Cassini

Linaro Limited.

Oct 29, 2025

CONTENTS

CHAPTER

ONE

INTRODUCTION

Project Cassini is the open, collaborative, standards-based initiative to deliver a seamless cloud-native software experience for devices based on Arm Cortex-A.

Current release of Cassini distribution provides a framework for deployment and orchestration of applications (edge runtime) within containers and support for platform abstraction for security (PARSEC).

Cassini distribution includes support for provisioning the platform and updating cassini distribution software stack over the air.

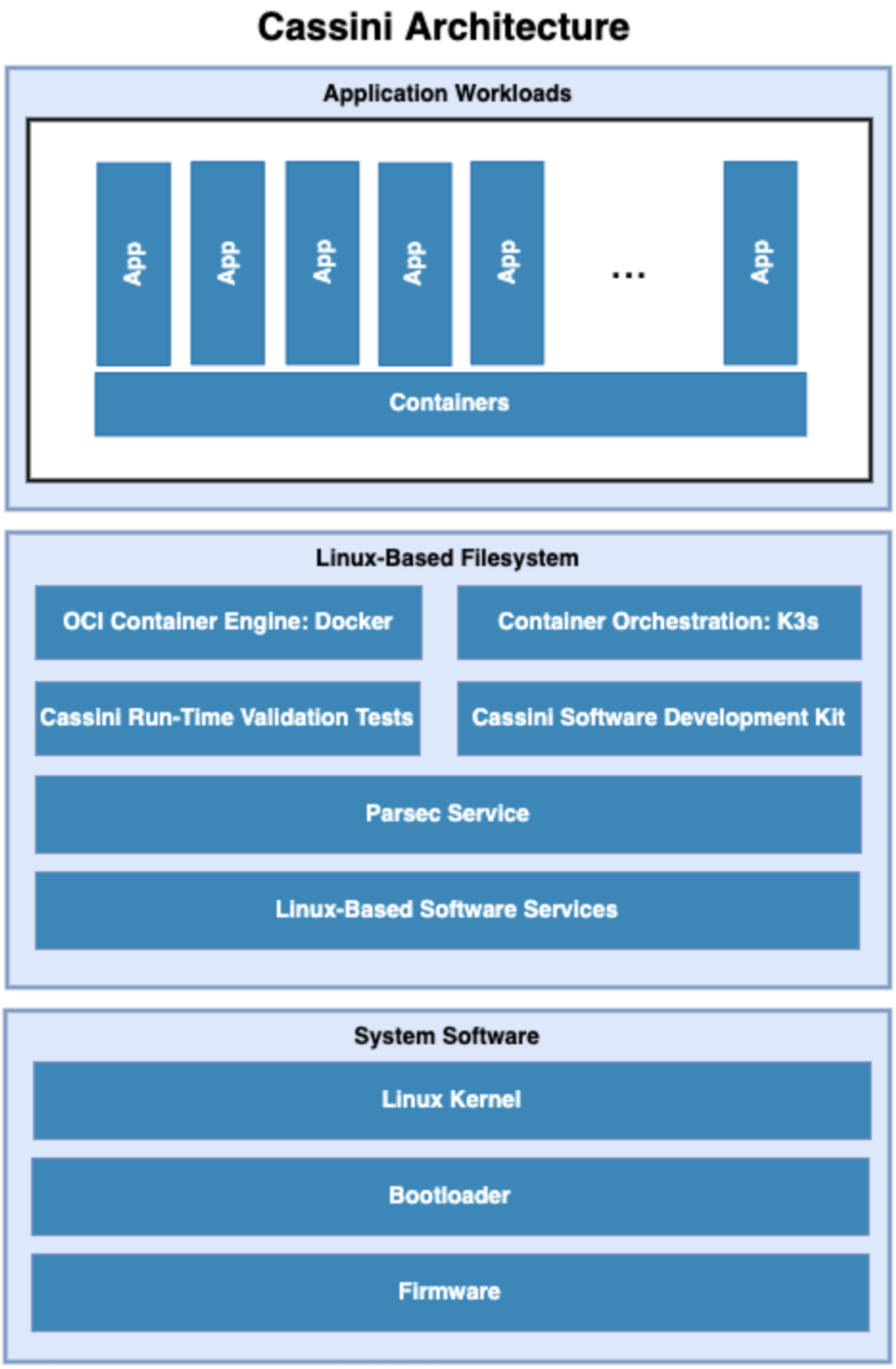
1.1 Use-Case Overview

Cassini aims to facilitate the deployment of application workloads via Docker and K3s use-case on the supported target platforms.

Instructions for achieving these use-cases are given in the *build* section, subject to relevant assumed technical knowledge as listed later in *documentation assumptions*.

1.2 Architecture

The following diagram illustrates the Cassini Architecture.


Cassini Architecture
Application Workloads
App
App
App
App
App
...
App
Containers
Linux-Based Filesystem
OCI Container Engine: Docker
Container Orchestration: K3s
Cassini Run-Time Validation Tests
Cassini Software Development Kit
Parsec Service
Linux-Based Software Services
System Software
Linux Kernel
Bootloader
Firmware

The different software layers are described below:

- **Application workloads**:

  User-defined container applications that are deployed and executed on the Cassini software stack. Note that the Cassini 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 Cassini project. The Cassini filesystem contains tools and services that provide Cassini core functionalities and facilitate secure deployment and orchestration of user application workloads. These tools and services include the Parsec service, the Docker container engine, the K3s container orchestration framework, together with their run-time dependencies. In addition, Cassini 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.

1.3 Features Overview

Cassini includes the following major features:

- Container engine and runtime with Docker and runc-opencontainers.
- Container workload orchestration with the K3s Kubernetes distribution.
- Parsec service and Parsec tool
- On-target development support with optionally included Software Development Kit.
- FTPM showcase and TPM tools.
- AWS greengrass integration.
- Device provisioning and over-the-air system update with mender.
- Validation support with optionally included run-time integration tests, and build-time kernel configuration checks.

Other features of Cassini include:

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

1.3.1 Documentation Assumptions

This documentation assumes a base level of knowledge related to different aspects of achieving the target use-case via Cassini:

- 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 Cassini-specific instructions to support these activities on an Cassini distribution image.

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

- The Yocto Project

  This documentation contains instructions for achieving Cassini's use-case using a set of included configuration files that provide standard build features and settings. However, Cassini 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 Cassini-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 Cassini distribution builds.

1.4 Repository Structure

The `meta-cassini` repository is structured as follows:

- `meta-cassini`:
  - `meta-cassini-bsp`

    A Yocto layer which holds board-specific recipes or append files that either:

    - will not be upstreamed (Cassini specific modifications)
    - have not been upstreamed yet
  - `meta-cassini-distro`

    A Yocto distribution layer providing top-level and general policies for the Cassini distribution images.
  - `meta-cassini-tests`

    A Yocto software layer with recipes that include run-time tests to validate Cassini functionalities.
  - `kas`

    Directory which contains files to support use of the kas build tool.

1.5 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.

1.6 Contributions and Issue Reporting

Guidance for contributing to the Cassini 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.

For known issues in this release, see *Release Notes*.

1.7 Feedback and support

To request support please contact Linaro at support@linaro.org.

1.8 Maintainer(s)

- Cassini Team

CHAPTER

TWO

USER MANUAL

2.1 Build, Deploy and Validate Cassini Image

The recommended approach for image build setup and customization is to use the kas build tool. To support this, Cassini 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 Cassini, before guidance is given on using those kas configuration files to set up the Cassini 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 `kas` directory contains kas configuration files to support building and customizing Cassini distribution images via kas. These kas configuration files contain default parameter settings for a Cassini distribution build. Here, the files are briefly introduced, classified by type:

- **Base Configs**: Configures common software components
  - `cassini.yml` to build an image for the Cassini distribution.
- **Build Cloud Configs**: Set and configure container runtime and cloud service
  - `no-cloud.yml` to include the default container runtime without a cloud service.
  - `greengrass.yml` to include Docker and the AWS IoT Greengrass V2 cloud service.

  If no cloud config is specified, Docker and K3s orchestration is included by default
- **Build OTA Update Config**: Set and configure Over-the-air update service
  - `no-ota.yml` to remove over-the-air update service.

  Mender is included by default
- **Build Modifier Configs**: Set and configure features of the Cassini distribution
  - `dev.yml` to configure the image for development using debug tweaks and disable *Security Hardening*.
  - `tests.yml` to include run-time validation tests into the image with debug tweaks.
- **Target Platform Configs**: Set the target platform

  For information on supported targets in Cassini and corresponding value for `MACHINE` **variable**, refer to *Target Platforms*.

These kas configuration files can be used to build a custom Cassini distribution by passing the **Base Config** and one **Target Platform Config** to the kas build tool. **Build Cloud Configs**, **Build OTA Update Config** and **Build Modifier Configs** are optional (only one of each can be included). Configuration files are separated with a colon in the kas execution command line, see examples below:

```
kas build <Base Config>:<Build Cloud Configs>:<Build OTA Update Config>:<Build Modifier␣
↪Configs>:<Target Platform Config>
```

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

2.1.1 Build Host Environment Setup

This documentation assumes an Ubuntu based build host, where the build steps have been validated on the Ubuntu 20.04 LTS (Focal Fossa) and 22.04 LTS (Jammy Jellyfish).

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 Cassini is to use the kas build tool. To install kas:

```
python3 -m pip install kas==4.3.2
```

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

To deploy a Cassini distribution image onto the supported target platform, `bmap-tools` is used. 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 a Cassini distribution image.

2.1.2 Download

The `meta-cassini` 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

git clone https://gitlab.com/Linaro/cassini/meta-cassini.git --branch main
cd meta-cassini
```

2.1.3 Build and Deploy

Refer to the platform guides instructions on how to build and deploy the Cassini images on supported platforms:

- *Getting Started with Arm Corstone-1000 for MPS3*
- *Getting Started with Arm Corstone-1000 FVP*
- *Getting Started with KV260*
- *Getting Started with Generic Arm64 Images*

2.1.4 Run

To run the deployed Cassini distribution image, simply boot the target platform.

The Cassini distribution image can be logged into as `cassini` user.

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

2.1.5 Validate

As an initial validation step, check that the appropriate Systemd services are running successfully,

- `docker.service`
- `k3s.service` is available unless a cloud modifier is included as part of the build config.
- `greengrass.service` is available if `greengrass.yml` is included as part of the build config.

A service can be checked by running the command:

```
systemctl status --no-pager --lines=0 <systemd.service>
```

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

More thorough run-time validation of Cassini components are provided as a series of integration tests, available if the `kas/tests.yml` kas configuration file was included in the image build.

Note: Due to performance limitations, K3s is not currently supported on the Arm Corstone-1000.

2.1.6 Reproducing the Cassini 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.

Deploying Application Workloads via Docker and K3s

This example deploys the Nginx web server 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. Boot the image and log-in as `cassini` user.
2. Ensure the target device can access the internet

   ```
   wget www.linaro.org
   ```

   The output should be similar to:

   ```
   --2023-12-02 12:42:10--  http://www.linaro.org/
   Resolving www.linaro.org... 18.165.227.69, 18.165.227.126, 18.165.227.43, ..
   ↪.
   Connecting to www.linaro.org|18.165.227.69|:80... connected.
   HTTP request sent, awaiting response... 301 Moved Permanently
   Location: https://www.linaro.org/ [following]
   --2023-12-02 12:42:10--  https://www.linaro.org/
   Connecting to www.linaro.org|18.165.227.69|:443... connected.
   HTTP request sent, awaiting response... 200 OK
   Length: 54811 (54K) [text/html]
   Saving to: 'index.html.1'

   index.html    100%[===============>]  53.53K   323KB/s    in 0.2s

   2023-12-02 12:42:26 (323 KB/s) - 'index.html' saved [54811/54811]
   ```

3. Deploy the example application workload:

   - **Deploy via Docker**

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

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

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

     ```
     sudo docker container list
     ```

   - **Deploy via K3s**

     3.1. 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
     ```

(continues on next page)

(continued from previous page)

```
      hostPort: 8082
EOT
```

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

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

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

```
wget localhost:8082
```

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

Note: Due to performance limitations, K3s is not currently supported on the Arm Corstone-1000.

2.2 Getting Started with Arm Corstone-1000 for MPS3

This document explains how to build, deploy, and boot the Cassini distro on the Arm Corstone-1000 for MPS3.

NOTE: Requires a micro SD card (at least 8 GB) and a USB drive (at least 16 GB)

Note: Due to performance limitations, K3s is not currently supported on the Arm Corstone-1000 for MPS3.

2.2.1 Build

The kas configuration file `kas/corstone1000-mps3.yml` can be used to build images which target the Corstone-1000 for MPS3.

2.2.2 Building MPS3 images

To build Corstone-1000 MPS3 images with default options:

```
kas build --update kas/cassini.yml:kas/corstone1000-mps3.yml
```

This will produce a Corstone-1000 firmware image here:

`build/tmp/deploy/images/corstone1000-mps3/corstone1000-flash-firmware-image-corstone1000-mps3.wic`

And a Cassini distribution image here:

`build/tmp/deploy/images/corstone1000-mps3/cassini-image-base-corstone1000-mps3.wic.gz`

    build/tmp/deploy/images/corstone1000-mps3/cassini-image-base-corstone1000-mps3.
    wic.bmap

For other build options, refer to *Build System*

2.2.3 Prepare the firmware image for FPGA (Micro SD card)

The user should download the FPGA bit file image from this link and under the section `AN550: Arm® Corstone™-1000 for MPS3 Version 2.0`.

Only copy the current directory structure under Boardfiles shown below on to the Micro SD Card.

```
config.txt
MB
├── BRD_LOG.TXT
├── HBI0309B
│   ├── AN550
│   │   ├── AN550_v2.bit
│   │   ├── an550_v2.txt
│   │   └── images.txt
│   ├── board.txt
│   └── mbb_v210.ebf
└── HBI0309C
    ├── AN550
    │   ├── AN550_v2.bit
    │   ├── an550_v2.txt
    │   └── images.txt
    ├── board.txt
    └── mbb_v210.ebf
SOFTWARE
├── an550_st.axf
├── bl1.bin
├── cs1000.bin
└── ES0.bin
```

To configure the board to boot automatically when powered on, edit `./config.txt` and change the value of `AUTORUN` from `FALSE` to `TRUE`.

Depending upon the MPS3 board version (printed on the MPS3 board HBI0309B or HBI0309C) you should update the `./AN550/images.txt` file so that the file points to the images under SOFTWARE directory.

Here is an example

```
;************************************************
;       Preload port mapping                    *
;************************************************
;  PORT 0 & ADDRESS: 0x00_0000_0000 QSPI Flash (XNVM) (32MB)
;  PORT 0 & ADDRESS: 0x00_8000_0000 OCVM (DDR4 2GB)
;  PORT 1        Secure Enclave (M0+) ROM (64KB)
;  PORT 2        External System 0 (M3) Code RAM (256KB)
;  PORT 3        Secure Enclave OTP memory (8KB)
;  PORT 4        CVM (4MB)
;************************************************

[IMAGES]
```

(continues on next page)

(continued from previous page)

```
TOTALIMAGES: 3      ;Number of Images (Max: 32)

IMAGE0PORT: 1
IMAGE0ADDRESS: 0x00_0000_0000
IMAGE0UPDATE: RAM
IMAGE0FILE: \SOFTWARE\bl1.bin

IMAGE1PORT: 0
IMAGE1ADDRESS: 0x00_0000_0000
IMAGE1UPDATE: AUTOQSPI
IMAGE1FILE: \SOFTWARE\cs1000.bin

IMAGE2PORT: 2
IMAGE2ADDRESS: 0x00_0000_0000
IMAGE2UPDATE: RAM
IMAGE2FILE: \SOFTWARE\es0.bin
```

The binaries are present in OUTPUT_DIR = `<_workspace>/build/tmp/deploy/images/corstone1000-mps3` directory.

1. Copy `bl1.bin` from OUTPUT_DIR to SOFTWARE directory of the Micro SD card.
2. Copy `corstone1000-flash-firmware-image-corstone1000-mps3.wic` from OUTPUT_DIR directory to SOFTWARE directory of the Micro SD card and rename the wic image to `cs1000.bin`.
3. Copy `es_flashfw.bin` from OUTPUT_DIR directory to SOFTWARE directory of the Micro SD card and rename to `es0.bin`.

NOTE: Renaming of the images are required because MCC firmware has limitation of 8 characters before .(dot) and 3 characters after .(dot).

2.2.4 Prepare the distro image for FPGA (USB image)

Use the `lsblk` command to determine USB drive and bmap tool to copy the cassini distro to it.

