-time tests to validate EWAOL functionalities.

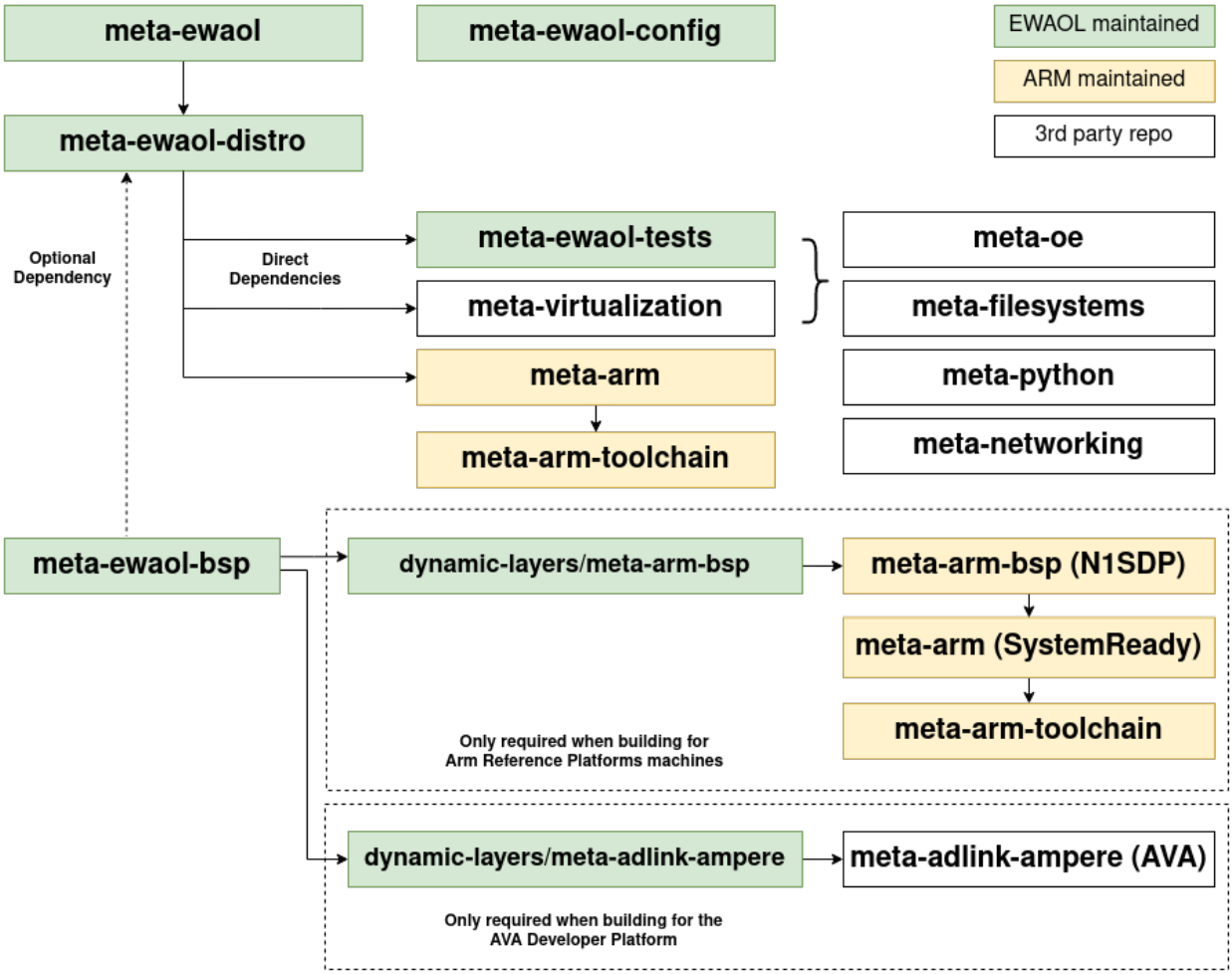
- `meta-ewaol-bsp`

  Yocto BSP layer with target platform specific extensions for particular EWAOL distribution images.

  - For the N1SDP hardware target platform, this layer currently extends the `n1sdp` machine definition from the `meta-arm-bsp` layer for EWAOL virtualization distribution images.. The `meta-ewaol-bsp` layer contains an additional grub configuration file with Xen boot entry and a custom kickstart `ewaol-virtualization-n1sdp-efidisk.wks.in` file. There is also a `xen-devicetree.bb` recipe, to generate a devicetree with extra modules nodes required by Xen to start the Control VM (Dom0). In addition, the Xen devicetree together with a Xen efi binary are included into the final wic image in the `boot` partition.
  - For the AVA hardware target platform, this layer provides additional `systemd` configuration for ethernet interfaces based on the `i40e` as used on the AVA Developer Platform.

Layer Dependency Overview

The following diagram illustrates the layers which are integrated by the EWAOL project, which are further expanded on below. The layer revisions are related to the EWAOL `kirkstone-dev` branch.



EWAOL depends on the following layer dependency sources:

```
URL: https://git.yoctoproject.org/git/poky
layers: meta, meta-poky
branch: kirkstone
revision: HEAD

URL: https://git.openembedded.org/meta-openembedded
```

(continues on next page)

(continued from previous page)

```
layers: meta-filesystems, meta-networking, meta-oe, meta-python
branch: kirkstone
revision: HEAD

URL: https://git.yoctoproject.org/git/meta-virtualization
layers: meta-virtualization
branch: kirkstone
revision: HEAD
```

Additional layer dependency sources may be conditionally required, depending on the specific EWAOL distribution image being built.

The first additional layer dependency source is the `meta-arm` repository, which provides three Yocto layers:

```
URL: https://git.yoctoproject.org/git/meta-arm
layers: meta-arm, meta-arm-bsp, meta-arm-toolchain
branch: kirkstone
revision: HEAD
```

The layers required from `meta-arm` depend on the EWAOL distribution image:

- EWAOL SDK distribution images require `meta-arm` and `meta-arm-toolchain`, as the `gator-daemon` package is installed on the rootfs.
- An EWAOL virtualization distribution image requires `meta-arm` and `meta-arm-toolchain`, as by default a bundled Guest VM image based on the `generic-arm64` `MACHINE` is built.
- An EWAOL distribution image built for the N1SDP hardware target platform requires `meta-arm`, `meta-arm-bsp`, and `meta-arm-toolchain`.

These layers are described as follows:

- `meta-arm`:
  - URL: https://git.yoctoproject.org/cgit/cgit.cgi/meta-arm/tree/meta-arm.
  - Clean separation between Firmware and OS.
  - The canonical source for SystemReady firmware.
- `meta-arm-bsp`:
  - URL: https://git.yoctoproject.org/cgit/cgit.cgi/meta-arm/tree/meta-arm-bsp.
  - Board specific components for Arm target platforms.
- `meta-arm-toolchain`:
  - URL: https://git.yoctoproject.org/meta-arm/tree/meta-arm-toolchain.
  - Provides toolchain for Arm target platforms

The second additional layer dependency source is the `meta-adlink-ampere` repository, which provides a single Yocto layer:

```
URL: https://github.com/ADLINK/meta-adlink-ampere.git
layers: meta-adlink-ampere
branch: kirkstone
revision: HEAD
```

This Yocto layer provides BSP components required when building an EWAOL distribution image for the AVA hardware target platform.

