Apertis Platform Technical Vision

Contents

Overview

The intention of this document is to outline the Apertis technical direction and vision for the Apertis platform as it would be used on a device, container or VM.

This document does not cover the overall Apertis infrastructure and its general principles around for example open source license expectations[1], image building[2] and the release process[3]. These topics are covered in their own respective documents.

Apertis is, by design, a very flexible platform to build on. However, as a project we cannot directly support and test every possible setup that can be built using Apertis. Because of that our development and testing is focused on a few specific hardware reference platforms[4] which can be used as the basis to explore a wide range of use-cases. Apertis users are of course still free to choose their hardware platforms and configure their systems differently, utilising as much or as little of that provided by Apertis as they see fit!

The following sections look at various scenarios in which Apertis can be used on devices and the technical building blocks that Apertis intends to make available for them. As such they all describe a reasonably full-featured setup for each scenario, however an actual deployment can still choose to only use a subset of available features.

As this document provides a forward-looking vision of the Apertis platform not all features mentioned are yet implemented or even fully defined.

[1] https://em.pages.apertis.org/apertis-website/policies/license-expectations/
[2] https://em.pages.apertis.org/apertis-website/guides/image_building/
[3] https://em.pages.apertis.org/apertis-website/guides/apertis_release_process/
[4] https://em.pages.apertis.org/apertis-website/reference_hardware/

Fixed function device scenario

This setup represents an "appliance" or fixed function devices. That is to say the device is intended to be used for a specific functionality which cannot be extended by installing extra software on top of the base system. This scenario also assumes it's operating in a headless fashion without a comprehensive graphical user interface. It is still possible to have some amount of user interactions for example via simple buttons or knobs for user inputs and lights and/or segmented displays for outputs, however typical inputs and outputs for these devices will be attached peripherals and sensors.

Typically these devices are expected to be connected to an IP network and have the ability to either directly or indirectly connected to internet services (e.g. via ethernet, wifi, 5G).

As security and integrity is paramount the device supports a fully verified boot sequence (secure boot, verified boot or similar) to ensure untampered firmware, kernel, etc are used. The system (root) filesystem is integrity measured for all executables to get a fully trusted system.

To further improve integrity and privacy of the system a Trusted Execution environment is available for trusted applications (e.g. optee). The TEE environment or a dedicated chip (e.g. TPM) are available to support remote attestation as well support for encrypted areas (filesystems etc) which can only be accessed by a specific device and only when it has been booted into a known state.

On the running OS systemd[5] for overall system startup and management. This also includes managing the usage of linux capabilities, resource constraints, system call filtering, sandboxing of services through linux namespaces and providing start-on-demand as well as watchdog services.

On top of systemd the Apparmor LSM is used to further constraint the behaviour of system processes and provide a second line of protection increasing the defense in depth.

In the current world of software nothing is secure if it's not also up to date with the latest security fixes. As such the system comes with an ostree based over the air update system with the capability to integrate with cloud device management and fleet management systems such as Hawkbit.

Building upon the fleet management integration, while updates provide one piece of the puzzle telemetric support provides another aspect to enable remote management and monitoring of devices.

As a lot of this functionality needs network access, so of course Apertis supports various ways of accessing networks whether this is via wired connection, wifi or mobile networks.

[5] https://www.freedesktop.org/wiki/Software/systemd/

feature	documentation	status
Verified boot sequence	fully verified boot sequence[6]	Implemented
Integrity validate root filesystem	Security[7]	Concept: Requires Update
Filesystem Encryption		
Trusted Execution Environment	Trusted Execution Environment[8]	Concept: Up-to-date
System service lifecycle management	system startup and management[9]	Partially Implemented
Apparmor (LSM)	Apparmor[10]	Implemented
OTA/system upgrade	OSTree based[11]	Implemented
OTA Fleet management	Preparing hawkBit for Production Use[12]	Concept: Up-to-date
Network connectivity	Connectivity[13]	Concept: Requires Update

HMI device scenario

This setup is targeted at devices with a HMI. Typically this will be a modern graphical user interface driven via a touch-screen and/or other inputs as well as the capability of audio in and outputs. Some devices may also have one or more cameras or other sensors attached. Furthermore these devices can be extended via the installation of user-facing applications.