```
lsblk
sudo bmaptool copy --bmap cassini-image-base-corstone1000-mps3.wic.bmap cassini-image-
↪base-corstone1000-mps3.wic.gz /dev/<usb drive>
```

Note: *bmaptool* may fail if the host auto-mounts partitions on the USB drive. If these issues are seen then, disable auto-mounting or use *parted* or *fdisk* to remove any existing partitions.

2.2.5 Running the software on FPGA

Insert SD card and USB drive before switching ON the device.

On the host machine, connect the board via USB.

If there are no other TTY USB devices, then the three ports from the MPS3 will be connected as follows:

- ttyUSB0 for MCC, OP-TEE and Secure Partition
- ttyUSB1 for Boot Processor (Cortex-M0+)
- ttyUSB2 for Host Processor (Cortex-A35)

The rest of this guide assumes there are no other TTY USB devices on the host machine.

Connect to the serial console(s) using any terminal client (`picocom`, `minicom`, or `screen` should all work).

For example, run the following commands to open new picocom sessions for each port:

```
sudo picocom -b 115200 /dev/ttyUSB0
sudo picocom -b 115200 /dev/ttyUSB1
sudo picocom -b 115200 /dev/ttyUSB2
```

Note: `sudo` should not be required if the current user is in the `dialout` group

Note: See notes under *Run-Time Integration Tests* before running validation steps.

Note: See notes under Corstone-1000 SystemReady IR on USB drive models stable with MPS3 FPGA.

2.3 Getting Started with Arm Corstone-1000 FVP

This document explains how to build and boot the Cassini distro on the Arm Corstone-1000 FVP (Fast Model Fixed Virtual Platform).

Note: Due to performance limitations, K3s is not currently supported on the Arm Corstone-1000 FVP.

2.3.1 Build

The provided kas configuration file `kas/corstone1000-fvp.yml` can be used to build images that are targeting the Corstone-1000 FVP.

Note: To build and run any image for the Corstone-1000 FVP the user has to accept its EULA, which can be done by executing the following command in the build environment:

```
export FVP_CORSTONE1000_EULA_ACCEPT=True
```

2.3.2 Building FVP images

To build Corstone-1000 FVP images with default options:

```
kas build --update kas/cassini.yml:kas/corstone1000-fvp.yml
```

Or if using kas-container:

```
kas-container --runtime-args "-e FVP_CORSTONE1000_EULA_ACCEPT=True" build \
kas/cassini.yml:kas/corstone1000-fvp.yml
```

This will produce a Corstone-1000 firmware image here:

`build/tmp/deploy/images/corstone1000-fvp/corstone1000-flash-firmware-image-corstone1000-fvp.wic`

And a Cassini distribution image here:

`build/tmp/deploy/images/corstone1000-fvp/cassini-image-base-corstone1000-fvp.wic`

For other build options, refer to *Build System*

2.3.3 Running the FVP

To start the FVP and get the console:

```
kas shell -c "../layers/meta-arm/scripts/runfvp --console" \
kas/cassini.yml:kas/corstone1000-fvp.yml
```

Or if using kas-container:

```
kas-container --runtime-args "-e FVP_CORSTONE1000_EULA_ACCEPT=True" \
shell -c "/work/layers/meta-arm/scripts/runfvp --console" \
kas/cassini.yml:kas/corstone1000-fvp.yml
```

By default, the Corstone-1000 FVP is configured for user mode networking. For more information and instructions on how to configure networking with Fixed Virtual Platforms, refer to the Fast Models Reference Guide.

Note: See notes under *Run-Time Integration Tests* before running validation steps.

2.3.4 Validation

The following validation tests can be performed on the Cassini Reference Stack:

- System Integration Tests:
  - Cassini Architecture Stack:

    ```
    TESTIMAGE_AUTO=1 kas build kas/cassini.yml:kas/corstone1000-fvp.yml
    ```

    Or if using kas-container:

```
kas-container --runtime-args "-e FVP_CORSTONE1000_EULA_ACCEPT=True -e TESTIMAGE_
↪AUTO=1" build \
kas/cassini.yml:kas/corstone1000-fvp.yml
```

The previous test takes around 2 minutes to complete.

A similar output should be printed out:

```
NOTE: Executing Tasks
Creating terminal default on host_terminal_0
default: Waiting for login prompt
RESULTS:
RESULTS - linuxboot.LinuxBootTest.test_linux_boot: PASSED (23.70s)
SUMMARY:
cassini-image-base () - Ran 1 test in 23.704s
cassini-image-base - OK - All required tests passed (successes=1, skipped=0,␣
↪failures=0, errors=0)
```

2.4 Getting Started with KV260

This document explains how to build, deploy, and boot the Cassini distro on Xilinx KV260 Platform.

2.4.1 Building KV260 Images

Note: When building on `main` branch, the preparation script: `kas/scripts/generate_kv260_env.py` must be executed before attempting the following steps. This script pins the SHAs for layers involved in KV260 build according to the meta-ts base.yml to follow the same update pace as meta-ts.

One of the provided kas configuration files can be used to build images which target KV260:

1. `kas/kv260-psa.yml` uses U-Boot as BL33, and has trusted-services.
2. `kas/kv260-edk2.yml` uses EDK-II as BL33, and has trusted-services.
3. `kas/kv260-ftpm.yml` uses U-Boot as BL33, and has fTPM, and no trusted-services.

These will be referred to below as <kv260-variant>

To build an image with default options for any variant:

```
kas build --update kas/cassini.yml:kas/<kv260-variant>.yml
```

Builds will produce the firmware images here:

`build/tmp/deploy/images/<zynqmp-machine>/ImageA.bin`

`build/tmp/deploy/images/<zynqmp-machine>/ImageB.bin`

and a Cassini distribution image here:

`build/tmp/deploy/images/<zynqmp-machine>/cassini-image-base-<zynqmp-machine>.rootfs.wic.gz`

`build/tmp/deploy/images/<zynqmp-machine>/cassini-image-base-<zynqmp-machine>.rootfs.wic.bmap` where <zynqmp-machine> can be derived from this table:

Table 1: <zynqmp-machine> Values

<kv260-variant>	<zynqmp-machine>
kv260-psa	zynqmp-kria-starter-psa
kv260-edk2	zynqmp-kria-starter-psa
kv260-ftpm	zynqmp-kria-starter

For other build options, refer to *Build System*

2.4.2 Flashing the Firmware

1. Connect KV260 Ethernet port to **the host machine**
2. Power up the device while holding **FWUEN** button
3. In a browser, visit: `http://192.168.0.111/`, this will open **Xilinx tool** for flashing the firmware
4. Upload ImageA.bin and ImageB.bin
5. Reset the device

Note: The ethernet port on **the host machine** must be configured to have an IP address on the same network as `192.168.0.111`. For example:

- IP address: `192.168.0.110`
- Subnet mask: `255.255.0.0`

2.4.3 Flashing the Distro Image

1. Insert the SD card into **the host machine**
2. Check if the SD card is seen by **the host machine** via `lsblk`.

   This will output, for example:

```
NAME        MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
<sd card>   179:0    0    16G  0 disk
├─p1        179:1    0   256M  0 part
└─p2        179:2    0   512M  0 part
```

3. Flash the image onto the SD card using `bmap-tools`:

```
sudo bmaptool copy --bmap cassini-image-base-<zynqmp-machine>.rootfs.wic.bmap␣
↪cassini-image-base-<zynqmp-machine>.rootfs.wic.gz /dev/<sd card>
```

4. Eject the SD card from **the host machine**, and insert it into KV260

2.4.4 Connecting to the serial port

1. Connect a cable between the USB port of **the host machine** and the micro-USB port of KV260 and then power on the device.
2. Check for new TTY USB devices detected by **the host machine**, via:

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

   This will output, for example:

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

   - `ttyUSB1` is used for logs of both secure and non-secure sides.
   - PMU uses one of the other ports, while the rest are not used at the moment.
3. Connect to the serial console using any terminal client (picocom, minicom, or screen should all work).

   ```
   sudo picocom -b 115200 /dev/ttyUSB1
   ```

2.5 Getting Started with Generic Arm64 Images

This document explains how to build and deploy a generic Arm64 distro image to an SD card.

Note: Refer to the platform's user manual for details on how to configure the platform to boot the deployed image.

2.5.1 Building Generic Arm64 Images

The provided kas configuration file `kas/genericarm64.yml` can be used to build images which target the arm64 machines. To build an image with default options:

```
kas build --update kas/cassini.yml:kas/genericarm64.yml
```

This will produce a Cassini generic Arm64 distribution image here:

> `build/tmp/deploy/images/genericarm64/cassini-image-base-genericarm64.rootfs.wic.gz`
>
> `build/tmp/deploy/images/genericarm64/cassini-image-base-genericarm64.rootfs.wic.bmap`

For other build options, refer to *Build System*

Note: This machine does not build any firmware components. For firmware build options, please refer to the platform specific guidelines.

2.5.2 Flashing the Distro Image

1. Insert the SD card into **the host machine**
2. Check if the SD card is seen by **the host machine** via `lsblk`.

   This will output, for example:

```
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
<sd card>    179:0    0    16G  0 disk
├─p1         179:1    0   256M  0 part
└─p2         179:2    0   512M  0 part
```

3. Flash the image onto the SD card using `bmap-tools`:

```
sudo bmaptool copy --bmap cassini-image-base-genericarm64.rootfs.wic.bmap␣
↪cassini-image-base-genericarm64.rootfs.wic.gz /dev/<sd card>
```

4. Eject the SD card from **the host machine**, and insert it into any arm64 machine

CHAPTER

THREE

DEVELOPER MANUAL

3.1 User Accounts

Cassini distribution images contain the following user accounts:

- `root` with administrative privileges enabled by default. The login is disabled if `cassini-security` is included in `DISTRO_FEATURES`.

  Note: When `cassini-test` distro feature is enabled then `root` login is enabled. Currently, running `inline tests` in LAVA require login as `root` to run transfer-overlay commands.

- `cassini` with administrative privileges enabled with `sudo`.
- `user` without administrative privileges.
- `test` with administrative privileges enabled with `sudo`. This account is created only if `cassini-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-cassini-distro/recipes-extended/sudo/files/cassini_admin_group.in`.

If `cassini-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.

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

3.2 Build System

A Cassini distribution can be built by setting the target platform via the `MACHINE` BitBake variable. In addition, the desired distribution features via the `DISTRO_FEATURES` BitBake variable. Finally, customizing those features via feature-specific modifiable variables, if needed.

This chapter provides an overview of Cassini's support for the kas build tool. All the available distribution image features and supported target platforms are 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.

3.2.1 kas Build Tool Support

The kas build tool enables automatic fetch and inclusion of layer sources, as well as parameter and feature specifications 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 `kas` directory contains kas configuration files that support building images via kas for the Cassini project, and fall into three ordered categories:

- **Base Config**
- **Build Modifier Configs**
- **Target Platform Configs**

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

To enable builds for a supported target platform or configure each Cassini distribution image feature, kas configurations files 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 and Deploy* section of the User Manual.

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

The kas build tool also provides a graphical user interface to build the Cassini distribution with different configs:

```
kas menu kas/Kconfig
```

3.2.2 Target Platforms

Corstone-1000 for MPS3

- **Corresponding value for `MACHINE` variable**: `corstone1000-mps3`
- **Target Platform Config**: `kas/corstone1000-mps3.yml`

To read documentation about the Corstone-1000, see the Arm Corstone-1000 Technical Overview.

For more information about the software stack for the Corstone-1000, see Arm Corstone-1000 Software.

Corstone-1000 FVP

- **Corresponding value for `MACHINE` variable**: `corstone1000-fvp`
- **Target Platform Config**: `kas/corstone1000-fvp.yml`

To read documentation about the Corstone-1000 FVP, see the Fast Models Fixed Virtual Platforms (FVP) Reference Guide.

Kria KV260 with U-boot

- **Corresponding value for `MACHINE` variable**: `zynqmp-kria-starter-psa`
- **Target Platform Config**: `kas/kv260-psa.yml`

This supported target platform is Kria KV260, implemented in meta-trustedsubstrate.

This target uses U-Boot as BL33.

To read the Kria KV260 documentation, see: Kria KV260 User Guide and Kria KV260 Data sheet

Kria KV260 with EDK-II

- **Corresponding value for `MACHINE` variable**: `zynqmp-kria-starter-psa`
- **Target Platform Config**: `kas/kv260-edk2.yml`

This supported target platform is Kria KV260, implemented in meta-trustedsubstrate.

This target uses EDK-II as BL33.

Kria KV260 with fTPM

- **Corresponding value for `MACHINE` variable**: `zynqmp-kria-starter`
- **Target Platform Config**: `kas/kv260-ftpm.yml`

This supported target platform is Kria KV260, implemented in meta-trustedsubstrate.

This target uses U-Boot as BL33. It also uses fTPM, and has no trusted services.

Generic Arm64

- **Corresponding value for `MACHINE` variable**: `genericarm64`
- **Target Platform Config**: `kas/genericarm64.yml`

This MACHINE is used to represent a generic 64-bit platform that implements the Arm SystemReady specification, has a working firmware and can boot via EFI.

This machine configuration does not build any firmware components.

All the above include common configuration from `kas/include/arm-base.yml` which defines layers and dependencies required when building for all Arm-based platforms.

Moreover, `corstone1000-fvp.yml`, and `corstone1000-mps3.yml` **Target Platform Config** include common configuration for Arm-maintained platforms from `kas/include/arm-machines.yml` which defines the BSPs, layers, and dependencies required when building for each platform.

While all `kv260-*.yml` **Target Platform Config** include configuration from `kas/include/xilinx-machines.yml` which defines specific layers and dependencies required when building for this platform.

3.2.3 Distribution Image Features

For a particular target platform, the available Cassini 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.

Cassini Architecture

Cassini distribution image can be configured via kas using **Base Config**. This includes a set of common configuration from a base Cassini kas configuration file:

- `kas/include/cassini-base.yml`

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

- **Corresponding value in** `DISTRO` **variable**: `cassini`.
- **Base Config**: `kas/cassini.yml`.

This Cassini distribution image feature enables the `cassini-image-base` build target, to build an Cassini distribution image.

The **Base Config** for this distribution image feature sets the build target to `cassini-image-base`.

To build Cassini distribution image, provide the **Base Config** to the kas build command. For example, to build a Cassini distribution image for the KV260 hardware target platform, run the following command:

```
kas build kas/cassini.yml:kas/kv260-psa.yml
```

Other Cassini Features

Developer Support

- **Corresponding value in** `DISTRO_FEATURES` **variable**: `cassini-dev`.
- **Build Modifier Config**: `kas/dev.yml`.

This Cassini distribution feature includes packages appropriate for development, such as the `allow-empty-password empty-root-password allow-root-login post-install-logging` packages and removing security-hardening features.

```
kas build kas/cassini.yml:kas/dev.yml:kas/kv260-psa.yml
```

Run-Time Integration Tests

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

  This Cassini distribution feature includes the Cassini test suites provided to validate the image is running successfully with the expected Cassini 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-cassini-tests` as a Yocto layer source for the build, and appends the `cassini-test` feature to `DISTRO_FEATURES` for the build.

  To include run-time integration tests in a Cassini distribution image, provide the **Build Modifier Config** to the kas build command. For example, to include the tests in a Cassini distribution image for the KV260 hardware target platform, run the following command:

  ```
  kas build kas/cassini.yml:kas/tests.yml:kas/kv260-psa.yml
  ```

  The size of the root filesystem is extended via the `CASSINI_ROOTFS_EXTRA_SPACE` BitBake variable, to `2000000` Kilobytes, which is required by this integration tests if *Parsec service* is enabled.

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

Cloud Service

This Cassini distribution feature includes container runtime and orchestration components.

`VIRTUAL-RUNTIME_cloud_service` is used to select one of three options.

No Cloud

- **Corresponding value in** `DISTRO_FEATURES` **variable**: `cassini-cloud`.
- **Corresponding value in** `VIRTUAL-RUNTIME_cloud_service` **variable**: `no-cloud`.
- **Build Cloud Configs**: `kas/no-cloud.yml`.

  This Cassini distribution feature provides an image with a default container runtime option (podman) from meta-virtualization without including any cloud provider.

  To build a Cassini distribution image without a cloud provider, provide the **Build Cloud Config** to the kas build command. For example:

  ```
  kas build kas/cassini.yml:kas/no-cloud.yml:kas/kv260-psa.yml
  ```

K3s orchestration

- **Corresponding value in `DISTRO_FEATURES` variable**: `cassini-cloud`.
- **Corresponding value in `VIRTUAL-RUNTIME_cloud-service` **variable**: ``k3s-cloud`.
- **Build Cloud Configs**: N/A (enabled by default).

  This Cassini distribution feature includes the K3s cloud orchestration.

  If *Run-Time Integration Tests* Build Modifier for this distribution image feature then it will automatically include the required *K3s Orchestration Tests* test in the ptest framework of the image.

  To include K3s orchestration in a Cassini distribution image, provide the **Build Cloud Config** to the kas build command. For example:

  The size of the root filesystem is extended via the `CASSINI_ROOTFS_EXTRA_SPACE` BitBake variable, to `2000000` Kilobytes, which is required by this cloud orchestration.