1.3.5 Security Hardening

EWAOL distribution images can be hardened to reduce potential sources or attack vectors of security vulnerabilities. EWAOL security hardening modifies the distribution to:

- Force password update for each user account after first logging in. An empty and expired password is set for each user account by default.
- Enhance the kernel security, kernel configuration is extended with the `security.scc` in `KERNEL_FEATURES`.
- Enable the 'Secure Computing Mode' (seccomp) Linux kernel feature by appending `seccomp` to `DISTRO_FEATURES`.
- Ensure that all available packages from `meta-openembedded`, `meta-virtualization` and `poky` layers are configured with: `--with-libcap[-ng]`.
- Remove `debug-tweaks` from `IMAGE_FEATURES`.
- Disable all login access to the `root` account.
- Sets the umask to `0027` (which translates permissions as `640` for files and `750` for directories).

Security hardening is not enabled by default, see *Security Hardening* for details on including the security hardening on the EWAOL distribution image.

EWAOL security hardening does not reduce the scope of the *Run-Time Integration Tests*.

1.3.6 Software Development Kit (SDK)

EWAOL SDK distribution images enable users to perform common development tasks on the target, such as:

- Application and kernel module compilation
- Remote debugging
- Profiling
- Tracing
- Runtime package management

The precise list of packages and image features provided as part of the EWAOL SDK can be found in `meta-ewaol-distro/conf/distro/include/ewaol-sdk.inc`.

The Yocto project provides guidance for some of these common development tasks, for example kernel module compilation, profiling and tracing, and runtime package management.

See *Software Development Kit (SDK)* for details on including the SDK on an EWAOL distribution image.

If the SDK is included on an EWAOL virtualization distribution image, the SDK will be available on both the Control VM and any EWAOL Guest VMs built during the image build process.

1.3.7 Validation

Build-Time Kernel Configuration Check

After the kernel configuration has been produced during the build, it is checked to validate the presence of necessary kernel configuration to comply with specific EWAOL functionalities.

A list of required kernel configs is used as a reference, and compared against the list of available configs in the kernel build. All reference configs need to be present either as module (`=m`) or built-in (`=y`). A BitBake warning message is produced if the kernel is not configured as expected.

The following kernel configuration checks are performed:

- **Container engine support**:

  Check performed via: `meta-ewaol-distro/classes/containers_kernelcfg_check.bbclass`. By default Yocto Docker config is used as the reference.

- **K3s orchestration support**:

  Check performed via: `meta-ewaol-distro/classes/k3s_kernelcfg_check.bbclass`. By default Yocto K3s config is used as the reference.

- **Xen virtualization support** (available for EWAOL virtualization distribution images):

  Check performed via: `meta-ewaol-distro/classes/xen_kernelcfg_check.bbclass`. By default Yocto Xen config is used as the reference.

Run-Time Integration Tests

The `meta-ewaol-tests` Yocto layer contains recipes and configuration for including run-time integration tests into an EWAOL distribution, to be run manually after booting the image.

The EWAOL run-time integration tests are a mechanism for validating EWAOL core functionalities. The integration test suites included on an EWAOL distribution image depend on its target architecture, as follows:

- **Baremetal architecture:**
  - *Container Engine Tests*
  - *K3s Orchestration Tests* (local deployment of a K3s pod)
  - *User Accounts Tests*
- **Virtualization architecture:**
  - Control VM:
    - *Container Engine Tests*
    - *K3s Orchestration Tests* (remote deployment of K3s pods on the Guest VMs, from the Control VM)
    - *User Accounts Tests*
    - *Xen Virtualization Tests*
  - Guest VM:
    - *Container Engine Tests*
    - *User Accounts Tests*

The tests are built as a Yocto Package Test (ptest), and implemented using the Bash Automated Test System (BATS).

Run-time integration tests are not included on an EWAOL distribution image by default, and must instead be included explicitly. See *Run-Time Integration Tests* within the Build System documentation for details on how to include the tests.

The test suites are executed using the `test` user account, which has `sudo` privileges. More information about user accounts can be found at *User Accounts*.

Running the Tests

If the tests have been included on the EWAOL distribution image, they may be run via the ptest framework, using the following command after booting the image and logging in:

```
ptest-runner [test-suite-id]
```

If the test suite identifier (`[test-suite-id]`) is omitted, all integration tests will be run. For example, running `ptest-runner` on the Control VM of an EWAOL virtualization distribution image produces output such as the following:

```
$ ptest-runner
START: ptest-runner
[...]
PASS:container-engine-integration-tests
[...]
PASS:k3s-integration-tests
[...]
PASS:user-accounts-integration-tests
[...]
PASS:virtualization-integration-tests
[...]
STOP: ptest-runner
```

Note: As different EWAOL architectures support different test suites. `ptest-runner -l` is a useful command to list the available test suites on the image.

Alternatively, a single standalone test suite may be run via a runner script included in the test suite directory:

```
/usr/share/[test-suite-id]/run-[test-suite-id]
```

Upon completion of the test-suite, a result indicator will be output by the script, as one of two options: `PASS:[test-suite-id]` or `FAIL:[test-suite-id]`, as well as an appropriate exit status.

A test suite consists of one or more 'top-level' BATS tests, which may be composed of multiple assertions, where each assertion is considered a named sub-test. If a sub-test fails, its individual result will be included in the output with a similar format. In addition, if a test failed then debugging information will be provided in the output of type `DEBUG`. The format of these results are described in *Test Logging*.

Test Logging

Test suite execution outputs results and debugging information into a log file. As the test suites are executed using the `test` user account, this log file will be owned by the `test` user and located in the `test` user's home directory by default, at:

```
/home/test/runtime-integration-tests-logs/[test-suite-id].log
```

Therefore, reading this file as another user will require `sudo` access. The location of the log file for each test suite is customizable, as described in the detailed documentation for each test suite below. The log file is replaced on each new execution of a test suite.

The log file will record the results of each top-level integration test, as well as a result for each individual sub-test up until a failing sub-test is encountered.

Each top-level result is formatted as:

```
TIMESTAMP RESULT:[top_level_test_name]
```

Each sub-test result is formatted as:

```
TIMESTAMP RESULT:[top_level_test_name]:[sub_test_name]
```

Where `TIMESTAMP` is of the format `%Y-%m-%d %H:%M:%S` (see Python Datetime Format Codes), and `RESULT` is either `PASS`, `FAIL`, or `SKIP`.

On a test failure, a debugging message of type `DEBUG` will be written to the log. The format of a debugging message is:

```
TIMESTAMP DEBUG:[top_level_test_name]:[return_code]:[stdout]:[stderr]
```

Additional informational messages may appear in the log file with `INFO` or `DEBUG` message types, e.g. to log that an environment clean-up action occurred.

Test Suites

The test suites are detailed below.

Container Engine Tests

The container engine test suite is identified as:

```
container-engine-integration-tests
```

for execution via `ptest-runner` or as a standalone BATS suite, as described in *Running the Tests*.

On an EWAOL virtualization distribution image, the container engine test suite is available for execution on both the Control VM and the Guest VM. In addition, as part of running the test suite on the Control VM, an extra test will be performed which logs into each Guest VM and runs the container engine test suite on it, thereby reporting any test failures of a Guest VM as part of the Control VM's test suite execution.

The test suite is built and installed in the image according to the following BitBake recipe: `meta-ewaol-tests/recipes-tests/runtime-integration-tests/container-engine-integration-tests.bb`.

Currently the test suite contains three top-level integration tests, which run consecutively in the following order.