  ```
  kas build kas/cassini.yml:kas/kv260-psa.yml
  ```

AWS IoT Greengrass

- **Corresponding value in `DISTRO_FEATURES` variable**: `cassini-cloud`.
- **Corresponding value in `VIRTUAL-RUNTIME_cloud_service` variable**: `greengrass-cloud`.
- **Build Cloud Configs**: `kas/greengrass.yml`.

  This Cassini distribution feature includes the AWS IoT Greengrass cloud service.

  To include AWS IoT Greengrass cloud service in a Cassini distribution image, provide the **Build Cloud Config** to the kas build command. For example:

  The size of the root filesystem is extended via the `CASSINI_ROOTFS_EXTRA_SPACE` BitBake variable, to `2000000` Kilobytes, which is required by this cloud service.

  ```
  kas build kas/cassini.yml:kas/greengrass.yml:kas/kv260-psa.yml
  ```

Over-the-Air Update

This Cassini distribution feature includes over-the-air update support using mender.

`VIRTUAL-RUNTIME_ota_update` is used to select one of two options.

Mender

- **Corresponding value in `DISTRO_FEATURES` variable**: `cassini-ota`.
- **Corresponding value in `VIRTUAL-RUNTIME_ota_update` variable**: `mender-ota`.

  This Cassini distribution feature includes the mender client and relevant services by default.

No OTA

- **Corresponding value in** `DISTRO_FEATURES` **variable**: `cassini-ota`.
- **Corresponding value in** `VIRTUAL-RUNTIME_ota_update` **variable**: `no-ota`.
- **Build Cloud Configs**: `kas/no-ota.yml`.

  This Cassini distribution feature provides an image without any Over-the-air update client.

  To remove the update client in Cassini distribution image, provide the **Build OTA Update Config** to the kas build command. For example:

  ```
  kas build kas/cassini.yml:kas/no-ota.yml:kas/kv260-psa.yml
  ```

Security Service

This Cassini distribution feature configures parsec-service backend for the Cassini distribution image.

`VIRTUAL-RUNTIME_security_provider` is used to select one of three options.

PSA Provider

- **Corresponding value in** `VIRTUAL-RUNTIME_security_provider` **variable**: `psa-provider`.
- **Build Cloud Configs**: `kas/psa-security.yml`.

  This Cassini distribution feature provides an image with parsec services that uses Trusted-Services as the backend for crypto operations.

  To build a Cassini distribution image with Trusted-Services as the parsec backend, provide the **Build Security Provider Config** to the kas build command. For example:

  ```
  kas build kas/cassini.yml:kas/psa-security.yml:kas/kv260-psa.yml
  ```

SW Provider

- **Corresponding value in** `VIRTUAL-RUNTIME_security_provider` **variable**: `sw-provider`.
- **Build Cloud Configs**: `kas/sw-security.yml`.

  This Cassini distribution feature provides an image with parsec services that uses Mbed Crypto as the backend for crypto operations.

  To build a Cassini distribution image with Mbed Crypto as the parsec backend, provide the **Build Security Provider Config** to the kas build command. For example:

  ```
  kas build kas/cassini.yml:kas/sw-security.yml:kas/kv260-psa.yml
  ```

TPM Provider

- **Corresponding value in `VIRTUAL-RUNTIME_security_provider` variable**: `tpm2-provider`.
- **Build Cloud Configs**: `kas/tpm2-security.yml`.

  This Cassini distribution feature provides an image with parsec services that uses TPM as the backend for crypto operations.

  By default, Cassini build will set VIRTUAL-RUNTIME_security_provider to `tpm2-provider` if `cassini-tpm` feature is included.

  However, to force using the TPM as the parsec backend, provide the **Build Security Provider Config** to the kas build command.

  For example:

```
kas build kas/cassini.yml:kas/tpm2-security.yml:kas/kv260-psa.yml
```

Parsec service

Corresponding value in `DISTRO_FEATURES` variable: `cassini-parsec`.

This Cassini distribution feature adds parsec-service and parsec-tool to the Cassini distribution image.

The value `cassini-parsec` is appended to `DISTRO_FEATURES` in `meta-cassini-distro/conf/distro/cassini.conf`. Therefore, parsec service is included in the Cassini distribution image by default. If parsec-service is not required then the value `cassini-parsec` can be removed from `DISTRO_FEATURES` in the `<distro name>.conf` of the downstream distribution. To build Cassini distribution image with parsec-service for the KV260 hardware target platform, run the following command:

```
kas build kas/cassini.yml:kas/kv260-psa.yml
```

Security Hardening

- **Corresponding value in `DISTRO_FEATURES` variable**: `cassini-security`.

  This Cassini distribution feature configures user accounts, packages, remote access controls and other image features to provide extra security hardening for the Cassini distribution image.

  Security hardening is enabled by default in the Cassini distribution image and the base configuration appends the `cassini-security` feature to `DISTRO_FEATURES` for the build. To remove it in the Cassini distribution image, `kas/dev.yml` can be used. For example:

```
kas build kas/cassini.yml:kas/dev.yml:kas/kv260-psa.yml
```

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

TPM Support

Corresponding value in `DISTRO_FEATURES` variable: `cassini-tpm`.

This Cassini distribution feature adds tpm2 tools to the Cassini distribution image.

The value `cassini-tpm` is not appended to `DISTRO_FEATURES` by default. If TPM support is required, then the value `cassini-tpm` can be added to `DISTRO_FEATURES` in the `<distro name>.conf` of the downstream distribution. Note that the firmware must support TPM/fTPM as well.

To build Cassini distribution image with fTPM support for the KV260 hardware target platform, run the following command:

```
kas build kas/cassini.yml:kas/kv260-ftpm.yml
```

3.2.4 Additional Distribution Image Customizations

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

Filesystem Customization

Adding Extra Rootfs Space

The size of the root filesystem can be extended via the `CASSINI_ROOTFS_EXTRA_SPACE` BitBake variable. The value of this variable is appended to the `IMAGE_ROOTFS_EXTRA_SPACE` BitBake variable.

Tuning the Filesystem Compilation

The Cassini 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 `CASSINI_GENERIC_ARM64_FILESYSTEM` to `"0"`. The file system compilation tune used when `CASSINI_GENERIC_ARM64_FILESYSTEM` is enabled can be changed by setting `CASSINI_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:

```
CASSINI_GENERIC_ARM64_FILESYSTEM: "1"              # Enable generic file system␣
↪(1 or 0).
CASSINI_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:

```
CASSINI_GENERIC_ARM64_FILESYSTEM="0" kas build kas/cassini.yml:kas/kv260-psa.
↪yml
```

3.2.5 Manual BitBake Build Setup

In order to build an Cassini 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 `cassini` in `local.conf`.
- Configure the image `DISTRO_FEATURES` 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:

- `cassini-image-base`

As the kas build configuration files within the `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.

3.3 Yocto Layers

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

- `meta-cassini-bsp`

  A Yocto layer which holds board-specific recipes or append files that either:

  - will not be upstreamed (Cassini specific modifications)
  - have not been upstreamed yet

- `meta-cassini-distro`

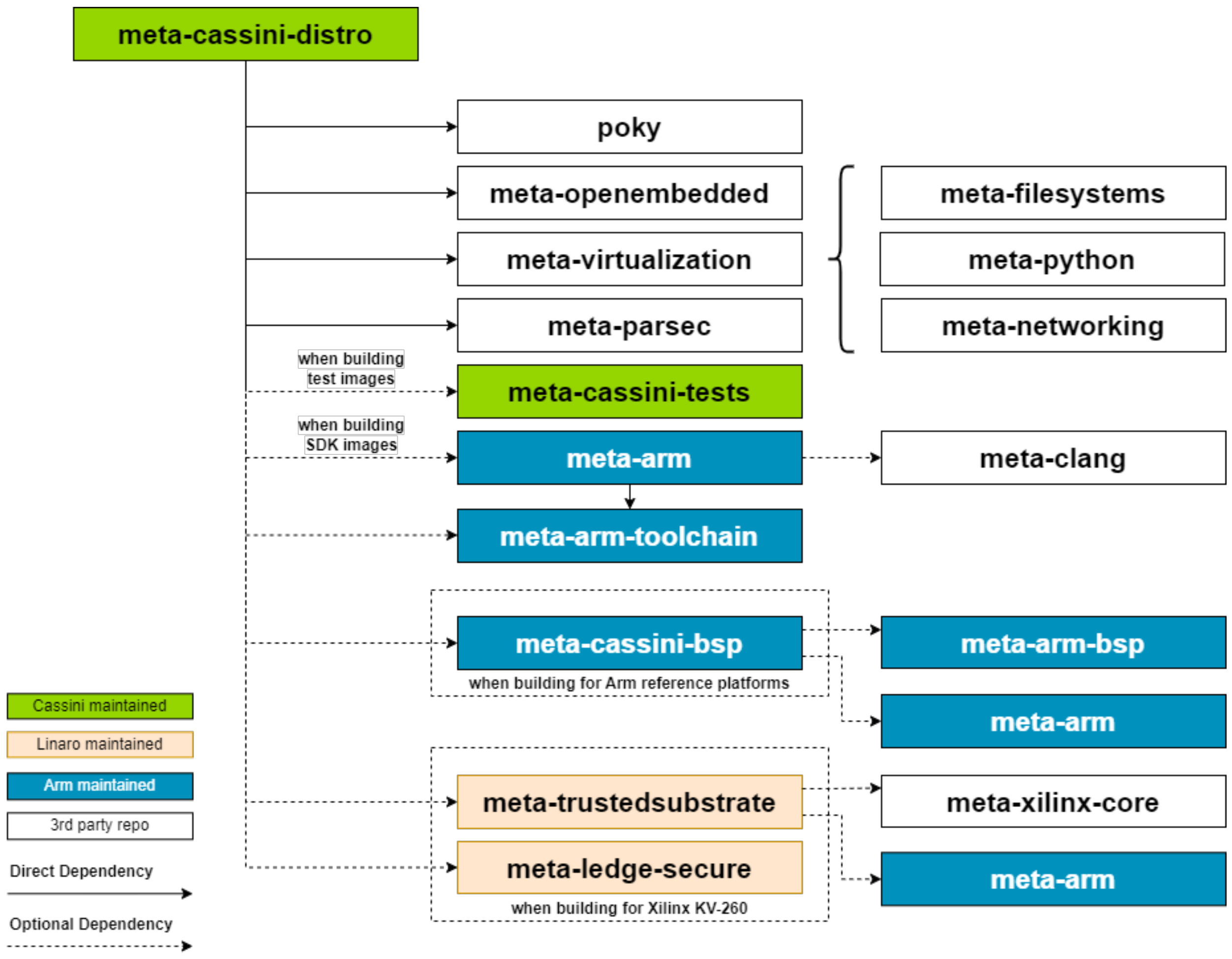
  A Yocto distribution layer providing top-level and general policies for the Cassini distribution images.

- `meta-cassini-tests`

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

3.3.1 Layer Dependency Overview

The following diagram illustrates the layers which are integrated by the Cassini project, which are further expanded on below. The layer revisions are related to the Cassini `v2.1.0` release.



Cassini distribution depends on the following layer dependency sources:

```
URL: https://git.yoctoproject.org/git/poky
layers: meta, meta-poky
branch: master
revision: ecd195a3ef96b7d1b41344e6399bfae60483a6c8

URL: https://git.openembedded.org/meta-openembedded
layers: meta-filesystems, meta-networking, meta-oe, meta-python
branch: master
revision: 5d54a52fbeb69dba7b8ae11db98af4813951fa61

URL: https://git.yoctoproject.org/git/meta-virtualization
layer: meta-virtualization
branch: master
revision: 9e9f60e959f3710fb7a16b9829d950c3d94c0d4a

URL: https://git.yoctoproject.org/git/meta-security
```

(continues on next page)

(continued from previous page)

```
layers: meta-parsec
branch: master
revision: e2c44c8b5d02591ec0be3266d6667e16725bcb92

URL: https://github.com/kraj/meta-clang
layers: meta-clang
branch: master
revision: c709a5196f1e8654425b43b478064395386c36d4
```

Additional layers are conditionally required, depending on the specific Cassini distribution image being built and the platform being targeted.

```
URL: https://gitlab.arm.com/cassini/meta-cassini-bsp
layers: meta-cassini-bsp
branch: master
revision: 33e73ab2305dcb744eb44a9d5e5e48a846476954

URL: https://git.yoctoproject.org/git/meta-arm
layers: meta-arm, meta-arm-bsp, meta-arm-toolchain
branch: master
revision: 18bc3f9389907f805b0a8ad4b6543bbdd0274d5e

URL: https://github.com/Wind-River/meta-secure-core
layers: meta, meta-efi-secure-boot, meta-signing-key
branch: master
revision: 59d7e90542947c342098863b9998693ac79352b0

URL: https://gitlab.com/Linaro/trustedsubstrate/meta-ts.git
layers: meta-trustedsubstrate
branch: master
revision: 2ab172ff5b22506f4ece8e8c5e0f7728ed8722c3

URL: https://gitlab.com/Linaro/trustedsubstrate/meta-ledge-secure.git
layers: meta-ledge-secure
branch: main
revision: b4aa684ec00652e0c42576c998184e468e55030a

URL: https://github.com/Xilinx/meta-xilinx
layers: meta-xilinx-core
branch: master
revision: 1d98f8981e3157aa265ca141a0fc9e6e2640394f

URL: https://github.com/aws/meta-aws
layers: meta-aws
branch: master
revision: cfabc95aa0f2847fa3c55f8ba3f1cd11cf8906c7

URL: https://github.com/mendersoftware/meta-mender
layers: meta-mender
branch: scarthgap
revision: 05e4f995ea3461c16b2ca9aa012b12d01d9a91e6
```

- Cassini firmware images built for the Corstone-1000 platforms requires `meta-cassini-bsp`, meta-arm, meta-arm-bsp, and meta-arm-toolchain.

- Cassini firmware images built for the Corstone-1000 also use efi-secure-boot and its dependencies to generate certificates for UEFI capsule images
- Cassini firmware images built for KV260 also require meta-arm, meta-arm-toolchain, meta-trustedsubstrate, meta-xilinx-core, and meta-ledge-secure.
- Cassini distro images built with the greengrass-cloud feature also require meta-aws and meta-multimedia.
- Cassini distro images built with the mender-ota feature also require meta-mender.

3.4 Security Hardening

Cassini distribution images can be hardened to reduce potential sources or attack vectors of security vulnerabilities. Cassini 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 and poky layers are configured with: --with-libcap[-ng].
- Remove empty-root-password allow-root-login post-install-logging from IMAGE_FEATURES
- Enable allow-empty-password to allow empty password on Cassini image boot.
- Disable all login access to the root account.

Note: When cassini-test distro feature is enabled then root login is enabled. Currently, running inline tests in LAVA require login as root to run transfer-overlay commands.

- Sets the umask to 0027 (which translates permissions as 640 for files and 750 for directories).

Security hardening is enabled by default, see *Security Hardening* for details.

Note: Cassini security hardening does not reduce the scope of the *Run-Time Integration Tests*.

3.5 Validation

3.5.1 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 Cassini 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-cassini-distro/classes/containers_kernelcfg_check.bbclass`. By default Yocto Docker config is used as the reference.

- **K3s orchestration support**:

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

3.5.2 Run-Time Integration Tests

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

The Cassini run-time integration tests are a mechanism for validating Cassini core functionalities. The following integration test suites are included in the Cassini distribution image:

- *Container Engine Tests*
- *Parsec OpenSSL Provider Tests*
- *K3s Orchestration Tests* (not supported on Corstone-1000)
- *User Accounts Tests*
- *Parsec simple end-to-end Tests*
- *Platform Security Architecture API 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 in a Cassini 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*.

Note: Container Engine and K3s Orchestration tests require access to the internet e.g. to download container images from external image hubs.

Note: When running on platforms with limited performance, the default Linux networking services may timeout before they can initialize properly. The base image provides a helper script to make sure the network is working before tests are run.

```
sudo wait-online.sh eth0
```

This step is currently necessary on Corstone-1000 platforms (FVP and MPS3).

Preparing the device

Before running the tests, the device under test should be reset to make sure no unnecessary processes are running. In addition, when using the Corstone-1000 for MPS3, the secure flash used by *Platform Security Architecture API Tests* should be wiped. The process for doing this is described here Clean Secure Flash Before Testing.

Running the Tests

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

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

If the test suite identifier (`[test-suite-id]`) is omitted, all integration tests will be run. For example, running `ptest-runner` 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
[...]
STOP: ptest-runner
```

Note: `ptest-runner -l` is a useful command to list the available test suites in the image.

Note: `[-t timeout]` specifies a timeout in seconds and must be supplied if the test takes longer than the default (**300**). You can use the duration estimates for each test to set this value.

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

Duration: *up to 30 min*

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 *Preparing the device*.

The test suite is built and installed in the image according to the following BitBake recipe: `meta-cassini-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

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*

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

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

Parsec OpenSSL Provider Tests

Duration: *up to 20 min*

The parsec openssl provider test suite is identified as:

```
parsec-openssl-provider-tests
```

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

The test suite is built and installed in the image according to the following BitBake recipe: `meta-cassini-tests/recipes-tests/runtime-integration-tests/parsec-openssl-provider-tests.bb`.

Currently the test suite contains the following tests, which run consecutively in the following order.

1. `pull image` is composed of a single sub-test:
    1.1. Pull the image via the `docker image pull` and retry thrice in case of failure
2. `run container` is composed of multiple sub-tests:
    2.1. Run a containerized detached workload via the `docker run` command to create and start a container
    2.2. Check the container is running via the `docker inspect` command
    2.3. Check parsec openssl provider connectivity to the host parsec service via `docker exec` command
3. `stop container` is composed of multiple sub-tests:
    3.1. Remove the running container via the `docker remove` command
        - Stop the container
        - Remove the container from the container list
    3.2. Check the container is not found via the `docker inspect` command

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

`` `POP_TEST_IMAGE ``: defines the container image
    Default: `registry.gitlab.com/linaro/cassini/meta-cassini/parsec-openssl-provider-image:latest`
`POP_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*
`POP_TEST_CLEAN_ENV`: enable test environment clean-up
    Default: `1` (enabled)
    See *Parsec OpenSSL Provider Environment Clean-Up*

Note: Due to performance limitations, this test takes around 4 hours on Corstone-1000 FVP and 20 hours on Corstone-1000 MPS3.

Parsec OpenSSL Provider Environment Clean-Up

A clean environment is expected when running the parsec openssl provider tests. For example, if the target image already exists within the parsec openssl provider 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 `POP_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 `POP_TEST_IMAGE`
- Removal of the image given by `POP_TEST_IMAGE`, if it exists

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

TPM OpenSSL Provider Tests

Duration: *up to 20 min*

The TPM openssl provider test suite is identified as:

```
tpm-openssl-provider-tests
```

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

The test suite is built and installed in the image according to the following BitBake recipe: `meta-cassini-tests/recipes-tests/runtime-integration-tests/tpm-openssl-provider-tests.bb`.

It's only available when *cassini-tpm* feature is enabled, and contains the following tests:

1. `Health check` is a sanity check that ensures the TPM provider is
   in the providers list.
2. `Integration Tests` are imported from tpm2-openssl tests.

The tests can be customized via environment variables passed to the execution, each prefixed by `TOP_`

`TOP_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*

`TOP_TEST_CLEAN_ENV`: enable test environment clean-up
  Default: `1` (enabled)
  See *TPM OpenSSL Provider Environment Clean-Up*

TPM OpenSSL Provider Environment Clean-Up

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

Currently, no clean-up is required as each api test cleans up temporary files before exiting.

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

K3s Orchestration Tests

Duration: *up to 10 min*

The K3s test suite is identified as:

```
k3s-integration-tests
```

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

The test suite is built and installed in the image according to the following BitBake recipe within `meta-cassini-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 web server.

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.

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*

Note: Only supported when *K3s cloud* service is selected.

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

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

User Accounts Tests

Duration: *up to 10 min*

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 *Preparing the device*.

The test suite is built and installed in the image according to the following BitBake recipe within `meta-cassini-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 with appropriate access permissions on the Cassini distribution image. The validation performed by the test suite is dependent whether or not it has been configured with *Cassini Security Hardening*.

As the configuration of user accounts is modified for Cassini distribution image which is built with Cassini security hardening, additional security-related validation is included in the test suite for this image. These additional tests validate that the appropriate password requirements and that the mask configuration for permission control of newly created files and directories is applied correctly.

The test suite therefore contains following integration tests:

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

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*

User Accounts Environment Clean-Up

As the user accounts integration tests only modify the system for images built with Cassini 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 Cassini security hardening are:

- Reset the password for the `test` user account
- Reset the password for the non-privileged Cassini user account

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.

Parsec simple end-to-end Tests

Duration: *up to 5 hours*

The Parsec simple end2end test suite is identified as:

```
parsec-simple-e2e-tests
```

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

The test suite is built and installed in the image according to the following BitBake recipe within `meta-cassini-tests/recipes-tests/runtime-integration-tests/parsec-simple-e2e-tests.bb`.

The test suite validates Parsec service in Cassini distribution image by running simple end2end tests available in parsec-tool.

The tests can be customized via environment variables passed to the execution, each prefixed by `PS_` to identify the variable as associated to the Parsec simple end2end tests:

`PS_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*

`PS_TEST_CLEAN_ENV`: enable test environment clean-up
  Default: 1 (enabled)
  See *Parsec Simple End2End Tests Environment Clean-Up*

Parsec Simple End2End Tests Environment Clean-Up

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

Currently, no clean-up is required as simple end2end tests script `parsec-cli-tests.sh` cleans up temporary files before exiting.

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

Platform Security Architecture API Tests

Duration: *up to 1 hour*

The Platform Security Architecture API test suite is identified as:

> `psa-arch-tests`

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

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

The test suite validates security requirements of PSA Certified API's Architecture on Arm-based platforms available in psa-api-tests.

The tests can be customized via environment variables passed to the execution, each prefixed by `PSA_` to identify the variable as associated to the PSA API tests:

`PSA_ARCH_TESTS_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*

`PSA_ARCH_TESTS_TEST_CLEAN_ENV`: enable test environment clean-up

> Default: 1 (enabled)
> See *Platform Security Architecture API Tests Environment Clean-Up*

Platform Security Architecture API Tests Environment Clean-Up

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

Currently, no clean-up is required as each api test cleans up temporary files before exiting.

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

3.6 Mender Validation

3.6.1 Overview

Mender is a secure and robust Open source over-the-air (OTA) update manager for Embedded Linux devices. It is used to validate end-to-end OTA functionality across Cassini platforms.

It offers:

- A/B system updates
- Capsule updates for firmware
- Robust rollback mechanisms
- Secure authentication and artifact verification
- Device grouping and phased rollout support

Duration: up to 60 minutes

This section describes the manual testing process for validating Mender client on Cassini platforms. It ensures end-to-end OTA functionality including provisioning, mender system update and mender capsule update.

The validation workflow begins with the following preparation steps:

- *Build with Mender Support*
- *Setup Mender Server*

This integration prepares the device for Mender OTA flow through:

- *DUT Provisioning*

This integration validates the Mender OTA flow through:

- *Mender System Update*
- *Mender Capsule Update*

Note: EDK2 on the KV260 platform currently **does not support capsule updates** due to limitations in the EDK2 firmware stack.

Cleans up the test environment for future test runs:

- *Stop Mender Server*

Mender environment variables:

- *BRANCH*: Specifies the Git branch tag for the Docker image used
- *CAPSULE*: Name of the capsule update image to be used.
- *HOST_MACHINE_USER* : Username of the host machine running Mender server
- *MACHINE* : Target platform machine name
- *MENDER_SERVER_DIR* : Mender server directory on the host machine
- *MENDER_DUT_TEST_DIR* : Mender test directory on DUT. It needs to be part of `/data`
- *MENDER_SERVER_HTTPS_PORT* : HTTPS port exposed to clients (default: 443)
- *MENDER_SERVER_HTTP_PORT* : HTTP port exposed to clients (default: 80)

- *MENDER_SERVER_IP* : IP address of the the host machine running Mender server
- *MENDER_SERVER_NAME* : Mender server name
- *MENDER_FW_GUID* : GUID for the ESRT entry and the capsule payload.
- *MENDER_CAPSULE_VERSION* : Version for the capsule payload. This is verified against ESRT entry version after the update.
- *MENDER_FW_VERSION* : Current version for the ESRT entry.

3.6.2 Build with Mender Support

Cassini images include Mender client support by default. For more details, see *Over-the-Air Update*. It is recommended to build with *Developer Support*.

These are the required environment variables, refer *here* for more details.

- *CAPSULE*
- *MACHINE*
- *MENDER_SERVER_DIR*

Create a clean test directory

```
rm -Rf ${MENDER_SERVER_DIR}
mkdir -p ${MENDER_SERVER_DIR}/artifacts
```

Copy the generated *.mender* artifact to a designated directory for performing system update

```
cp ${BUILDDIR}/tmp/deploy/images/${MACHINE}/cassini-image-base-${MACHINE}.mender \
  ${MENDER_SERVER_DIR}/artifacts/${MACHINE}.mender
```

Copy the generated *${CAPSULE}* artifact to a designated directory for performing capsule update

```
cp ${BUILDDIR}/tmp/deploy/images/${MACHINE}/${CAPSULE} \
  ${MENDER_SERVER_DIR}/artifacts/${MACHINE}.uefi.capsule
```

Important: The name of `${CAPSULE}` depends on the target platform used to build the capsule.

- For `corstone1000`, the capsule name is typically `${MACHINE}-v6.uefi.capsule`
- For `kv260`, the capsule name is typically `{{ MACHINE }}_fw.capsule`

3.6.3 Setup Mender Server

The Mender server is run on a host machine OS which is formed of small container services. Pull the Mender utility Docker image and launch the container on the host machine.

Note: The Mender server currently runs exclusively on x86_64 architecture, and its build process has been tested and validated on Ubuntu 22.04 LTS.

These are the required environment variables, refer *here* for more details.

- *MENDER_SERVER_DIR*

- *MACHINE*
- *BRANCH*

```
docker pull registry.gitlab.com/linaro/cassini/meta-cassini/mender-utility-image:$
↪{BRANCH}
docker run --rm \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v ${MENDER_SERVER_DIR}:${MENDER_SERVER_DIR} \
  -it registry.gitlab.com/linaro/cassini/meta-cassini/mender-utility-image:${BRANCH}
```

This opens a shell environment and these are the required environment variables, refer *here* for more details.

- *MENDER_SERVER_DIR*
- *MACHINE*
- *MENDER_SERVER_IP*
- *MENDER_SERVER_HTTPS_PORT*
- *MENDER_SERVER_HTTP_PORT*

Also, these are only required for full capsule update:

- *MENDER_FW_GUID*
- *MENDER_FW_VERSION*
- *MENDER_CAPSULE_VERSION*
- **Configure the Mender Server** :

```
export MENDER_SERVER_NAME="ms-${MACHINE}"
cd ${MENDER_SERVER_DIR}
cp -r /home/mender-server/. ${MENDER_SERVER_DIR}/
source ./mender_env_server.sh
source ./docker-compose-override.sh
```

- **Start the Mender server** :

```
docker compose --project-name "${MENDER_SERVER_NAME}" up -d --quiet-pull
docker compose --project-name "${MENDER_SERVER_NAME}" ps
```

- **User Setup** :

  Create a test user for the Mender server UI/API

  ```
  docker compose --project-name "${MENDER_SERVER_NAME}" exec useradm \
    useradm create-user --username "${MENDER_SERVER_USERNAME}" \
  --password "${MENDER_SERVER_PASSWORD}"
  ```

  Authenticate using mender-cli

  ```
  mender-cli login \
    --server "${MENDER_SERVER_URL_HTTPS_PORT}" \
    --skip-verify \
    --username "${MENDER_SERVER_USERNAME}" \
    --password "${MENDER_SERVER_PASSWORD}"
  ```

- **Artifact Upload** :

  These are the requirements to perform a **mender system update**.

  - Upload an unsigned mender artifact

```
cp ${MENDER_SERVER_DIR}/artifacts/${MACHINE}.mender ${MENDER_SERVER_DIR}/artifacts/$
↪{MACHINE}-unsigned.mender
mender-artifact modify -n unsigned-image ${MENDER_SERVER_DIR}/artifacts/${MACHINE}-
↪unsigned.mender
mender-cli artifacts upload -k --server ${MENDER_SERVER_URL_HTTPS_PORT} ${MENDER_
↪SERVER_DIR}/artifacts/${MACHINE}-unsigned.mender
```

  - Upload a signed mender artifact

```
mender-artifact modify -n release-2 -k ${MENDER_SERVER_DIR}/keys/private.key $
↪{MENDER_SERVER_DIR}/artifacts/${MACHINE}.mender
mender-cli artifacts upload -k --server ${MENDER_SERVER_URL_HTTPS_PORT} ${MENDER_
↪SERVER_DIR}/artifacts/${MACHINE}.mender
```

  These are the requirements to perform a **mender capsule update**.

  - Upload a compatible mender capsule

```
# Set GUID and versions
export MENDER_FW_GUID=<guid>
export MENDER_FW_VERSION=<old-version>
export MENDER_CAPSULE_VERSION=<new-version>

mender-artifact write module-image \
  -T uefi-capsule \
  -n compatible-capsule-update \
  -o ${MENDER_SERVER_DIR}/artifacts/${MACHINE}-compatible-capsule-update.mender \
  -f ${MENDER_SERVER_DIR}/artifacts/${MACHINE}.uefi.capsule \
  -t ${MACHINE} \
  -k ${MENDER_SERVER_DIR}/keys/private.key \
  --provides "uefi-firmware.${MENDER_FW_GUID}.version:${MENDER_CAPSULE_VERSION}" \
  --depends "uefi-firmware.${MENDER_FW_GUID}.version:${MENDER_FW_VERSION}"

mender-cli artifacts upload \
  -k --server ${MENDER_SERVER_URL_HTTPS_PORT} \
  "${MENDER_SERVER_DIR}/artifacts/${MACHINE}-compatible-capsule-update.mender"
```

  These are the requirements to perform a **mender rollback capsule update**.

  - Use capsule-tool.py to generate a tampered capsule

```
${MENDER_SERVER_DIR}/systemready-scripts/capsule-tool.py --tamper \
  --output ${MENDER_SERVER_DIR}/artifacts/tampered-${MACHINE}.uefi.capsule \
  ${MENDER_SERVER_DIR}/artifacts/${MACHINE}.uefi.capsule
```

  - Upload a compatible rollback mender capsule

```
# Set GUID and versions
export MENDER_FW_GUID=<guid>
export MENDER_FW_VERSION=<old-version>
```

(continues on next page)

(continued from previous page)

```
export MENDER_CAPSULE_VERSION=<new-version>

mender-artifact write module-image \
  -T uefi-capsule \
  -n compatible-rollback-capsule-update \
  -o ${MENDER_SERVER_DIR}/artifacts/${MACHINE}-compatible-rollback-capsule-update.
↪mender \
  -f ${MENDER_SERVER_DIR}/artifacts/tampered-${MACHINE}.uefi.capsule \
  -t ${MACHINE} \
  -k ${MENDER_SERVER_DIR}/keys/private.key \
  --provides "uefi-firmware.${MENDER_FW_GUID}.version:${MENDER_CAPSULE_VERSION}" \
  --depends "uefi-firmware.${MENDER_FW_GUID}.version:${MENDER_FW_VERSION}"

mender-cli artifacts upload \
  -k --server ${MENDER_SERVER_URL_HTTPS_PORT} \
  "${MENDER_SERVER_DIR}/artifacts/${MACHINE}-compatible-rollback-capsule-update.
↪mender"
```

To check the uploaded artifacts

```
mender-cli artifacts list -k --server ${MENDER_SERVER_URL_HTTPS_PORT}
```

3.6.4 DUT Provisioning

This step is performed on the Device Under Test (DUT) to enable Mender client-server communication.

Requirements:

- *Running Mender server*

These are the required environment variables, refer *here* for more details.

- *MENDER_DUT_TEST_DIR*
- *HOST_MACHINE_USER*
- *MENDER_SERVER_DIR*
- *MENDER_SERVER_NAME*
- *MENDER_SERVER_IP*
- *MENDER_SERVER_HTTPS_PORT*
- *MENDER_SERVER_HTTP_PORT*

Login and switch to root user

```
sudo su -
```

Create a working directory for Mender testing

```
mkdir -p ${MENDER_DUT_TEST_DIR}
chown -R cassini:cassini ${MENDER_DUT_TEST_DIR}
cd ${MENDER_DUT_TEST_DIR}
```

Copy Mender helper scripts and certificates from the host machine to the DUT

```
scp -o "StrictHostKeyChecking no" ${HOST_MACHINE_USER}@${MENDER_SERVER_IP}:${MENDER_
↪SERVER_DIR}/mender_*.sh ${MENDER_DUT_TEST_DIR}
scp -o "StrictHostKeyChecking no" ${HOST_MACHINE_USER}@${MENDER_SERVER_IP}:${MENDER_
↪SERVER_DIR}/compose/certs/mender.crt ${MENDER_DUT_TEST_DIR}
```

Note: This step is required for `corstone1000-fvp` machine to prevent unexpected shutdown during the preparation of mender provisioning.