1. `run container` is composed of four sub-tests:
    1.1. Run a containerized detached workload via the `docker run` command
        - Pull an image from the network

- Create and start a container

1.2. Check the container is running via the `docker inspect` command

1.3. Remove the running container via the `docker remove` command

- Stop the container

- Remove the container from the container list

1.4. Check the container is not found via the `docker inspect` command

2. `container network connectivity` is composed of a single sub-test:

2.1. Run a containerized, immediate (non-detached) network-based workload via the `docker run` command

- Create and start a container, re-using the existing image

- Update package lists within container from external network

3. `run container engine integration tests on Guest VMs from the Control VM` is only executed on the Control VM. On a Guest VM this test is skipped. The test is composed of the following sub-tests:

3.1. Check that Xendomains is initialized `systemctl status`

For each Guest VM:

3.2. Check the Guest VM is running via `xendomains status`

3.3. Run the container engine integration tests on the Guest VM

- Uses an Expect script to log-in and execute the `ptest-runner container-engine-integration-tests` command

- This command will therefore run only the first and second top-level integration tests of the container engine integration test suite on the Guest VM

The tests can be customized via environment variables passed to the execution, each prefixed by `CE_` to identify the variable as associated to the container engine tests:

`CE_TEST_IMAGE`: defines the container image

Default: `docker.io/library/alpine`

`CE_TEST_LOG_DIR`: defines the location of the log file

Default: `/home/test/runtime-integration-tests-logs/`

Directory will be created if it does not exist

See *Test Logging*

`CE_TEST_CLEAN_ENV`: enable test environment clean-up

Default: `1` (enabled)

See *Container Engine Environment Clean-Up*

`CE_TEST_GUEST_VM_BASENAME`: defines the base Xen domain name (hostname) to determine which Guest VMs to include as part of the test suite execution

Only available when running the tests on an EWAOL virtualization distribution image

Any Guest VMs that have a Xen domain name starting with this value will be tested as part of executing the suite on the Control VM

Default: `${EWAOL_GUEST_VM_HOSTNAME}`

With standard configuration, the default Guest VMs will therefore be all Guest VMs with domain names which start with `ewaol-guest-vm`

Container Engine Environment Clean-Up

A clean environment is expected when running the container engine tests. For example, if the target image already exists within the container engine environment, then the functionality to pull the image over the network will not be validated. Or, if there are running containers from previous (failed) tests then they may interfere with subsequent test executions.

Therefore, if `CE_TEST_CLEAN_ENV` is set to `1` (as is default), running the test suite will perform an environment clean before and after the suite execution.

The environment clean operation involves:

- Determination and removal of all running containers of the image given by `CE_TEST_IMAGE`
- Removal of the image given by `CE_TEST_IMAGE`, if it exists
- Clearing the passwords set when the test suite accessed each Guest VM, performed only when running the test suite on a virtualization distribution image with *Security Hardening* enabled.

If enabled then the environment clean operations will always be run, regardless of test-suite success or failure.

K3s Orchestration Tests

The K3s test suite is identified as:

```
k3s-integration-tests
```

for execution via `ptest-runner` or as a standalone BATS suite, as described in *Running the Tests*.

The test suite is built and installed in the image according to the following BitBake recipe within `meta-ewaol-tests/recipes-tests/runtime-integration-tests/k3s-integration-tests.bb`.

Currently the test suite contains a single top-level integration test which validates the deployment and high-availability of a test workload based on the Nginx webserver. The test suite is dependent on the target EWAOL architecture, as follows.

For EWAOL baremetal distribution images, the K3s integration tests consider a single-node cluster, which runs a K3s server together with its built-in worker agent. The containerized test workload is therefore deployed to this node for scheduling and execution.

For EWAOL virtualization distribution images, the K3s integration tests consider a cluster comprised of multiple nodes: the Control VM running a K3s server, and each target Guest VM running a K3s agent which is connected to the server. The containerized test workload is configured to only be schedulable on the Guest VMs, meaning that the server on the Control VM orchestrates a test application with replicas that are deployed and executed on each of the target Guest VMs. In addition to the same initialization procedure that is carried out when running the tests on a baremetal distribution image, initialization for virtualization distribution images includes connecting a K3s agent running on each target Guest VM to the Control VM's K3s server (if it is not already connected). To do this, before the tests run, the Systemd service that provides the K3s agent on each Guest VM is configured with a Systemd service unit override that provides the IP and authentication token of the Control VM's K3s server, and this service is then started. The K3s integration test suite therefore expects that there are target Guest VMs available when running on a virtualization distribution image, and will not create any if they are not found.

In both cases, the test suite will not be run until the appropriate K3s services are in the 'active' state, and all 'kube-system' pods are either running, or have completed their workload.

1. `K3s container orchestration` is composed of many sub-tests, grouped here by test area:

   Workload Deployment:

   1.1. Deploy test Nginx workload from YAML file via `kubectl apply`

1.2. Ensure Pods are initialized via `kubectl wait`
1.3. Create NodePort Service to expose Deployment via `kubectl create service`
1.4. Get the IP of the node(s) running the Deployment via `kubectl get`
1.5. Ensure web service is accessible on the node(s) via `wget`
Deployment Upgrade:
1.6. Check initial image version of running Deployment via `kubectl get`
1.7. Get all pre-upgrade Pod names running Deployment via `kubectl get`
1.8. Upgrade image version of Deployment via `kubectl set`
1.9. Ensure a new set of Pod names have been started via `kubectl wait` and `kubectl get`
1.10. Check Pods are running the upgraded image version via `kubectl get`
1.11. Ensure web service is still accessible on the node(s) via `wget`
Server Failure Tolerance:
1.12. Stop K3s server Systemd service via `systemctl stop`
1.13. Ensure web service is still accessible on the node(s) via `wget`
1.14. Restart the Systemd service via `systemctl start`
1.15. Check K3S server is again responding to `kubectl get`

The tests can be customized via environment variables passed to the execution, each prefixed by `K3S_` to identify the variable as associated to the K3s orchestration tests:

`K3S_TEST_LOG_DIR`: defines the location of the log file
Default: `/home/test/runtime-integration-tests-logs/`
Directory will be created if it does not exist
See *Test Logging*

`K3S_TEST_CLEAN_ENV`: enable test environment clean-up
Default: `1` (enabled)
See *K3s Environment Clean-Up*

`K3S_TEST_GUEST_VM_BASENAME`: defines the base Xen domain name (hostname) to determine which Guest VMs to include in the test suite execution
Only available when running the tests on an EWAOL virtualization distribution image
Any Guest VMs that have a Xen domain name starting with this value will be connected to the K3s server, to form a cluster
Default: `${EWAOL_GUEST_VM_HOSTNAME}`
With standard configuration, the default Guest VMs are therefore all Guest VMs which have a domain name starting with `ewaol-guest-vm`

K3s Environment Clean-Up

A clean environment is expected when running the K3s integration tests, to ensure that the system is ready to be validated. For example, the test suite expects that the Pods created from any previous execution of the integration tests have been deleted, in order to test that a new Deployment successfully initializes new Pods for orchestration.

Therefore, if `K3S_TEST_CLEAN_ENV` is set to `1` (as is default), running the test suite will perform an environment clean before and after the suite execution.

The environment clean operation involves:

- Deleting any previous K3s test Service

- Deleting any previous K3s test Deployment, ensuring corresponding Pods are also deleted

For virtualization distribution images, additional clean-up operations are performed:

- Deleting each Guest VM node from the K3s cluster
- Stopping the K3s agent running on each Guest VM, and deleting any test Systemd service override on each Guest VM
- Clearing the passwords set when the test suite accessed each Guest VM, performed only when running the test suite on a virtualization distribution image with *Security Hardening* enabled.

If enabled then the environment clean operations will always be run, regardless of test-suite success or failure.