```
timedatectl
sudo systemctl restart systemd-timesyncd.service
timedatectl
sudo ip link set eth0 down
```

Place the Mender server certificate in the correct location and adjust the permissions. Ensure the helper scripts are executable.

```
cp ${MENDER_DUT_TEST_DIR}/mender.crt /etc/mender
chmod 644 /etc/mender/mender.crt
chmod +x ${MENDER_DUT_TEST_DIR}/mender_*.sh
```

Enable helper functions to perform mender testing.

```
source ${MENDER_DUT_TEST_DIR}/mender_test_helper.sh
```

Restart Mender-related services to apply changes.

```
systemctl restart mender-authd
systemctl restart mender-updated
```

Note: This step is required for `corstone1000-fvp` machine to prevent unexpected shutdown during the preparation of mender provisioning.

```
sudo ip link set eth0 up
```

Run provisioning script to register DUT with the Mender server.

```
provision_device
```

3.6.5 Mender System Update

Once the Mender server is running and the Device Under Test (DUT) is provisioned, the next step is to deliver your update payloads to the server for performing system update on DUT.

Requirements:

- *Mender artifact*
- *Artifacts are uploaded*
- *Running Mender server*
- *Device Under Test (DUT) is provisioned*

This section covers two different scenarios.

- Signed artifacts for secure deployments.
- Unsigned artifact to demonstrate failure deployments.
- **Setup** :

  Copy artifact key from the host machine to the DUT.

  ```
  scp -o "StrictHostKeyChecking no" ${HOST_MACHINE_USER}@${MENDER_SERVER_IP}:${MENDER_
  ↪SERVER_DIR}/keys/artifact-verify-key.pem ${MENDER_DUT_TEST_DIR}
  ```

  Install the artifact key to validate the incoming mender artifacts which are deployed from the Mender server.

  ```
  cp ./artifact-verify-key.pem /etc/mender
  ```

- **Deploying Updates** :

  To trigger an **unsigned system update**, we deploy an unsigned artifact.

  ```
  create_artifact_deployment unsigned-image
  check_for_unsigned_update
  ```

  Expected outcome:

  – The device rejects the artifact due to missing or invalid signature.

  – This also validates proper enforcement of security policies.

  To trigger a **signed system update**, we deploy a signed artifact.

  ```
  create_artifact_deployment release-2
  wait_for_signed_update
  ```

  Expected outcome:

  – The artifact passes signature validation.

  – The Mender client downloads and installs the update.

  – The device reboots upon successful deployment.

  Note: Please allow up to 30 minutes for the DUT to automatically reboot after running these commands.

- **Final Verification** :

  Final check to confirm that the **signed system update** was successfully completed and acknowledged by DUT to the Mender server.

  This step ensures the DUT, is able to **securely communicate** with the correct Mender server that was used during the provisioning and deployment steps.

  Once the DUT has rebooted after applying the mender system update login and switch to root user:

  ```
  sudo su -
  ```

  These are the required environment variables, refer *here* for more details.

  – *MENDER_DUT_TEST_DIR*

  – *MENDER_SERVER_HTTPS_PORT*

  - *MENDER_SERVER_HTTP_PORT*
  - *MENDER_SERVER_IP*
  - *MENDER_SERVER_NAME*

```
cd  ${MENDER_DUT_TEST_DIR}
cp  ${MENDER_DUT_TEST_DIR}/artifact-verify-key.pem /etc/mender
cp  ${MENDER_DUT_TEST_DIR}/mender.crt /etc/mender
chmod 644 /etc/mender/mender.crt
```

Note: This step is required for `corstone1000-fvp` machine to prevent unexpected shutdown during the validation of mender system update.

```
sudo ip link set eth0 down
```

Setup environment variables and load helper functions used for final validation.

```
source ./mender_test_helper.sh
```

Restart Mender authentication service to reinitialize secure communication.

```
systemctl restart mender-authd
```

Note: This step is required for `corstone1000-fvp` machine to prevent unexpected shutdown during the validation of mender system update.

```
sudo ip link set eth0 up
```

Finally, check to confirm the signed update was applied successfully.

```
check_for_signed_update
```

3.6.6 Mender Capsule Update

Note: Before executing mender capsule updates, ensure that the firmware is capable of handling UEFI capsules.

Additionally, `BootOrder` needs to be set once to prioritize booting from EFI/UpdateCapsule before any updates.

For example, via these steps in U-Boot shell:

```
# Add a new boot option: Boot1001 that boots from
# EFI/UpdateCapsule
# mmc <device-index>:<partition> should be the ESP partition
# where EFI/UpdateCapsule is located
efidebug boot add -b 1001 cap mmc <device-index>:<partition> EFI/UpdateCapsule

# Prepend the new boot option to the current order
# <existing_boot_options> can be checked via
# `efidebug boot order` command without any arguments.
efidebug boot order 1001 <existing_boot_options>
```

Once the Mender server is running and the Device Under Test (DUT) is provisioned, the next step is to deliver your update payloads to the server for performing capsule update on DUT

Requirements:

- *Mender artifact*
- *Artifacts are uploaded*
- *Running Mender server*
- *Device Under Test (DUT) is provisioned*
- *Copy artifact key on DUT*

This section covers only the compatible capsule scenario.

- **Update module**

  The uefi-capsule update module is part of Cassini image by default.

- **Deploying Updates** :

  To trigger an **compatible rollback capsule update**, we deploy a compatible-rollback-capsule-update.

  ```
  create_artifact_deployment "compatible-rollback-capsule-update"
  ```

  Expected outcome:

  – The mender capsule passes signature validation.
  – The DUT downloads the artifact by checking the compatibility.
  – The update module will install the capsule to `/boot/efi/EFI/UpdateCapsule` directory.
  – The DUT will be rebooted automatically.
  – The firmware will handle capsule update automatically after *this modification*.
  – The firmware detects that the capsule is tampered and rejects the update.
  – After booting into Linux, the update module will verify new ESRT table against the received capsule information.
  – But ESRT entries do not match the expected versions, causing the update to fail.
  – Mender triggers an automatic rollback and the DUT will be rebooted automatically.
  – On the next boot, the update module runs rollback verification by comparing current ESRT entries with the previous versions.
  – Upon success, these logs will be visible in `journalctl -u mender-updated`:

    ```
    Rollback succeeded for <guid>
    Rollback succeeded for all firmware entries
    ```

  To trigger an **compatible capsule update**, we deploy a compatible-capsule-update.

  ```
  create_artifact_deployment "compatible-capsule-update"
  ```

  Expected outcome:

  – The mender capsule passes signature validation.
  – The DUT downloads the artifact by checking the compatibility.

- The update module will install the capsule to `/boot/efi/EFI/UpdateCapsule` directory.
- The DUT will be rebooted automatically.
- The firmware will handle capsule update automatically after *this modification*.
- After booting into Linux, the update module will verify new ESRT table against the received capsule information.
- Upon success, these logs will be visible in `journalctl -u mender-updated`:

  ```
  Update succeeded for <guid>
  Update succeeded for all capsule payloads
  ```

3.6.7 Stop Mender Server

These are the required environment variables, refer *here* for more details.

- *MENDER_SERVER_NAME*
- *MENDER_SERVER_DIR*

Tear down any previous Mender server containers and remove mender-server directory.

```
docker compose --project-name "${MENDER_SERVER_NAME}" down -v --remove-orphans
rm -rf ${MENDER_SERVER_DIR}
```

3.7 Building the documentation

The Cassini project is currently configured so that *Read the Docs* builds the project documentation using Ubuntu 22.04 and Python 3.10.

The sources for the documentation are found in the `documentation` folder. To setup the host to build the documentation locally, install the required packages on the host as follows:

```
python3 -m pip install -U -r documentation/requirements.txt
```

To build and generate the documentation in html format, run:

```
sphinx-build -b html -a -W documentation public
```

To render and explore the documentation, simply open *meta-cassini/public/index.html* in a web browser.

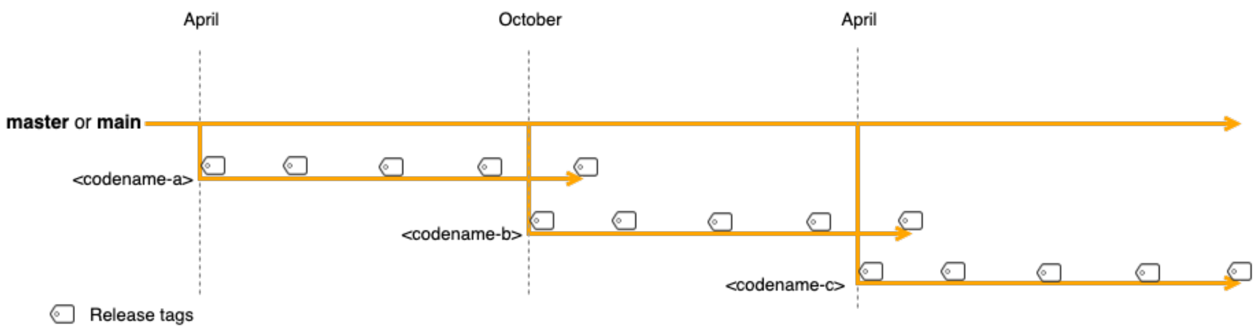
CHAPTER

FOUR

CODELINE MANAGEMENT

The Cassini project is developed and released 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.

4.1 Yocto Release Process Overview

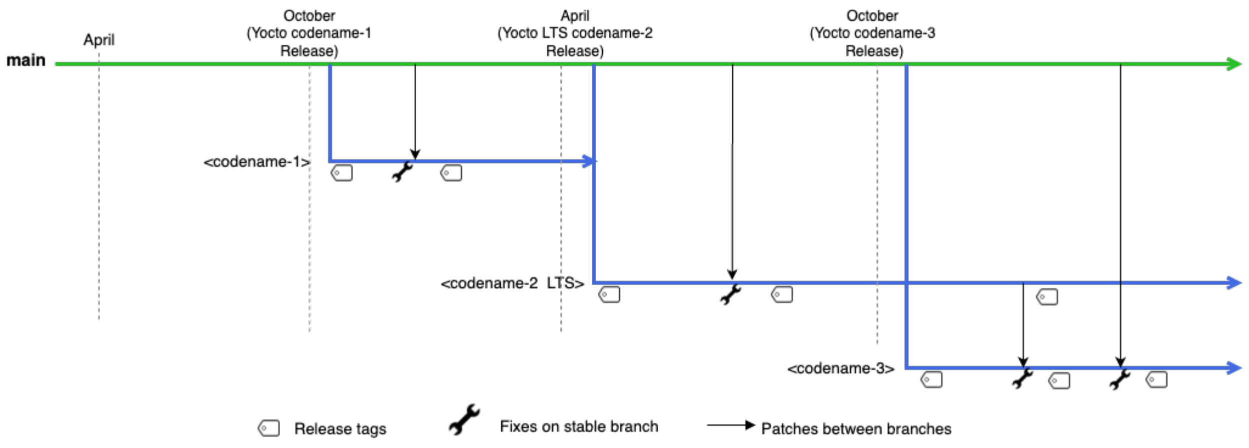


The diagram above gives an overview of the Yocto branch and release process:

- Development happens primarily in the main (or master) 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, scarthgap).
- 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.

4.2 Cassini Branch and Release Process



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

4.2.1 Cassini 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-cassini` is actively developed on this `main` branch.

4.2.2 Cassini release branches

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

4.2.3 Cassini release tags

- Cassini is tagged using the version format `v<Major>.<Minor>.<Patch>`.
- Tags are always applied to commits from the release branch.
- The *Major* version is incremented for LTS Yocto releases.
- The *Minor* version is incremented for stable Yocto releases that is not LTS.
- *Patch* releases are mainly used for hot fixes, which are first merged into the main branch and may then be back-ported to stable or LTS branches.
- 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.

CHAPTER

FIVE

CONTRIBUTING

We welcome contribution from everyone via the `meta-cassini` public Gitlab repository: https://gitlab.com/Linaro/cassini/meta-cassini. For general introduction about Cassini distribution, refer to *Introduction*.

5.1 License

The Cassini distribution is released under *License*.

Please use an SPDX license identifier in every source file following the recommendations to make it easier for users to understand and review licenses.

```
/*
* SPDX-License-Identifier: MIT
*/
```

5.2 Contributing to Cassini distribution

This project uses the GitLab project forking workflow.

Every commit must have at least one `Signed-off-by:` line from the committer to certify that the contribution is made under the terms of the `Developer's Certificate of Origin`.

The full text of `Developer's Certificate of Origin` can be found in sign-your-work-the-developer-s-certificate-of-origin. Due to the significance of the `Developer's Certificate of Origin`, part of it is copied below.

```
The sign-off is a simple line at the end of the explanation for the
patch, which certifies that you wrote it or otherwise have the right to
pass it on as an open-source patch.  The rules are pretty simple: if you
can certify the below:

Developer's Certificate of Origin 1.1
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

By making a contribution to this project, I certify that:

    (a) The contribution was created in whole or in part by me and I
        have the right to submit it under the open source license
        indicated in the file; or
```

(continues on next page)

(continued from previous page)

```
    (b) The contribution is based upon previous work that, to the best
        of my knowledge, is covered under an appropriate open source
        license and I have the right under that license to submit that
        work with modifications, whether created in whole or in part
        by me, under the same open source license (unless I am
        permitted to submit under a different license), as indicated
        in the file; or

    (c) The contribution was provided directly to me by some other
        person who certified (a), (b) or (c) and I have not modified
        it.