User Accounts Tests

The User Accounts test suite is identified as:

```
user-accounts-integration-tests
```

for execution via `ptest-runner` or as a standalone BATS suite, as described in *Running the Tests*.

The test suite is built and installed in the image according to the following BitBake recipe within `meta-ewaol-tests/recipes-tests/runtime-integration-tests/user-accounts-integration-tests.bb`.

The test suite validates that the user accounts described in *User Accounts* are correctly configured on the EWAOL distribution image. Therefore, the validation performed by the test suite is dependent on the target architecture, and on whether or not it has been configured with *EWAOL Security Hardening*, as follows.

For a baremetal image, the test suite validates that the expected user accounts are configured and appropriate access permissions are in place. For a virtualization image, the test suite is available on both the Control VM and the Guest VM(s), and includes the same validation as the baremetal test suite on the respective VM's local user accounts. However, as part of running the test suite on the Control VM, an extra test will be performed which logs into the Guest VMs and runs the user accounts test suite on them, thereby reporting any test failures of a Guest VM as part of the Control VM's test suite execution.

As the configuration of user accounts is modified for EWAOL distribution images that are built with EWAOL security hardening, additional security-related validation is included in the test suite for these images, both on EWAOL baremetal and virtualization distribution images. These additional tests validate that the appropriate password requirements and root-user access restrictions are correctly imposed, and that the mask configuration for permission control of newly created files and directories is applied correctly.

The test suite therefore contains three top-level integration tests, two of which are conditionally executed, as follows:

1. `user accounts management tests` is composed of three sub-tests:
    1.1. Check home directory permissions are correct for the default non-privileged EWAOL user account, via the filesystem `stat` utility
    1.2. Check the default privileged EWAOL user account has `sudo` command access
    1.3. Check the default non-privileged EWAOL user account does not have `sudo` command access
2. `user accounts management additional security tests` is only included for images configured with EWAOL security hardening, and is composed of four sub-tests:
    2.1. Log-in to a local console using the non-privileged EWAOL user account
        - As part of the log-in procedure, validate the user is prompted to set an account password
    2.2. Check that log-in to a local console using the root account fails
    2.3. Check that SSH log-in to localhost using the root account fails
    2.4. Check that the umask value is set correctly

3. `run user accounts integration tests on the Guest VM from the Control VM` is only included for EWAOL virtualization distribution images, and is only executed on the Control VM. On a Guest VM this test is skipped. The test is composed of the following sub-tests:

    3.1. Check that Xendomains is initialized `systemctl status`

    For each Guest VM:

        3.2. Check the Guest VM is running via `xendomains status`

        3.3. Run the user accounts integration tests on the Guest VM

            - Uses an Expect script to log-in and execute the `ptest-runner user-accounts-integration-tests` command

            - This command will therefore run only the first and second (if EWAOL security hardening is configured) top-level integration tests of the user accounts integration test suite on the Guest VM

The tests can be customized via environment variables passed to the execution, each prefixed by `UA_` to identify the variable as associated to the user accounts tests:

`UA_TEST_LOG_DIR`: defines the location of the log file

Default: `/home/test/runtime-integration-tests-logs/`

Directory will be created if it does not exist

See *Test Logging*

`UA_TEST_CLEAN_ENV`: enable test environment clean-up

Default: `1` (enabled)

See *User Accounts Environment Clean-Up*

`UA_TEST_GUEST_VM_BASENAME`: defines the base Xen domain name (hostname) to determine which Guest VMs to include as part of the test suite execution

Only available when running the tests on an EWAOL virtualization distribution image

Any Guest VMs that have a Xen domain name starting with this value will be tested as part of executing the suite on the Control VM

Default: `${EWAOL_GUEST_VM_HOSTNAME}`

With standard configuration, the default Guest VMs will therefore be all Guest VMs with domain names which start with `ewaol-guest-vm`

User Accounts Environment Clean-Up

As the user accounts integration tests only modify the system for images built with EWAOL security hardening, clean-up operations are only performed when running the test suite on these images.

In addition, the clean-up operations will only occur if `UA_TEST_CLEAN_ENV` is set to `1` (as is default).

The environment clean-up operations for images built with EWAOL security hardening are:

- Reset the password for the `test` user account
- Reset the password for the non-privileged EWAOL user account
- Clearing the passwords set when the test suite accessed each Guest VM, performed only when running the test suite on a virtualization distribution image with *Security Hardening* enabled.

After the environment clean-up, the user accounts will return to their original state where the first log-in will prompt the user for a new account password.

If enabled then the environment clean operations will always be run, regardless of test-suite success or failure.

Xen Virtualization Tests

The Xen Virtualization test suite is identified as:

```
virtualization-integration-tests
```

for execution via `ptest-runner` or as a standalone BATS suite, as described in *Running the Tests*.

The test suite is built and installed in the image according to the following BitBake recipe within `meta-ewaol-tests/recipes-tests/runtime-integration-tests/virtualization-integration-tests.bb`.

The test suite is only available for images that target the virtualization architecture.

Currently the test suite contains two top-level integration tests, which validate that the target Guest VMs are correctly running, and validate that they can be managed successfully from the Control VM. These tests are as follows:

For each Guest VM:

1. `validate Guest VM is running` is composed of two sub-tests:
    1.1. Check that Xen reports the Guest VM as running via `xendomains status`
    1.2. Check that the Guest VM is operational and has external network access
        - Log-in to the Guest VM and access its interactive shell via `xl console`
        - Ping an external IP with the `ping` utility
2. `validate Guest VM management` is composed of five sub-tests:
    2.1. Check that Xen reports the Guest VM as running via `xendomains status`
    2.2. Shutdown the Guest VM via `systemctl stop`
    2.3. Check that Xen reports the Guest VM as not running via `xendomains status`
    2.4. Start the Guest VM via `systemctl start`
    2.5. Check that Xen reports the Guest VM as running via `xendomains status`

The tests can be customized via environment variables passed to the execution, each prefixed by `VIRT_` to identify the variable as associated to the virtualization integration tests:

`VIRT_TEST_LOG_DIR`: defines the location of the log file

Default: `/home/test/runtime-integration-tests-logs/`

Directory will be created if it does not exist

See *Test Logging*

`VIRT_TEST_CLEAN_ENV`: enable test environment clean-up

Default: `1` (enabled)

See *Xen Virtualization Environment Clean-Up*

`VIRT_TEST_GUEST_VM_BASENAME`: defines the base Xen domain name (hostname) to determine which Guest VMs to validate

Default: `${EWAOL_GUEST_VM_HOSTNAME}`

With standard configuration, the default Guest VMs will therefore be all Guest VMs with domain names which start with `ewaol-guest-vm`

Prior to execution, the Xen Virtualization test suite expects the `xendomains.service` Systemd service on the Control VM to be running or in the process of initializing.

Xen Virtualization Environment Clean-Up

The Xen Virtualization integration tests only modify the system environment when the test suite is executed on an image with *Security Hardening* enabled, as accessing a Guest VM on a security hardened image requires setting the user account password.

There is therefore only a single environment clean operation performed for this test suite:

- Clearing the passwords set when the tests accessed each Guest VM, performed only when running the test suite with *Security Hardening* enabled.