    (d) I understand and agree that this project and the contribution
        are public and that a record of the contribution (including all
        personal information I submit with it, including my sign-off) is
        maintained indefinitely and may be redistributed consistent with
        this project or the open source license(s) involved.

then you just add a line saying::

    Signed-off-by: Random J Developer <random@developer.example.org>

using your real name (sorry, no pseudonyms or anonymous contributions.)
```

5.3 Commit guidelines

Commits and patches added should follow the OpenEmbedded patch guidelines with the following additions.

The component being changed in the shortlog should be prefixed with the layer name (without meta-), for example:

> cassini-config: Decrease frobbing level
>
> cassini-distro: Enable foobar v2
>
> cassini-doc: Added foobar v2 documentation

While specific to the Linux kernel, refer also to the Linux kernel patch guidance. In the above, pay particular attention to the guidance on how to make your changes easy to review.

5.3.1 Describe your changes

Describe the problem you are fixing or the feature you are adding. The commits themselves show **how** the code is being changed so the commit messages should explain to the reviewer (in plain English) **what** is being changed and **why**.

5.3.2 Separate your changes

Separate each logical change into a separate commit. Each commit should implement a single, cohesive idea which should be justifiable on its own merits. Separate complex commits by dividing large problems or features into smaller ideas which can be applied one at a time. A commit which makes similar changes to multiple files should be separated from a commit which makes an unrelated change to a single file.

5.3.3 Commit messages guidelines

Commit messages should follow the guidelines below, for reasons explained by Chris Beams in How to Write a Git Commit Message:

- The commit subject and body must be separated by a blank line.
- The commit subject must start with a capital letter.
- The commit subject must not be longer than 72 characters.
- The commit subject must not end with a period.
- The commit body must not contain more than 72 characters per line.
- Commits that change 30 or more lines across at least 3 files should describe these changes in the commit body.
- Use issues and merge requests' full URLs instead of short references, as they are displayed as plain text outside of GitLab.
- The merge request should not contain more than 10 commit messages.
- The commit subject should contain at least 3 words.

Important notes:

- If the guidelines are not met, the MR may not pass the Danger checks.
- Consider enabling Squash and merge if your merge request includes "Applied suggestion to X files" commits, so that Danger can ignore those.
- The prefixes in the form of *[prefix]* and *prefix:* are allowed (they can be all lowercase, as long as the message itself is capitalized). For instance, *danger: Improve Danger behavior* and *[API] Improve the labels endpoint* are valid commit messages.

Why these standards matter

1. Consistent commit messages that follow these guidelines make the history more readable.
2. Concise standard commit messages helps to identify breaking changes for a deployment or ~"main:broken" quicker when reviewing commits between two points in time.

Commit message template

Example commit message template that can be used on your machine that embodies the above (guide for how to apply template):

```
# (If applied, this commit will...) <subject>        (Max 72 characters)
# |<----          Using a Maximum Of 72 Characters                ---->|


# Explain why this change is being made
# |<----   Try To Limit Each Line to a Maximum Of 72 Characters   ---->|

# Provide links or keys to any relevant tickets, articles or other resources
# Use issues and merge requests' full URLs instead of short references,
# as they are displayed as plain text outside of GitLab

# --- COMMIT END ---
# --------------------
# Remember to
#    Capitalize the subject line
#    Use the imperative mood in the subject line
#    Do not end the subject line with a period
#    Subject must contain at least 3 words
#    Separate subject from body with a blank line
#    Commits that change 30 or more lines across at least 3 files should
#    describe these changes in the commit body
#    Use the body to explain what and why vs. how
#    Can use multiple lines with "-" for bullet points in body
#    For more information: https://cbea.ms/git-commit/
# --------------------
```

5.4 Changelog entries

This section contains instructions for when and how to generate a changelog entry file, as well as information and history about our changelog process.

5.4.1 Overview

Each bullet point, or **entry**, in our CHANGELOG.md file is generated from the subject line of a Git commit. Commits are included when they contain the Changelog Git trailer. When generating the changelog, author and merge request details are added automatically.

The Changelog trailer accepts the following values:

- feature: New feature added/enabled
- bug: Bug fix
- deprecated: New deprecation
- removed: Feature removal
- security: Security fix
- performance: Performance improvement

- other: Other

An example of a Git commit to include in the changelog is the following:

```
Update git vendor to gitlab

Now that we are using gitaly to compile git, the git version isn't known
from the manifest, instead we are getting the gitaly version. Update our
vendor field to be `gitlab` to avoid cve matching old versions.

Changelog: changed
MR: https://gitlab.com/foo/bar/-/merge_requests/123
```

Overriding the associated merge request

GitLab automatically links the merge request to the commit when generating the changelog. If you want to override the merge request to link to, you can specify an alternative merge request using the MR trailer:

```
Update git vendor to gitlab

Now that we are using gitaly to compile git, the git version isn't known
from the manifest, instead we are getting the gitaly version. Update our
vendor field to be `gitlab` to avoid cve matching old versions.

Changelog: changed
MR: https://gitlab.com/foo/bar/-/merge_requests/123
```

The value must be the full URL of the merge request.

5.4.2 What warrants a changelog entry?

- Security fixes **must** have a changelog entry, with `Changelog` trailer set to `security`.
- Any user-facing change **must** have a changelog entry. Example: "meta-cassini now supports AWS Greengrass as a cloud option"
- A fix for a regression introduced and then fixed in the same release (such as fixing a bug introduced during a release candidate) **should not** have a changelog entry.
- Any developer-facing change (such as refactoring, technical debt remediation, or test suite changes) **should not** have a changelog entry.
- Any contribution from a community member, no matter how small, **may** have a changelog entry regardless of these guidelines if the contributor wants one.
- Any experimental changes **should not** have a changelog entry.
- An MR that includes only documentation changes **should not** have a changelog entry.

5.4.3 Writing good changelog entries

A good changelog entry should be descriptive and concise. It should explain the change to a reader who has *zero context* about the change. If you have trouble making it both concise and descriptive, err on the side of descriptive.

- **Bad:** Use newest version.
- **Good:** Updated to latest U-Boot version to get FF-A support.

The first example provides no context of where the change was made, or why, or how it benefits the user.

- **Bad:** Update syntax.
- **Good:** Update bitbake files to new append syntax to allow use with > hardknott yocto versions.

Again, the first example is too vague and provides no context.

- **Bad:** Fixes and Improves , usage in config files.
- **Good:** Fix parsec config file so that parsec can encrypt large payloads from clients.

The first example is too focused on implementation details. The user doesn't care that we changed comma's they care about the *end result* of those changes.

- **Bad:** Extended parsec input buffer for encrypt operations
- **Good:** Allow parsec to encrypt message up to 512Kb in size when using incremental encryption API's

The first example focuses on *how* we fixed something, not on *what* it fixes. The rewritten version clearly describes the *end benefit* to the user (larger possible data sets), and *when* (calling the incremental encryption API's).

Use your best judgement and try to put yourself in the mindset of someone reading the compiled changelog. Does this entry add value? Does it offer context about *where* and *why* the change was made?

5.4.4 How to generate a changelog entry

Git trailers are added when committing your changes. This can be done using your text editor of choice. Adding the trailer to an existing commit requires either amending to the commit (if it's the most recent one), or an interactive rebase using `git rebase -i`.

To update the last commit, run the following:

```
git commit --amend
```

You can then add the Changelog trailer to the commit message. If you had already pushed prior commits to your remote branch, you have to force push the new commit:

```
git push -f origin your-branch-name
```

To edit older (or multiple commits), use `git rebase -i HEAD~N` where `N` is the last N number of commits to rebase. Let's say you have 3 commits on your branch: A, B, and C. If you want to update commit B, you need to run:

```
git rebase -i HEAD~2
```

This starts an interactive rebase session for the last two commits. When started, Git presents you with a text editor with contents along the lines of the following:

```
pick B Subject of commit B
pick C Subject of commit C
```

To update commit B, change the word pick to reword, then save and quit the editor. Once closed, Git presents you with a new text editor instance to edit the commit message of commit B. Add the trailer, then save and quit the editor. If all went well, commit B is now updated.

For more information about interactive rebases, take a look at the Git documentation.

5.5 Submitting changes

Thank you for your interest in contributing to Cassini distribution. To contribute, follow the *instructions* and ensure you adhere to *commit guidelines*.

5.6 Merge criteria

- The merge request must receive at least 2 approvals from *Cassini distro maintainers*
- `meta-cassini` pipelines are passed
- No regression on code coverage

CHAPTER

SIX

CONTINUOUS INTEGRATION AND DEVELOPMENT (CI/CD)

6.1 Introduction

Project Cassini defines a GitLab CI/CD pipeline to help developers and reviewers by detecting issues at an early stage when a merge request is created.

To create a merge request on this project the user needs to be a member of the project and refer to *Contributing* for further details.

6.1.1 Overview

The following diagram illustrates brief overview of Project Cassini automated testing.

- **External Tools**:

  These tools are used by jobs in the pipeline and built in Docker containers ensuring running them in confined environment.

  – **Packages**:

    These are Debian packages provided from ubuntu.com

  – **jfrog**:

    The jfrog tool is provided from jfrog.io to communicate with *artifactory*.

  – **SCT parser**:

    The SCT parser is used to parse test results.

- **External Images**:

  External images used directly by the pipeline are pulled from the following sources:

- **Base Image**:

  The Docker Hub is a service provided by Docker for finding and sharing container images. The images are some of the most secure images on Docker Hub and official released versions which can be used as a base image for usage.

- **gitlab.com**:

  These are prebuilt images provided by gitlab.com to include jobs like static analysis.

- **Quay**:

  A buildah image is provided by Red Hat (via Quay) to build custom container images. These images, which are used in later stages of the pipeline, are stored in the GitLab Registry.

• **GitLab CI/CD**:

CI/CD is a GitLab feature which validates, request infrastructure for, and runs a GitLab pipeline using Container Registry.

- **GitLab pipelines**:

  See Gitlab pipelines

- **Container Registry**:

  This stores custom container images which are used by some pipeline jobs.

6.1.2 Architecture

The following diagram illustrates various components involved when the user raises a merge request for changes to be accepted for Project Cassini.

The different sections are described below:

• **Pipeline configuration**:

This consists of further components:

- **Dangerfile**:

  Configures and includes a danger-review job that is used to perform a variety of automated checks on the code under test.

  Danger is a Ruby Gem that runs in the CI environment. It only posts one comment and updates its content on subsequent danger-review runs.

- **.gitlab-ci.yml**:

  The meta-cassini repository defines the configuration of pipeline, when user raises merge request for changes to be accepted.

   - **Gitlab Templates**:

     These provide templates when included for the project to enable/configure tools and functionalities for the pipelines which can be referred in *GitLab Templates*.

   - **.pre-commit-config.yaml**:

     Configures pre-commit hooks that invokes a program that parses a config file and analyzes files, potentially producing formatted output representing issues in those files.

     Please refer related documentation in *Code Quality* .

- **GitLab CI/CD**:

  A brief overview can be found in *GitLab pipeline*.

  This consists of further components:

   - **GitLab Runners**:

     GitLab Runner is an application that works with *GitLab CI/CD* to run jobs in a pipeline.

   - **Container Registry**:

     The Container Registry holds the custom images required for specific pipeline jobs, which includes:

      ∗ DangerBot Image

      ∗ Code Climate plugins

      ∗ FVP Image

      ∗ IDT (IoT Device Tester) Greengrass Image

      ∗ Parsec OpenSSL Image

      ∗ Mender Utility Image

      ∗ LAVA Test Image

      ∗ Utility Image

- **Testing**:

  This consists of further components:

   - **LAVA test framework**:

     The LAVA provides the capabilities to test the built images for Cassini distribution on supported target platforms.

- **Artifactory**:

  This provides a database to store results from the LAVA test framework.

6.2 GitLab Templates

The GitLab Templates project defines common CI/CD configuration elements which can be included in other GitLab projects. The components used in the Cassini CI/CD pipelines are:

- **Changelog**:

  This updates the CHANGELOG.md file when a merge request modifies CASSINI_VERSION in cassini-release.yml. This will cause a new commit added to the MR and cancel/re-trigger the pipeline.

- **Child pipelines**:

  This creates the merge, trigger, and collate-results jobs for the child pipeline.

- **Workflow**:

  This defines some common rules to control when CI pipelines should and shouldn't run. For example, don't run the pipeline for pushes to a branch when there is already an MR open for that branch

- **Danger review**:

  This job runs the danger-bot on merge request pipelines to report issues early.

- **Docker images**:

  Set up Docker images with a predefined set of configurations to be used by the pipeline using *buildah*.

- **Static analyzer**:

  Sets up Sast (Security analyzer), and Code quality (Code quality analyzer) and generates Code quality HTML reports using pre-commit hooks.

- **Lava testing**:

  Setup to submit jobs to LAVA Test framework and retrieve results when they are complete.

- **Auto release**:

  These jobs create a GitLab release and attach release notes based on changes to the Changelog. The release note is generated from the git commit history and requires knowledge of the current package version number.

- **Kas setup**:

  Set up the required configurations for building a Yocto-based project using *kas*.

6.3 Code Quality

The CI/CD pipeline uses GitLab's Code Quality feature to perform static analysis of code, scripts, and documentation.

6.3.1 Usage

Code quality tools are configured as hooks in .pre-commit-config.yaml, sourced from official repositories. See the pre-commit and pre-commit hooks references for details on available hook types and usage.

6.3.2 Hooks

- **cspell**:

  Runs the cspell spelling checker over files in a project.

- **oelint-adv**:

  This hook runs an opinionated linter (see oelint-adv) over bitbake recipes and checks them for conformance to the OpenEmbedded style guide.

- **shell-check**:

  Gives warnings and suggestions for bash/sh shell scripts (see shell-check).

- **flake8**:

  Runs Python code style and quality checks, including PEP 8 compliance, import ordering, and other configurable rules. Provides clear feedback on style violations, unused variables, and potential code issues (see flake8).

- **yamllint**:

  Validates the structure and syntax of yaml/yml files (see yamllint).

6.4 GitLab pipeline

A brief overview of GitLab pipeline

6.4.1 Parent pipeline

This pipeline is responsible for setup, configuring tools in the Docker images, and performing static analysis (Code Quality, Danger-Review, secret detection, and container scanning).

- **.pre**:
  - Build the Docker images required for the project (For example debian-buster OS) for different architectures and push them to the registry. This involves IDT Greengrass, LAVA and Utility Docker (arm64, x86_64) images.
  - Pull the generated Docker images and config and create a manifest for LAVA and Utility usage.
  - Run danger-bot for reviewing and report issues early.
  - Regenerate the changelog when the project version number is changed.
- **Setup**:
  - Setup stage to ensure all the configurations have a valid yaml file.
  - Merge all the jobs into one file which defines each stage of the child pipeline.
- **Build**:
  - Build the documentation.
  - Create a child pipeline.
- **Test**:
  - Collate-results from *child pipeline*.
  - Detect any secrets present in the codebase.
  - Test the code quality using pre-commit hooks.
  - Generate code quality report using pre-commit hooks.

  - Run Gitlab semgrep analyzer.

- **Release**:

  - Creating the release and notes.

6.4.2 Child pipeline

This pipeline is responsible to build cassini distro images, setup, submit and report back results from the LAVA test framework perform.

- **Setup**:

  - Update external repositories that are required for Cassini distribution.
  - Extract the required FVP version details from the codebase.

- **Build**:

  - Image build for all supported platforms depending on rules or changes to the codebase.
  - Check if Cassini distro is compatible with layers definitions, this is based on each platform and multiple layers included in the distro.

- **Test**:

  - Install the required FVP in a Docker image.
  - Prepare the built Cassini distro images for LAVA test framework.
  - Submit jobs to the LAVA test framework.
  - Wait for event from LAVA test framework for completion and return to *parent pipeline*

- **Cleanup**:

  - Clean the sstate cache and download directory which is older than specific number of days.

6.5 Amazon Web Services (AWS) IoT Device Tester (IDT)

AWS IoT Device Tester (IDT) is a downloadable testing framework that helps us validate IoT devices, see AWS IoT Device Tester for Greengrass V2.

The IDT is installed with tools and configured in a Docker container image with required credentials of AWS account for testing Greengrass on a device.

These credentials are required by `aws-cli` to perform necessary setup for Device Under Test (DUT) and needs to be configured as GitLab variables to be used by GitLab pipelines:

- *AWS_ACCESS_KEY_ID*

- *AWS_SECRET_ACCESS_KEY*
- *AWS_DEFAULT_REGION*

The Docker container provisions the Device to work with AWS and IDT and generates a config file, which is used on the DUT to restart the Greengrass service. Then IDT runs the configured required tests and uploads to artifactory (optional).

6.5.1 Overview

A brief overview of IDT setup and running it on Cassini GitLab CI/CD.

The IoT Device Tester is setup as follows:

- **IDT Setup** :

  The `idt_setup.sh` should be executed manually **once** to setup the **project** to use IoT Device Tester with AWS. It creates the required AWS Identity and Access Management (IAM) role, Internet of Things (IoT) Token exchange role, IoT role alias, IoT Thing Group and policies for IDT. The following parameters needs to be configured:

  - *GG_HOME* : Home directory of Greengrass service
  - *IDT_ROLE* : IAM role
  - *IDT_ROLE_POLICY* : IAM role policy
  - *IDT_ROLE_SESSION_DURATION* : IAM role session timeout duration
  - *IOT_TE_ROLE* : IoT Token exchange role
  - *IOT_TE_ROLE_POLICY* : IoT Token exchange policy
  - *IOT_TE_ROLE_ALIAS* : IoT Token exchange role alias
  - *IOT_TE_ROLE_POLICY_ALIAS* : IoT Token exchange policy alias
  - *IOT_THING_POLICY* : IoT thing policy
  - *IOT_THING_GROUP* : IoT thing group
  - *AWS_BOUNDARY_POLICY* : Boundary policy for IAM or IoT role (optional).

The IoT Device Tester is run in LAVA with the following steps:

- **Provision DUT**:

  This step is executed on IDT Docker container image for every DUT with unique `IOT_THING_NAME`. Depending on the available Greengrass plugin, a tool is used to generate the signed RSA key and a certificate to work with AWS for IDT on DUT as follows:

  1. parsec-tool should be used in case of Parsec plugin.
  2. tpm2_ptool should be used in case of Pkcs11 plugin (*cassini-tpm* feature is enabled). The tpm2-pkcs11 store directory is /opt/tpm2-pkcs11/ by default in Cassini builds.

Further, these parameters needs to be configured as GitLab variables to be used by GitLab pipelines when *setup* was performed:

- *GG_HOME*
- *IOT_THING_GROUP*
- *IOT_THING_POLICY*
- *IOT_TE_ROLE_POLICY_ALIAS*
- *IOT_TE_ROLE_ALIAS*

The provisioning script and all the following scripts need an extra environment variable:

- *GG_PROVIDER*

This parameter defines the Greengrass plugin that will be used and can be one of 2 values:

1. Parsec.
2. TPM (in case *cassini-tpm* is enabled).

The generated configuration is then transferred to DUT and the Greengrass service is restarted with folder permissions set.

- **Configure IDT for DUT**:

  This step is used to configure IDT installed on Docker container image with details of DUT and require the following:

  - *THING_IP* : IP address of DUT
  - *TARGET_MACHINE* : Machine name of DUT
  - *TARGET_PORT* : Port number to be used (default 22)

- **Assume role and run IDT**:

  After performing, *provisioning* and *configuring IDT*. The following parameters are required to run the tests:

  - *TEST_SESSION_NAME* : Test session name (optional, *see*)
  - *TEST_TIMEOUT_MULTIPLIER* : Set to extend the default timeout for tests

  This step will attempt to assume `IDT_ROLE` before running the IDT test suite. If this fails, the test suite will run with the permissions granted to the AWS user. These parameters needs to be configured as GitLab variables to be used by GitLab pipelines:

  - *IDT_ROLE*
  - *IDT_ROLE_SESSION_DURATION*

- **Cleanup**:

  This is used to perform cleanup of `IOT_THING_NAME` which represents the DUT name effectively when IDT tests have completed on CI/CD.

CHAPTER SEVEN

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.
```

7.1 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/

CHAPTER

EIGHT

RELEASE NOTES

8.1 v1.0.0

8.1.1 Known Issues or Limitations

All platforms

- This release uses Docker CE 23.0.2 which is vulnerable to CVE-2023-3750 and CVE-2023-2700. This release is therefore superseded by v1.0.1 which updates to Docker Moby v24.0.5. The update fixes the CVEs, and fixes a regression.
- The Parsec service configuration in this release enables detailed error trace which could include potentially sensitive information (key names and policies).
- There is a performance issue with RSA key generation - RSA key generation is much slower than expected

Corstone-1000

- RSA key generation fails with “PsaErrorDataInvalid” when using Parsec.
- Due to limited performance, characters may be dropped if too much data is sent too quickly. Consider inserting delays between characters if sending files or large buffers.
- Due to the limited performance, K3S is not currently supported.

N1SDP

- Support for the N1SDP platform in Cassini is primarily intended for development, test, and demonstration of features for infrastructure platforms which typically use EDK2 and Trusted Services without a secure enclave.
- Due to a limitation of the platform hardware (it does not have enough Secure world RAM) Trusted Services is configured to run from Normal world RAM. This configuration is not compliant with the PSA specifications. Platforms intended for production should be configured by the platform provider to be compliant with the PSA specifications.
- Booting the distribution image from USB storage device means storage performance may be may limited by that device. If unexpected issues or test failures are seen when booting from USB, try using a USB device with better performance or alternatively try booting from a SATA storage device.

8.1.2 Known Test Failures

All platforms

- Due to RSA key performance, the following tests take more time to complete than is stated in the developer guide:
  - Parsec simple end-to-end Tests may take up to 5 hours to complete
  - OP-TEE Sanity Tests may take up to 12 hours to complete
- The following tests are expected to fail:
  - Platform Security Architecture API Tests:
    - psa-crypto-api-test 262 (Test psa_hash_suspend - SHA224)
    - psa-crypto-api-test 263 (Test psa_hash_resume - SHA224)

Corstone-1000

- The following tests are expected to fail:
  - Parsec simple end-to-end Tests:
    - All RSA key tests fail with "PsaErrorDataInvalid" or "PsaErrorDoesNotExist"
    - The following are failures which are known to occur with the latest release of the Corstone-1000 platform software from meta-arm (CORSTONE1000-2023.06). See Test Report for CORSTONE1000-2023.06.
  - ACS BSA tests:
    - Check Cryptographic extensions (Failed on PE)
    - Check PMU Overflow signal (Invalid Interrupt ID number 0x750062)
    - Memory mapped timer check (Read-write check failed for CNTBaseN.CNTP_CTL, expected value 3)
    - Generate Mem Mapped SYS Timer Intr (Sys timer interrupt not received on 34)
    - Wake from System Timer Int (Received Failsafe interrupt)
  - ACS EBBR tests:
    - UEFI Compliant - Hii protocols must be implemented
    - UEFI Compliant - Boot from network protocols must be implemented
    - UEFI Compliant - DECOMPRESS protocol must exist
    - BS.ConnectController - InterfaceTestCheckpoint14 (F48D1C2D-1EBA-4E4C-A16D-748A01ABE6C1)
    - BS.ConnectController - InterfaceTestCheckpoint14 (25CFFDF5-D252-4515-AF8F-D8DB68F022C3)
    - BS.ConnectController - InterfaceTestCheckpoint14 (555913E8-BA56-4C68-80B5-A96B8A3AFCB1)
    - BS.GetNextMonotonicCount - high 32 bit increase by 1 (F48D1C2D-1EBA-4E4C-A16D-748A01ABE6C1)
    - BS.GetNextMonotonicCount - high 32 bit increase by 1 (E8B96EA0-6413-4947-AD1A-31EEF868A372)

    * BS.GetNextMonotonicCount - high 32 bit increase by 1 (0EC16C83-177D-461A-9622-42508C99D966)
    * RT.SetTime - Verify year after change
    * RT.SetTime - Verify month after change
    * RT.SetTime - Verify daylight after change
    * RT.SetTime - Verify time zone after change
    * RT.SetTime - Verify year after change
    * RT.SetTime - Verify month after change
    * RT.SetTime - Verify daylight after change
    * RT.SetTime - Verify time zone after change
    * RT.UpdateCapsule - invoke UpdateCapsule with invalid ScatterGatherList
    * RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags
    * RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags
  - ACS SCT Tests:
    * HII_DATABASE_PROTOCOL.ExportPackageLists (ExportPackageLists() returns EFI_BUFFER_TOO_SMALL)
    * HII_DATABASE_PROTOCOL.SetKeyboardLayout (SetKeyboardLayout() returns EFI_INVALID_PARAMETER)
  - ACS FWTS Tests:
    * Validity of fw_class in UEFI ESRT Table for EBBR (The fw_classis set to default u-boot raw guid)

N1SDP

- Some test failures are expected as the platform support is currently incomplete (pending further feature development):
  - Platform Security Architecture API Tests:
    * psa-api-iat-test 601 (fails with actual code 42, expected 0)
    * psa-api-ps-test 414 (fails with actual code 0, expected -134)
  - ACS BSA tests:
    * Check Arch symmetry across PE (Timed out for PE index = 2)
    * Check for AdvSIMD and FP support (PSCI_CPU_ON: cpu already on)
    * Check PE 4KB Granule Support (PSCI_CPU_ON: cpu already on)
    * Check HW Coherence support (PSCI_CPU_ON: cpu already on)
    * Check Cryptographic extensions (PSCI_CPU_ON: cpu already on)
    * Check Little Endian support (PSCI_CPU_ON: cpu already on)
    * Check EL1 and EL0 implementation (PSCI_CPU_ON: cpu already on)
    * Check for PMU and PMU counters (PSCI_CPU_ON: cpu already on)
    * Check num of Breakpoints and type (PSCI_CPU_ON: cpu already on)

    * Check Synchronous Watchpoints (PSCI_CPU_ON: cpu already on)
    * Check CRC32 instruction support (PSCI_CPU_ON: cpu already on)
    * Check Speculation Restriction (PSCI_CPU_ON: cpu already on)
    * Check Speculative Str Bypass Safe (PSCI_CPU_ON: cpu already on)
    * Check PEs Impl CSDB,SSBB,PSSBB (PSCI_CPU_ON: cpu already on)
    * Check PEs Implement SB Barrier (PSCI_CPU_ON: cpu already on)
    * Check PE Impl CFP,DVP,CPP RCTX (PSCI_CPU_ON: cpu already on)
    * Check PPI Assignments for OS (EL0-Phy timer not mapped to PPI recommended range)
    * Wake from EL0 PHY Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
    * Wake from EL0 VIR Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
    * Wake from EL2 PHY Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
  - ACS EBBR tests:
    * UEFI Compliant - Boot from network protocols must be implemented
    * BS.AllocatePool - Type is EfiMaxMemoryType
  - ACS SCT Tests:
    * BS.SetWatchdogTimer (should not reset after 3.5s)
  - ACS FWTS Tests:
    * Error: uefivarinfo (initialisation failed)

8.2 v1.0.1

8.2.1 Known Issues or Limitations

All platforms

- There is a performance issue with RSA key generation - RSA key generation is much slower than expected

Corstone-1000

- RSA key generation fails with “PsaErrorDataInvalid” when using Parsec.
- Due to limited performance, characters may be dropped if too much data is sent too quickly. Consider inserting delays between characters if sending files or large buffers.
- Due to the limited performance, K3S is not currently supported.

N1SDP

- Support for the N1SDP platform in Cassini is primarily intended for development, test, and demonstration of features for infrastructure platforms which typically use EDK2 and Trusted Services without a secure enclave.
- Due to a limitation of the platform hardware (it does not have enough Secure world RAM) Trusted Services is configured to run from Normal world RAM. This configuration is not compliant with the PSA specifications. Platforms intended for production should be configured by the platform provider to be compliant with the PSA specifications.
- Booting the distribution image from USB storage device means storage performance may be may limited by that device. If unexpected issues or test failures are seen when booting from USB, try using a USB device with better performance or alternatively try booting from a SATA storage device.

8.2.2 Known Test Failures

All platforms

- Due to RSA key performance, the following tests take more time to complete than is stated in the developer guide:
  - Parsec simple end-to-end Tests may take up to 5 hours to complete
  - OP-TEE Sanity Tests may take up to 12 hours to complete
- The following tests are expected to fail:
  - Platform Security Architecture API Tests:
    - psa-crypto-api-test 262 (Test psa_hash_suspend - SHA224)
    - psa-crypto-api-test 263 (Test psa_hash_resume - SHA224)

Corstone-1000

- The following tests are expected to fail:
  - Parsec simple end-to-end Tests:
    - All RSA key tests fail with "PsaErrorDataInvalid" or "PsaErrorDoesNotExist"
    - The following are failures which are known to occur with the latest release of the Corstone-1000 platform software from meta-arm (CORSTONE1000-2023.06). See Test Report for CORSTONE1000-2023.06.
  - ACS BSA tests:
    - Check Cryptographic extensions (Failed on PE)
    - Check PMU Overflow signal (Invalid Interrupt ID number 0x750062)
    - Memory mapped timer check (Read-write check failed for CNTBaseN.CNTP_CTL, expected value 3)
    - Generate Mem Mapped SYS Timer Intr (Sys timer interrupt not received on 34)
    - Wake from System Timer Int (Received Failsafe interrupt)
  - ACS EBBR tests:
    - UEFI Compliant - Hii protocols must be implemented
    - UEFI Compliant - Boot from network protocols must be implemented

    * UEFI Compliant - DECOMPRESS protocol must exist
    * BS.ConnectController - InterfaceTestCheckpoint14 (F48D1C2D-1EBA-4E4C-A16D-748A01ABE6C1)
    * BS.ConnectController - InterfaceTestCheckpoint14 (25CFFDF5-D252-4515-AF8F-D8DB68F022C3)
    * BS.ConnectController - InterfaceTestCheckpoint14 (555913E8-BA56-4C68-80B5-A96B8A3AFCB1)
    * BS.GetNextMonotonicCount - high 32 bit increase by 1 (F48D1C2D-1EBA-4E4C-A16D-748A01ABE6C1)
    * BS.GetNextMonotonicCount - high 32 bit increase by 1 (E8B96EA0-6413-4947-AD1A-31EEF868A372)
    * BS.GetNextMonotonicCount - high 32 bit increase by 1 (0EC16C83-177D-461A-9622-42508C99D966)
    * RT.SetTime - Verify year after change
    * RT.SetTime - Verify month after change
    * RT.SetTime - Verify daylight after change
    * RT.SetTime - Verify time zone after change
    * RT.SetTime - Verify year after change
    * RT.SetTime - Verify month after change
    * RT.SetTime - Verify daylight after change
    * RT.SetTime - Verify time zone after change
    * RT.UpdateCapsule - invoke UpdateCapsule with invalid ScatterGatherList
    * RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags
    * RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags

  - ACS SCT Tests:
    * HII_DATABASE_PROTOCOL.ExportPackageLists (ExportPackageLists() returns EFI_BUFFER_TOO_SMALL)
    * HII_DATABASE_PROTOCOL.SetKeyboardLayout (SetKeyboardLayout() returns EFI_INVALID_PARAMETER)
  - ACS FWTS Tests:
    * Validity of fw_class in UEFI ESRT Table for EBBR (The fw_classis set to default u-boot raw guid)

N1SDP

- Some test failures are expected as the platform support is currently incomplete (pending further feature development):
  - Platform Security Architecture API Tests:
    * psa-api-iat-test 601 (fails with actual code 42, expected 0)
    * psa-api-ps-test 414 (fails with actual code 0, expected -134)
  - ACS BSA tests:

    * Check Arch symmetry across PE (Timed out for PE index = 2)
    * Check for AdvSIMD and FP support (PSCI_CPU_ON: cpu already on)
    * Check PE 4KB Granule Support (PSCI_CPU_ON: cpu already on)
    * Check HW Coherence support (PSCI_CPU_ON: cpu already on)
    * Check Cryptographic extensions (PSCI_CPU_ON: cpu already on)
    * Check Little Endian support (PSCI_CPU_ON: cpu already on)
    * Check EL1 and EL0 implementation (PSCI_CPU_ON: cpu already on)
    * Check for PMU and PMU counters (PSCI_CPU_ON: cpu already on)
    * Check num of Breakpoints and type (PSCI_CPU_ON: cpu already on)
    * Check Synchronous Watchpoints (PSCI_CPU_ON: cpu already on)
    * Check CRC32 instruction support (PSCI_CPU_ON: cpu already on)
    * Check Speculation Restriction (PSCI_CPU_ON: cpu already on)
    * Check Speculative Str Bypass Safe (PSCI_CPU_ON: cpu already on)
    * Check PEs Impl CSDB,SSBB,PSSBB (PSCI_CPU_ON: cpu already on)
    * Check PEs Implement SB Barrier (PSCI_CPU_ON: cpu already on)
    * Check PE Impl CFP,DVP,CPP RCTX (PSCI_CPU_ON: cpu already on)
    * Check PPI Assignments for OS (EL0-Phy timer not mapped to PPI recommended range)
    * Wake from EL0 PHY Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
    * Wake from EL0 VIR Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
    * Wake from EL2 PHY Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
  - ACS EBBR tests:
    * UEFI Compliant - Boot from network protocols must be implemented
    * BS.AllocatePool - Type is EfiMaxMemoryType
  - ACS SCT Tests:
    * BS.SetWatchdogTimer (should not reset after 3.5s)
  - ACS FWTS Tests:
    * Error: uefivarinfo (initialisation failed)

8.3 v1.1.0

8.3.1 Known Issues or Limitations

Corstone-1000

- There is a performance issue with RSA key generation - RSA key generation is slower than expected. This is a known platform issue.
- Due to limited performance, characters may be dropped if too much data is sent too quickly. Consider inserting delays between characters if sending files or large buffers.