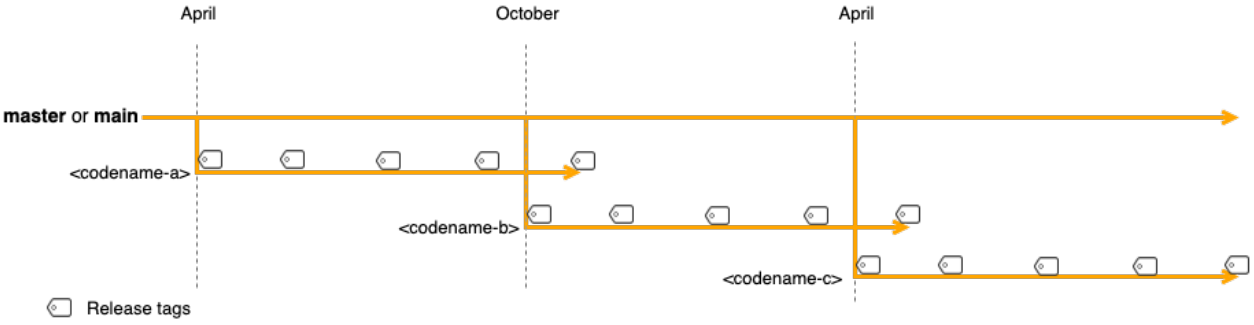
Cleaning up the account password will only occur if `VIRT_TEST_CLEAN_ENV` is set to 1 (as is default), in which case the environment clean will run before and after the suite execution.

If enabled then the environment clean operation will always be run, regardless of test-suite success or failure.

1.4 Codeline Management

The EWAOL project is released and developed based on Yocto's release branch process. This strategy allows us to make Major, Minor and Point/Patch Releases based on upstream stable branches, reducing the risk of having build and runtime issues.

1.4.1 Yocto Release Process Overview

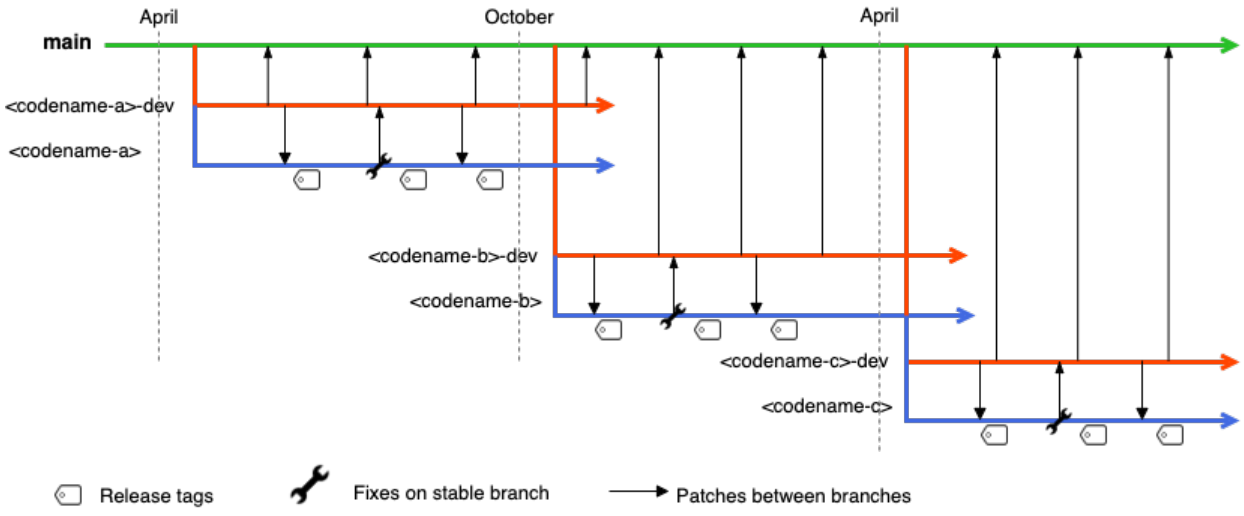


The diagram above gives an overview of the Yocto branch and release process:

- Development happens primarily in the master (or main) branch.
- The project has a major release roughly every 6 months where a stable release branch is created.
- Each major release has a *codename* which is also used to name the stable release branch (e.g. kirkstone).
- Once a stable branch is created and released, it only receives bug fixes with minor (point) releases on an unscheduled basis.
- The goal is for users and 3rd parties layers to use these codenamed branches as a means to be compatible with each other.

For a complete description of the Yocto release process, support schedule and other details, see the Yocto Release Process documentation.

1.4.2 EWAOL Branch and Release Process



EWAOL's branch and release process is based on the Yocto release process. The following sub-sections describe in more details the branch strategy for EWAOL's development and release process.

EWAOL main branch

- Represented by the green line on the diagram above.
- The repository's `main` branch is meant to be compatible with `master` or `main` branches from Poky and 3rd party layers.
- `meta-ewaol` is not actively developed on this `main` branch to avoid the instability inherited from Yocto development on the master branch.
- To reduce the effort required to move EWAOL to a new version of Yocto, this main branch is periodically updated with patches from the *EWAOL development branches* on a regular basis.

EWAOL development branches

- Represented by the red line on the diagram above.
- EWAOL uses development branches based/compatible with Yocto stable branches.
- A development branch in EWAOL is setup for each new Yocto release using the name convention `<codename>-dev` where *<codename>* comes from target Yocto release.
- The development branches in EWAOL are where fixes, improvements and new features are developed.
- On a regular basis, code from the development branch is ported over to the `main` branch to reduce the effort required to move EWAOL to a new version of Yocto.

EWAOL release branches

- Represented by the blue line on the diagram above.
- A new release branch in EWAOL is setup for each new Yocto release using the Yocto *codename* the branch targets.
- Hot fixes in the release branch are back ported to the development branch.
- Release branches are currently maintained not much longer than a Yocto release period (~7 months).

EWAOL release tags

- EWAOL is tagged using the version format `v<Major>.<Minor>.<Patch>`.
- Tags are always applied to commits from the release branch.
- The first release in a release branch is a *Major* release.
- Following releases in a release branch advance the *Minor* version number.
- *Patch* releases are mainly used for hot fixes which are then back ported to the development branch.
- Both *Major* and *Minor* releases may receive fixes, improvements and new features while *Patch* releases only receive fixes. Poky and 3rd party layers release/stable branches might be updated and pinned.

1.5 Contributing

EWAOL welcomes contributions via the `meta-ewaol` public Gitlab repository: https://gitlab.com/soafee/ewaol/meta-ewaol.

Contributions should follow the project's contribution guidelines, below.

1.5.1 Contribution Guidelines

The following is a set of guidelines that must be adhered to for contributions to be reviewed and accepted by the EWAOL project:

- **The contribution should be aligned with the goals and scope of the project.**

  EWAOL forms the reference software implementation of the SOAFEE project (https://soafee.io). Contributions should therefore be generally applicable to other downstream consumers of `meta-ewaol`.

- **Contributions should be high-quality, and should pass all tests within the repository's Quality Assurance (QA) check suite.**

  A contribution must adhere to a set of minimal standards defined by the project. These are grouped as follows, listed here as a link to more detail about the requirements:

  - *Commit Message*
  - *Documentation*
  - *Inclusive Language*
  - *License and Copyright Header*
  - *Python Code Quality*
  - *Shell Script Code Quality*
  - *Spelling*
  - *YAML Formatting*
  - *Yocto Layer Compatibility*

  A set of QA checks are provided by the project to help automatically validate that a contribution meets these minimal standards. However, each of the links above may include additional expectations that are not appropriate for automatic validation via an associated QA check, but should still be adhered to. Note that all contributions will undergo a code-review process.

  See *Quality Assurance Checks* for details on how to run the QA checks.

- **The contribution must be appropriately licensed.**

  It is expected that all contributions are licensed under the project's standard license (see *License* for details), except for files that represent modifications to externally licensed works (for example, patch files), which may be contributed under alternative licenses in order to be compliant with the licensing requirements of the associated external works.