- Due to the limited performance, K3S is not currently supported.
- Due to the limited performance, there may be a timeout when pulling the first image from Docker Hub after the Docker daemon has started. Before running the container tests, it may be necessary to run the following command:

  ```
  sudo docker image pull -q hello-world
  ```

N1SDP

- Support for the N1SDP platform in Cassini is primarily intended for development, test, and demonstration of features for infrastructure platforms which typically use EDK2 and Trusted Services without a secure enclave.
- Due to a limitation of the platform hardware (it does not have enough Secure world RAM) Trusted Services is configured to run from Normal world RAM. This configuration is not compliant with the PSA specifications. Platforms intended for production should be configured by the platform provider to be compliant with the PSA specifications.
- Booting the distribution image from USB storage device means storage performance may be may limited by that device. If unexpected issues or test failures are seen when booting from USB, try using a USB device with better performance or alternatively try booting from a SATA storage device.

8.3.2 Known Test Failures

All platforms

- The following tests are expected to fail:
  - Platform Security Architecture API Tests:
    - psa-crypto-api-test 262 (Test psa_hash_suspend - SHA224)
    - psa-crypto-api-test 263 (Test psa_hash_resume - SHA224)

Corstone-1000

- The following tests are expected to fail:
  - The following are failures which are known to occur with the latest release of the Corstone-1000 platform software from meta-arm (CORSTONE1000-2023.06). See Test Report for CORSTONE1000-2023.11.
  - ACS BSA tests:
    - Check Cryptographic extensions (Failed on PE)
    - Check PMU Overflow signal (Invalid Interrupt ID number 0x750062)
    - Memory mapped timer check (Read-write check failed for CNTBaseN.CNTP_CTL, expected value 3)
    - Generate Mem Mapped SYS Timer Intr (Sys timer interrupt not received on 34)
    - Wake from System Timer Int (Received Failsafe interrupt)
  - ACS EBBR tests:
    - UEFI Compliant - Hii protocols must be implemented
    - UEFI Compliant - DECOMPRESS protocol must exist
    - BS.ConnectController - InterfaceTestCheckpoint14 (4643E80E-A6BF-412C-B4FF-9629282BC831)

    * BS.ConnectController - InterfaceTestCheckpoint14 (25CFFDF5-D252-4515-AF8F-D8DB68F022C3)
    * BS.ConnectController - InterfaceTestCheckpoint14 (555913E8-BA56-4C68-80B5-A96B8A3AFCB1)
    * BS.GetNextMonotonicCount - high 32 bit increase by 1 (F48D1C2D-1EBA-4E4C-A16D-748A01ABE6C1)
    * BS.GetNextMonotonicCount - high 32 bit increase by 1 (E8B96EA0-6413-4947-AD1A-31EEF868A372)
    * BS.GetNextMonotonicCount - high 32 bit increase by 1 (0EC16C83-177D-461A-9622-42508C99D966)
    * RT.QueryVariableInfo - With Attributes being 0
    * RT.QueryVariableInfo - With being an invalid combination
    * RT.SetTime - Verify year after change
    * RT.SetTime - Verify month after change
    * RT.SetTime - Verify daylight after change
    * RT.SetTime - Verify time zone after change
    * RT.SetTime - Verify year after change
    * RT.SetTime - Verify month after change
    * RT.SetTime - Verify daylight after change
    * RT.SetTime - Verify time zone after change
    * RT.UpdateCapsule - invoke UpdateCapsule with invalid ScatterGatherList
    * RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags
    * RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags
  - ACS SCT Tests:
    * HII_DATABASE_PROTOCOL.NewPackageList - NewPackageList() returns EFI_SUCCESS with valid inputs."

N1SDP

- Some test failures are expected as the platform support is currently incomplete (pending further feature development):
  - Platform Security Architecture API Tests:
    * psa-api-iat-test 601 (fails with actual code 42, expected 0)
    * psa-api-ps-test 414 (fails with actual code 0, expected -134)
  - ACS BSA tests:
    * Check Arch symmetry across PE (Timed out for PE index = 2)
    * Check for AdvSIMD and FP support (PSCI_CPU_ON: cpu already on)
    * Check PE 4KB Granule Support (PSCI_CPU_ON: cpu already on)
    * Check HW Coherence support (PSCI_CPU_ON: cpu already on)

        - Check Cryptographic extensions (PSCI_CPU_ON: cpu already on)
        - Check Little Endian support (PSCI_CPU_ON: cpu already on)
        - Check EL1 and EL0 implementation (PSCI_CPU_ON: cpu already on)
        - Check for PMU and PMU counters (PSCI_CPU_ON: cpu already on)
        - Check num of Breakpoints and type (PSCI_CPU_ON: cpu already on)
        - Check Synchronous Watchpoints (PSCI_CPU_ON: cpu already on)
        - Check CRC32 instruction support (PSCI_CPU_ON: cpu already on)
        - Check Speculation Restriction (PSCI_CPU_ON: cpu already on)
        - Check Speculative Str Bypass Safe (PSCI_CPU_ON: cpu already on)
        - Check PEs Impl CSDB,SSBB,PSSBB (PSCI_CPU_ON: cpu already on)
        - Check PEs Implement SB Barrier (PSCI_CPU_ON: cpu already on)
        - Check PE Impl CFP,DVP,CPP RCTX (PSCI_CPU_ON: cpu already on)
        - Check PPI Assignments for OS (EL0-Phy timer not mapped to PPI recommended range)
        - Wake from EL0 PHY Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
        - Wake from EL0 VIR Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
        - Wake from EL2 PHY Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
    - ACS FWTS Tests:
        - Error: uefivarinfo (initialisation failed)

8.4 v2.0.0

8.4.1 Known Issues or Limitations

Corstone-1000

- There is a performance issue with RSA key generation - RSA key generation is slower than expected. This is a known platform issue.
- Due to limited performance, characters may be dropped if too much data is sent too quickly. Consider inserting delays between characters if sending files or large buffers.
- Due to the limited performance, K3s is not currently supported.

Corstone-1000 FVP

- RsaPkcs1v15Crypt operations do not work properly due to the limited support in crypto-cell drivers

N1SDP

- Support for the N1SDP platform in Cassini is primarily intended for development, test, and demonstration of features for infrastructure platforms which typically use EDK2 and Trusted Services without a secure enclave.
- Due to a limitation of the platform hardware (it does not have enough Secure world RAM) Trusted Services is configured to run from Normal world RAM. This configuration is not compliant with the PSA specifications. Platforms intended for production should be configured by the platform provider to be compliant with the PSA specifications.
- Booting the distribution image from USB storage device means storage performance may be may limited by that device. If unexpected issues or test failures are seen when booting from USB, try using a USB device with better performance or alternatively try booting from a SATA storage device.

KV260

- RCU stalls may be detected during boot or in some test cases

8.4.2 Known Test Failures

All platforms

- The following tests are expected to fail:
  - AWS Greengrass IoT Device Tester
    - GGV2Q cloudcomponent and mqttpubsub fail a SignatureException on the device under test
    - GGV2Q lambdadeployment randomly fails to deploy the test component

Corstone-1000

- The following tests are expected to fail:
  - AWS Greengrass IoT Device Tester
    - GGV2Q docker tests time out while trying to create a deployment on the device
- **The following are failures which are known to occur with the latest**
  release of the Corstone-1000 platform software from meta-arm (CORSTONE1000-2024.06). See Test Report for CORSTONE1000-2024.06.
  - ACS BSA tests:
    - Check Cryptographic extensions (Failed on PE)
    - Check PMU Overflow signal (Invalid Interrupt ID number 0x750062)
    - Memory mapped timer check (Read-write check failed for CNTBaseN.CNTP_CTL, expected value 3)
    - Generate Mem Mapped SYS Timer Intr (Sys timer interrupt not received on 34)
    - Wake from System Timer Int (Received Failsafe interrupt)
  - ACS EBBR tests:
    - UEFI Compliant - Hii protocols must be implemented
    - UEFI Compliant - DECOMPRESS protocol must exist

    - BS.ConnectController - InterfaceTestCheckpoint14 (4643E80E-A6BF-412C-B4FF-9629282BC831)
    - BS.ConnectController - InterfaceTestCheckpoint14 (25CFFDF5-D252-4515-AF8F-D8DB68F022C3)
    - BS.ConnectController - InterfaceTestCheckpoint14 (555913E8-BA56-4C68-80B5-A96B8A3AFCB1)
    - BS.GetNextMonotonicCount - high 32 bit increase by 1 (F48D1C2D-1EBA-4E4C-A16D-748A01ABE6C1)
    - BS.GetNextMonotonicCount - high 32 bit increase by 1 (E8B96EA0-6413-4947-AD1A-31EEF868A372)
    - BS.GetNextMonotonicCount - high 32 bit increase by 1 (0EC16C83-177D-461A-9622-42508C99D966)
    - RT.QueryVariableInfo - With Attributes being 0
    - RT.QueryVariableInfo - With being an invalid combination
    - RT.SetTime - Verify year after change
    - RT.SetTime - Verify month after change
    - RT.SetTime - Verify daylight after change
    - RT.SetTime - Verify time zone after change
    - RT.SetTime - Verify year after change
    - RT.SetTime - Verify month after change
    - RT.SetTime - Verify daylight after change
    - RT.SetTime - Verify time zone after change
    - RT.UpdateCapsule - invoke UpdateCapsule with invalid ScatterGatherList
    - RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags
    - RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags
  - ACS SCT Tests:
    - HII_DATABASE_PROTOCOL.ExportPackageLists - ExportPackageLists() returns EFI_BUFFER_TOO_SMALL with BufferSize indicates the buffer is too small
    - HII_DATABASE_PROTOCOL.NewPackageList - NewPackageList() returns EFI_SUCCESS with valid inputs."

Corstone-1000 FVP

- The following tests are expected to fail:
  - PARSEC simple E2E tests
    - Decrypt operations fail with PsaErrorHardwareFailure when using psa-security

N1SDP

- Some test failures are expected as the platform support is currently incomplete (pending further feature development):
  - Platform Security Architecture API Tests:
    - psa-api-iat-test 601 (fails with actual code 42, expected 0)
    - psa-api-ps-test 414 (fails with actual code 0, expected -134)
  - ACS BSA tests:
    - Check Arch symmetry across PE (Timed out for PE index = 2)
    - Check for AdvSIMD and FP support (PSCI_CPU_ON: cpu already on)
    - Check PE 4KB Granule Support (PSCI_CPU_ON: cpu already on)
    - Check HW Coherence support (PSCI_CPU_ON: cpu already on)
    - Check Cryptographic extensions (PSCI_CPU_ON: cpu already on)
    - Check Little Endian support (PSCI_CPU_ON: cpu already on)
    - Check EL1 and EL0 implementation (PSCI_CPU_ON: cpu already on)
    - Check for PMU and PMU counters (PSCI_CPU_ON: cpu already on)
    - Check num of Breakpoints and type (PSCI_CPU_ON: cpu already on)
    - Check Synchronous Watchpoints (PSCI_CPU_ON: cpu already on)
    - Check CRC32 instruction support (PSCI_CPU_ON: cpu already on)
    - Check Speculation Restriction (PSCI_CPU_ON: cpu already on)
    - Check Speculative Str Bypass Safe (PSCI_CPU_ON: cpu already on)
    - Check PEs Impl CSDB,SSBB,PSSBB (PSCI_CPU_ON: cpu already on)
    - Check PEs Implement SB Barrier (PSCI_CPU_ON: cpu already on)
    - Check PE Impl CFP,DVP,CPP RCTX (PSCI_CPU_ON: cpu already on)
    - Check PPI Assignments for OS (EL0-Phy timer not mapped to PPI recommended range)
    - Wake from EL0 PHY Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
    - Wake from EL0 VIR Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
    - Wake from EL2 PHY Timer Int (Invalid Interrupt ID number 0xAFAFAFAF)
  - ACS FWTS Tests:
    - Error: uefivarinfo (initialisation failed)

8.5 v2.1.0

8.5.1 Known Issues or Limitations

All platforms

- Cassini will no longer support parsec-service in future releases.

Corstone-1000

- Due to limited performance, characters may be dropped if too much data is sent too quickly. Consider inserting delays between characters if sending files or large buffers.
- Due to the limited performance, K3s is not currently supported.

Corstone-1000 FVP

- RsaPkcs1v15Crypt operations do not work properly due to the limited support in crypto-cell drivers

KV260 with EDK-II

- Data abort faults are encountered when using UEFI memory allocation boot services

8.5.2 Known Test Failures

All platforms

- The following tests are expected to fail:
  - AWS Greengrass IoT Device Tester
    - GGV2Q cloudcomponent and mqttpubsub fail a SignatureException on the device under test
    - GGV2Q lambdadeployment randomly fails to deploy the test component

Corstone-1000

- The following tests are expected to fail:
  - AWS Greengrass IoT Device Tester
    - GGV2Q docker tests time out while trying to create a deployment on the device
- The following are failures which are known to occur with the latest release of the Corstone-1000 platform software from meta-arm (CORSTONE1000-2024.11). See Test Report for CORSTONE1000-2024.11.
  - ACS BSA tests:
    - Check Arch symmetry across PE
    - Check for AdvSIMD and FP support
    - Check PE 4KB Granule Support
    - Check Cryptographic extensions

    - Check Little Endian support
    - Check EL1 and EL0 implementation
    - Check for PMU and PMU counters
    - Check PMU Overflow signal
    - Check num of Breakpoints and type
    - Check Synchronous Watchpoints
    - Check CRC32 instruction support
    - Memory mapped timer check
    - Generate Mem Mapped SYS Timer Intr
    - Wake from System Timer Int
  - ACS EBBR tests:
    - UEFI Compliant - Hii protocols must be implemented
    - UEFI Compliant - DECOMPRESS protocol must exist
    - BS.ConnectController - InterfaceTestCheckpoint14 (4643E80E-A6BF-412C-B4FF-9629282BC831)
    - BS.ConnectController - InterfaceTestCheckpoint14 (25CFFDF5-D252-4515-AF8F-D8DB68F022C3)
    - BS.ConnectController - InterfaceTestCheckpoint14 (555913E8-BA56-4C68-80B5-A96B8A3AFCB1)
    - BS.GetNextMonotonicCount - high 32 bit increase by 1 (F48D1C2D-1EBA-4E4C-A16D-748A01ABE6C1)
    - BS.GetNextMonotonicCount - high 32 bit increase by 1 (E8B96EA0-6413-4947-AD1A-31EEF868A372)
    - BS.GetNextMonotonicCount - high 32 bit increase by 1 (0EC16C83-177D-461A-9622-42508C99D966)
    - RT.QueryVariableInfo - With Attributes being 0
    - RT.QueryVariableInfo - With being an invalid combination
    - RT.SetTime - Verify daylight after change
    - RT.SetTime - Verify time zone after change
    - RT.SetTime - Verify year after change
    - RT.SetTime - Verify month after change
    - RT.SetTime - Verify time zone after change
  - ACS SCT Tests:
    - HII_DATABASE_PROTOCOL.ExportPackageLists - ExportPackageLists() returns EFI_BUFFER_TOO_SMALL with BufferSize indicates the buffer is too small
    - HII_DATABASE_PROTOCOL.NewPackageList - NewPackageList() returns EFI_SUCCESS with valid inputs.

Corstone-1000 FVP

- The following tests are expected to fail:
  - PARSEC simple E2E tests
    - Decrypt operations fail with PsaErrorHardwareFailure when using psa-security
  - Greengrass tests
    - Provisioning fails due to the RSA limitation. Hence, all IDT tests are expected to fail except for dependency and version checks.

KV260 with U-Boot

- The following tests are expected to fail:
  - ACS BSA tests:
    - Check Arch symmetry across PE
    - Check for AdvSIMD and FP support
    - Check PE 4KB Granule Support
    - Check Cryptographic extensions
    - Check Little Endian support
    - Check EL1 and EL0 implementation
    - Check for PMU and PMU counters
    - Check num of Breakpoints and type
    - Check Synchronous Watchpoints
    - Check CRC32 instruction support
    - SYS Timer if PE Timer not ON
    - Check Arm BSA UART register offsets
    - 16550 compatible UART
  - ACS EBBR tests:
    - Same failures for Corstone-1000, along with the following:
      - BS.ExitBootServices - ConsistencyTestCheckpoint1
      - RT.SetVariable - Non-volatile variable after system reset
      - RT.UpdateCapsule - invoke UpdateCapsule with invalid ScatterGatherList
      - RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags
      - RT.UpdateCapsule - invoke UpdateCapsule with invalid Flags

KV260 with EDK-II

- The following tests are expected to fail:
  - ACS is expected to fail during memory allocation tests.