- **Contributions must include appropriate documentation.**

  Contributions which change the documentation should be validated by running the *Documentation Build Validation* step.

- **The contribution should not introduce software regressions.**

  Contributions which introduce image build failures or integration test failures will not be accepted.

- **Contributions that introduce additional run-time functionality to EWAOL distribution images should be accompanied by run-time integration tests to validate the functionality.**

  Any additional run-time integration tests or test suites must be documented, and follow a similar design to the validation tests described in *Validation*.

- **Security aspects of contributions must be considered as part of EWAOL's Secure Development Lifecycle (SDL) process.**

EWAOL's SDL process is currently handled on a per-contribution basis, and it is expected that any security aspects raised by the project's maintainer(s) will be engaged with before the contribution can be accepted.

Example security aspects that must be considered as part of a contribution include:

- The contribution's effect on the management and storage of data onto temporary or persistent filesystems, and whether appropriate access controls to stored data (e.g. filesystem permissions) have been put in place.
- The contribution's effect on data communications (both transmitted or received) taking place on the EWAOL distribution, such as changes or additions involving client and server processes, and whether appropriate security mechanisms (e.g. secure protocols, data encryption) have been put in place.

1.5.2 Minimal Contribution Standards

Each contribution must meet a set of minimum requirements, listed below. A subset of these requirements are checked automatically by the QA check tooling, which is described in *Quality Assurance Checks*.

Note: More detail on the validation steps performed by each check are included as in-source documentation at the top of each check Python module within the `tools/qa-checks` directory. In addition, any failed validation will output the specific reason for the failure, enabling it to be fixed prior to submitting the contribution.

Commit Message

Each commit message of the contribution should adhere to the following requirements:

- The message title (first line) must not be blank.
- The title should include the associated layer or component as a prefix. For example, a commit that updates the documentation should be prefixed with `doc:`, or a commit that updates the run-time integration tests in the `meta-ewaol-tests` layer should be prefixed with `ewaol-tests:` (note that `meta-` should be dropped for brevity). More precise prefixes should be provided if the commit applies only to a particular component. For example, a commit that adds a Linux kernel patch to the `meta-ewaol-distro` layer should use a prefix `ewaol-distro/linux:`. Commits which effect multiple layers or components can include multiple prefixes separated via a comma.
- The first letter in the title after the prefix should be capitalized, if appropriate.
- The number of characters in the title must not be greater than 80.
- The second line must be blank to separate the message title and body.
- The number of characters in each line of the message body must not be greater than 80, unless this is unavoidable (for example, a URL).
- A sign-off must be be included in the message, with the following format: `Signed-off-by: Name <valid@email.dom>`. Note that the given email must also be formed correctly.

Please refer to the Git commit log of the repository for further examples of the expected format.

Documentation

The documentation should build successfully without errors or warnings. Validation of the documentation build is performed as part of the QA-check suite by default.

The rendered documentation should be checked to ensure the formatting is as expected, and no new formatting problems have been introduced. See *Documentation Build Validation* tool for details on building the rendered documentation.

Inclusive Language

The project aims to promote usage of inclusive language wherever possible, and so it is expected that contributions avoid introducing non-inclusive language into the repository.

To help identify potential usage of non-inclusive language, a check is performed as part of the QA check suite to highlight occurrences within the repository, so that alternatives may be considered. To support this automated check, a list of potentially non-inclusive terminology is maintained in the project configuration files at `meta-ewaol-config/qa-checks/non-inclusive-language.txt`.

As there are situations in which the terminology highlighted by the check is not used in a non-inclusive manner, or its usage is otherwise unavoidable, a usage can be tagged as excluded from the check by adding the text `inclusivity-exception` prior to the occurrence (either in the previous line or in the same line). So that this tag is not considered as part of the file's normal contents, it should be commented or otherwise excluded from the file's expected usage.

License and Copyright Header

Contributed files must contain a valid license and copyright header, following one of the two following formats, based on the source of the contribution:

1. Original works contributed to the project:

```
Copyright (c) YYYY(-YYYY), <Contributor>
SPDX-License-Identifier: <License name>
```

2. Modified externally-licensed works contributed to the project:

```
Based on: <original file>
In open-source project: <source project/repository>

Original file: Copyright (c) YYYY(-YYYY) <Contributor>
Modifications: Copyright (c) YYYY(-YYYY) <Contributor>

SPDX-License-Identifier: <License name>
```

Note: Please follow the contribution guideline relating to licensing in order to select the appropriate SPDX License Identifier for the contributed files.

The license and copyright header QA check expects the header lines to be commented. The current implementation therefore expects each line to begin with one of the following set of characters: `#`, `//`, `*`, `;`. Please refer to the current files within the repository for further guidance on how to include valid headers for different file types.

For each file with such a header, the final copyright year of the modifications must match or be later than the latest year that the file was modified in the git commit tree.

As some files within the project are inappropriate to license with a plain-text header (for example, `.png` image files), some file types are excluded as part of the QA check configuration. Running the QA check will highlight any files which are expected to include a valid header, but do not.

Python Code Quality

All Python code contributed to the project must pass validation by the Python style guide checker `pycodestyle`, which enforces style conventions based on the PEP 8 style guide for Python code. The precise Python style conventions that `pycodestyle` validates can be found in the pycodestyle Documentation.

Shell Script Code Quality

All shell scripts and BATS files contributed to the project must produce no warnings when passed to the Shellcheck static analysis tool, as made available by the `shellcheck-py` Python package.

Documentation for each specific check is documented within the Shellcheck wiki pages.

Spelling

The project expects documentation to have correct English (en-US) spelling. Words within documentation text files have their spelling validated via the `pyspellchecker` Python package.

As many project files are technical in nature with non-standard English words, a file containing a list of additional valid words exists at `meta-ewaol-config/qa-checks/ewaol-dictionary` which may be modified if the QA check erroneously highlights valid technical terminology.

YAML Formatting

All YAML files contributed to the project must pass validation as evaluated by the `yamllint` Python-based linter for YAML files, which should report no warnings or errors. This is run by default as part of the QA checks.

For more details on `yamllint` see yamllint documentation.

Yocto Layer Compatibility

Contributions must not break layer compatibility with the Yocto Project, as validated via the Yocto Project's `yocto-check-layer` script, documented as part of the Yocto Project Documentation at Yocto Check Layer Script.

As the validation script can take several minutes to run, it is not performed as part of the QA check script by default. Instead, it should be enabled by passing `--check=layer` to run only the layer compatibility check, or by passing `--check=all` to the script to run all the checks including the layer compatibility check. For example:

```
./tools/qa-checks/run-checks.py --check=layer
```

The layer compatibility QA check runs as a containerized application using Docker. Docker must therefore be installed on the host environment to perform this QA check. See the Docker documentation for installation instructions.

Further details for running the QA checks are given at *Quality Assurance Checks*.

1.5.3 Contribution Process

Adhering to the contributions guidelines listed above, contributions to the EWAOL project should be made using the process listed in this section.

Gitlab Account Setup

In order to contribute to the repository, it is necessary to have an account on https://gitlab.com/soafee. Please see **`https://gitlab.com/users/sign_in`_** for details of how to create an account.

Submission

Note: The mechanics of the EWAOL submission process has not yet been established. The process described here is therefore subject to change.

With an appropriate Gitlab account, a contribution can be submitted to https://gitlab.com/soafee/ewaol/meta-ewaol via the following process:

1. If the contribution relates to a Gitlab Issue (for example, fixes a reported bug, resolves a raised security concern, or implements a related feature request) please include the relevant `meta-ewaol` Gitlab Issue ID within the Git commit message(s) of the contribution.
2. Fork the `meta-ewaol` Gitlab repository.
3. Push changes to a branch on the forked repository. This contribution branch should be based on the latest development branch of `meta-ewaol`, which is: `kirkstone-dev`.
4. Submit a Merge Request to `meta-ewaol` using the contribution branch on the forked repository. Please include all information required by the project's Merge Request template.

1.5.4 Supporting Tools

To support contributions, the project provides tooling for building and validating the documentation, and for running automated quality-assurance validation related to the minimal standards listed in *Minimal Contribution Standards*. These tools are detailed below.

Documentation Build Validation

EWAOL provides a Python script to locally build and render the documentation, available at `tools/build/doc-build.py`. This script will install all necessary Python packages into a temporary Python Virtual Environment, and generate an HTML version of the documentation under `public/`. The script requires Python 3.8 or greater, and to build the EWAOL documentation should be called from the `meta-ewaol` directory via:

```
./tools/build/doc-build.py
```

The generated documentation can be accessed by opening `public/index.html` in a web browser.

For further information about the parameters, call the help function of the script:

```
./tools/build/doc-build.py --help
```

Quality Assurance Checks

The project provides tooling for running Quality Assurance (QA) checks on the repository. These checks aim to automatically validate that contributions adhere to a set of minimal standards, defined by the project and documented earlier at *Minimal Contribution Standards*.

The tooling is provided as a set of Python scripts that can be found within the `tools/qa-checks/` directory of the repository. In order to run the tool, the system must have installed Python 3 (version 3.8 or greater), the PyYAML Python package available via pip (5.4.1 is the project's currently supported version), and Git version 2.25 or greater.

Note: Git version 2.25 may not be available via the default PPAs included with Ubuntu 18.04. On this distribution, it can be made available via the Git stable releases PPA: `add-apt-repository ppa:git-core/ppa`

The QA-checks should be run for each commit of the contribution, by executing `run-checks.py` via the following command:

```
./tools/qa-checks/run-checks.py --check=all
```

The script should pass with no errors or warnings.

1.6 License

The software is provided under the MIT license (below).

```
Copyright (c) <year> <copyright holders>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to
deal in the Software without restriction, including without limitation the
rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice (including the next
paragraph) shall be included in all copies or substantial portions of the
Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
IN THE SOFTWARE.
```

1.6.1 SPDX Identifiers

Individual files contain the following tag instead of the full license text.

```
SPDX-License-Identifier: MIT
```

This enables machine processing of license information based on the SPDX License Identifiers that are here available: http://spdx.org/licenses/

1.7 Changelog & Release Notes

1.7.1 Unreleased

New Features

- Added support for building and running EWAOL on the AVA Developer Platform
- Added capability for Xen PCI device passthrough, currently supported for Guest VMs running on the AVA Developer Platform

Changed

Distro:

- Updated rootfs size management for EWAOL virtualization distribution images, to add build-time functionality that automatically configures the size of the Control VM rootfs sufficient to support all Guest VM root filesystems

Tests:

- Upgraded the run-time integration tests to include validation of all EWAOL Guest VMs that are deployed on an EWAOL virtualization distribution image

Tools:

- Expanded QA checks to add a non-inclusive language check module, which attempts to identify and highlight potential usage of non-inclusive terminology within the repository
- Parallelized the execution of QA checks, supported by a new `--number_threads` command line argument
- Made various other improvements to the QA checks to increase usability and maintainability
- Bumped dependency versions for the documentation build tool
- Updated kas minimal required version to `3.1`

Limitations

None.

Resolved and Known Issues

Resolved Issues:

- Resolved the K3s recipe build timeout issue reported as a *known issue* in the EWAOL `v1.0` release.
- Identified and resolved a rare race condition in the K3s integration test suite which could occasionally cause the validation to fail

1.7.2 v1.0

New Features

- Updated the EWAOL software stack to provide two system architectures: baremetal and virtualization
- Introduced EWAOL User Accounts with multi-level privileges
- Introduced EWAOL Security Hardening to reduce potential sources or attack vectors of security vulnerabilities

Changed

Third-party Yocto layers used to build the software stack:

```
URL: https://git.yoctoproject.org/git/poky
layers: meta, meta-poky
branch: kirkstone
revision: 453be4d258f71855205f45599eea04589eb4a369

URL: https://git.openembedded.org/meta-openembedded
layers: meta-filesystems, meta-networking, meta-oe, meta-python
branch: kirkstone
revision: 166ef8dbb14ad98b2094a77fcf352f6c63d5abf2

URL: https://git.yoctoproject.org/git/meta-virtualization
layers: meta-virtualization
branch: kirkstone
revision: 2fae71cdf0e8c6f398f51219bdf31eac76c662ec

URL: https://git.yoctoproject.org/git/meta-arm
layers: meta-arm, meta-arm-bsp, meta-arm-toolchain
branch: kirkstone
revision: fc09cc0e8db287600625e64905170a6de24f2686
```

Main software components versions:

- Systemd (version: `250.5`) as init system
- K3s container orchestration engine (version: `v1.22.6+k3s1+git4262c6b`)
- Docker (version: `20.10.12+ce+git906f57f`) as container engine
- runc-opencontainers (version: `1.1.0+git0+b9460f26b4`) as the OCI container runtime
- Xen (version: `4.16+stable0+f265444922`) as the type-1 hypervisor

Configs:

- Refactored kas configuration files, and separated into three ordered categories: “Architecture”, “Build Modifier” and “Target Platform” Configs

Distro:

- Introduced EWAOL Baremetal and Virtualization Distribution images
- Introduced Xen as type-1 hypervisor for EWAOL Virtualization Distribution images
- Introduced optional EWAOL Security Hardening distro feature
- Introduced EWAOL User Accounts (`ewaol`, `user` and `test`) with various privilege levels
- Introduced Filesystem Compilation Tuning where EWAOL root filesystems by default use the generic `armv8a-crc` tune for `aarch64` based target platforms
- Introduced `meta-ewaol-bsp` Yocto BSP layer with target platform specific extensions for particular EWAOL distribution images
- Introduced the following build-time kernel configuration checks:
  - K3s orchestration support
  - Xen virtualization support
- Added the installation of docker-ce instead of docker-moby on EWAOL root filesystems
- Added build information inclusion on EWAOL root filesystems

Documentation:

- Refactored the documentation structure to improve readability
- Introduced the Contribution Guidelines instructions

Tools:

- Expanded QA checks to also validate:
  - Documentation build
  - Yocto layer compatibility
  - YAML files formatting
- Generalized the documentation build tooling to allow building independent projects
- Updated Python minimal required version to `3.8`
- Updated Git minimal required version to `2.25`
- Updated kas minimal required version to `3.0.2`
- Updated kas configuration format version to `11`
- Added various fixes and improvements to QA checks tooling
- Dropped the deprecated CI-specific build tool

Tests:

- Introduced “Xen Virtualization Tests” and “User Accounts Tests” test suites
- Expanded appropriate test suites to also include validations of both Control and Guest VMs on EWAOL virtualization distribution images
- Configured all tests suites to be run as the `test` user account
- Added extra security checks for all test suites, performed when the Security Hardening distro feature is enabled

- Changed filesystem storage directories for test suite logs and temporary run-time files
- Refactored test recipes to share common code installed on the root filesystem

Limitations

None.

Resolved and Known Issues

Known Issues:

- The K3s recipe build involves fetching a substantial amount of source code which might fail due to connection timeout. If a similar error message as `ERROR: Task (/<...>/layers/meta-virtualization/recipes-containers/k3s/k3s_git.bb:do_fetch) failed with exit code '1'` is displayed, try re-running the build command until it completes.

1.7.3 v0.2.4

New Features

No new features were introduced.

Changed

Bug fixes as listed in *v0.2.4 Resolved and Known Issues*.

Limitations

None.

Resolved and Known Issues

Resolved issues from v0.2.3:

- ewaol-distro: Fix BitBake fetch for ostree recipe from meta-oe

1.7.4 v0.2.3

New Features

No new features were introduced.

Changed

Bug fixes as listed in *v0.2.3 Resolved and Known Issues*.

Limitations

None.

Resolved and Known Issues

Resolved issues from v0.2.2:

- qa-checks: Install pip for Python 3.6
- ewaol-distro: Fix BitBake fetch for runc-opencontainers recipe from meta-virtualization

1.7.5 v0.2.2

New Features

No new features were introduced.

Changed

Bug fixes as listed in *v0.2.2 Resolved and Known Issues*.

Limitations

None.

Resolved and Known Issues

Resolved issues from v0.2.1:

- ewaol-distro: libpcre and libpcre2 to fetch from sourceforge and github

1.7.6 v0.2.1

New Features

No new features were introduced.

Changed

Bug fixes as listed in *v0.2.1 Resolved and Known Issues*.

Limitations

None.

Resolved and Known Issues

Resolved issues from v0.2:

- qa-checks: shell check running in all relevant files within the repository
- qa-checks: shell check SC2288 fixes for integration tests scripts
- qa-checks: Consider latest git commit for matching file's copyright year
- qa-checks: Fix getting the last modification date of external works
- qa-checks: Disable SC2086 shellcheck for k3s-killall.sh from K3s package
- ewaol-distro: Fix BitBake fetch for go-fsnotify recipe from meta-virtualization

1.7.7 v0.2

New Features

- Introduced K3s container orchestration support, as well as its integration tests
- Removed support for the FVP Base-A reference platform
- Introduced EWAOL Software Development Kit (SDK) distro image type which includes packages and features to support software development on the target

Changed

Third-party Yocto layers used to build the software stack:

```
URI: git://git.yoctoproject.org/poky
layers: meta, meta-poky
branch: hardknott
revision: 269265c00091fa65f93de6cad32bf24f1e7f72a3

URI: git://git.openembedded.org/meta-openembedded
layers: meta-filesystems, meta-networking, meta-oe, meta-perl, meta-python
branch: hardknott
revision: f44e1a2b575826e88b8cb2725e54a7c5d29cf94a

URI: git://git.yoctoproject.org/meta-security
layers: meta-security
branch: hardknott
revision: 16c68aae0fdfc20c7ce5cf4da0a9fff8bdd75769
```

(continues on next page)

(continued from previous page)

```
URI: git://git.yoctoproject.org/meta-virtualization
layers: meta-virtualization
branch: hardknott
revision: 7f719ef40896b6c78893add8485fda995b00d51d

URI: git://git.yoctoproject.org/meta-arm
layers: meta-arm, meta-arm-bsp, meta-arm-toolchain
branch: hardknott
revision: 71686ac05c34e53950268bfe0d52c3624e78c190
```

Main software components versions:

- Systemd (version: `247.6`) as init system
- K3s container orchestration engine (version: `v1.20.11+k3s2`)
- Docker (version: `20.10.3+git11ecfe8a81b7040738333f777681e55e2a867160`) or Podman (version: `3.2.1+git0+ab4d0cf908`) as container engines
- runc-opencontainers (version: `1.0.0+rc93+git0+249bca0a13`) as the OCI

Configs:

- Only include meta-arm layers when required

Distro:

- Introduced EWAOL Software Development Kit (SDK) distro image type
- Introduced K3s container orchestration support

Documentation:

- Refactored README.md to not include it in the final rendered documentation

Tools:

- Introduced the kas-runner.py tool to support loading build environment configurations from yaml files. This tool is still in experimental stage and will be replacing kas-ci-build.py in the future
- Added '-j' and '–out-dir' parameters to kas-ci-build.py set the maximum number of CPU threads available for BitBake and allow user to change build directory
- Moved project specific configurations for QA checks to meta-ewaol-config
- Various improvements in QA checks for spelling, commit message and license header

Tests:

- Introduced K3s container orchestration integration tests
- Improved tests logging and cleanup tasks
- Multiple tests suites share the same base directory structure and common files

Limitations

None.

Resolved and Known Issues

None.

1.7.8 v0.1.1

New Features

No new features were introduced.

Changed

Documentation:

- Added manual BitBake build preparation documentation
- Added QA checks documentation
- Added meta-ewaol public repository URL
- CI Build Tool documentation fixes
- Refactor Sphinx auto section labels and cross-references links
- Added public documentation URL
- Added link to SOAFEE URL
- Refactored Layer Dependencies on README.md
- Added Gitlab Pages integration via .gitlab-ci.yml
- Updated kas installation instructions
- Increased the free storage requirement for building to 65 GBytes

Limitations

Same as *v0.1 Limitations*.

Resolved and Known Issues

None.

1.7.9 v0.1

New Features

The following features and components are included into the reference software stack implementation:

- EWAOL Yocto distribution based on `poky.conf` distro
- Systemd (version: `247.6`) as init system
- Docker (version: `20.10.3+git11ecfe8a81b7040738333f777681e55e2a867160`) or Podman (version: `3.2.1+git0+ab4d0cf908`) as container engines
- runc-opencontainers (version: `1.0.0+rc93+git0+249bca0a13`) as the OCI container runtime
- Development and Test image flavors
- Container engine tests
- Container runtime Kernel configuration check

Supported Arm Reference Platforms:

- Armv8-A Base RevC AEM FVP (FVP-Base) with `FVP_Base_RevC-2xAEMvA_11.14_21.tgz` package version.
- N1SDP

Quality Assurance Checks Tooling:

- Source code:
  - Shell scripts: shellcheck-py module
  - Python: pycodestyle module (PEP8)
  - Copyright notice inclusion
  - SPDX license identifier inclusion
- Documentation spelling (pyspellchecker module)
- Commit message rules

Build Tools:

- Documentation build
- CI build

Documentation Pages:

- Overview
- Project Quickstart
- Image Builds
- Image Validation
- Yocto Layers
- Codeline Management
- Tools
- License
- Changelog & Release Notes

Third-party Yocto layers used to build the software stack:

```
URI: git://git.yoctoproject.org/poky/meta
branch: hardknott
revision: da0ce760c5372f8f2ef4c4dfa24b6995db73c66c

URI: git://git.yoctoproject.org/poky/meta-poky
branch: hardknott
revision: da0ce760c5372f8f2ef4c4dfa24b6995db73c66c

URI: git://git.openembedded.org/meta-openembedded
branch: hardknott
revision: c51e79dd854460c6f6949a187970d05362152e84

URI: git://git.yoctoproject.org/meta-security
branch: hardknott
revision: c6b1eec0e5e94b02160ce0ac3aa9582cbbf7b0ed

URI: git://git.yoctoproject.org/meta-virtualization
branch: hardknott
revision: 3508b13acbf669a5169fafca232a5c4ee705dd16

URI: git://git.yoctoproject.org/meta-arm
branch: hardknott
revision: e82d9fdd49745a6a064b636f2ea1e02c1750d298
```

Changed

Initial version.

Limitations

- FVP-Base build and emulation only supported on x86_64-linux hosts

Resolved and Known Issues

None.