As a basis these devices have all the features and capabilities of the fixed function device scenario.

On top of this base functionality a user interface is available to launch different applications or functions via a touch-screen or other inputs. This user interface is composed from a base wayland compositor which can be re-used and customised for specific projects, for Apertis a reference UX shell is available in the form of the maynard compositor.

For video inputs as well as audio input and output pipewire is used as a routing daemon with wireplumber adding policy support. This allows multiple applications to use these streams at the same time while also being able to prioritise between them and implement more complex policies like for example audio ducking.

The additions of applications to the system can be done via the installation of flatpak-based application bundles, which allows applications to be installed with minimal dependencies on the base system. This also makes it possible to have

[6] https://em.pages.apertis.org/apertis-website/architecture/secure-boot/
[7] https://em.pages.apertis.org/apertis-website/concepts/security/
[8] https://em.pages.apertis.org/apertis-website/concepts/op-tee/
[9] https://em.pages.apertis.org/apertis-website/architecture/boot_process/
[10] https://em.pages.apertis.org/apertis-website/guides/apparmor/
[11] https://em.pages.apertis.org/apertis-website/guides/ostree/
[12] https://em.pages.apertis.org/apertis-website/concepts/preparing-hawkbit-for-production/
[13] https://em.pages.apertis.org/apertis-website/concepts/connectivity/

separate update lifecycles for applications and the base operating system as typically applications are updated at a far shorter cycle then the operating system itself. The usage of flatpak-based applications also enables the implementation of dynamic policies for what resources are available for an application. For example camera access might only be allowed for some applications.

Flatpak applications can either be provisioned via a fleet management system (such as Hawkbit) or from a "app store" application available on the system with application specific download methods.

feature	documentation	status
Reference HMI shell and compositor	Application Framework[14]	Concept: Up-to-date
Bluetooth	Connectivity: Bluetooth Support[15]	Concept: Requires Update
System toolkit		
Virtual system keyboard		
Audio routing and policy	Audio management[16]	Concept: Up-to-date
Video routing and policy		
Application framework integration	Application framework[17]	Concept: Up-to-date
Application management (store or fleet)	hawkBit[18]	Partially Implemented
Removable storage management		

Focus area Flatpak applications

As Flatpak applications are decoupled from the base system they are essentially their own dedicated setup.

To be able to easily build Flatpak applications targeting Apertis systems, while still taking the benefits of the Apertis maintenance, dedicated Apertis runtime including SDK variants and debug extensions (needed for development) will be provided. These runtime will include a basic set of libraries for libraries to rely on.

However certain applications or devices can have special needs for the libraries available in their standard runtime. Custom runtimes can also be created as needed.

For some devices specific extra or different libraries can be required. For example to support a device specific GL or Vulkan stack or device specific codec libraries. These can be shipped as a flatpak runtime extension, allowing multiple devices

[14] https://em.pages.apertis.org/apertis-website/concepts/application-framework/#compositor-libweston
[15] https://em.pages.apertis.org/apertis-website/concepts/connectivity/#bluetooth-support
[16] https://em.pages.apertis.org/apertis-website/concepts/audio-management/
[17] https://em.pages.apertis.org/apertis-website/concepts/application-framework/
[18] https://em.pages.apertis.org/apertis-website/guides/deployment-management/

to use the same base runtimes but adjust as needed to take full advantage of the underlying hardware.

Apart from this base infrastructure application may also have a need to access peripherals, sensors, audio inputs and outputs as well as other lower-level system interfaces. For this purpose flatpaks concepts of portals will be used, which allows the system to apply applications specific policies and permissions. An example policy is whether a given application can access the devices camera potentially only after explicit user interaction.

feature	documentation link	status
Apertis supported Flatpak runtimes		
Runtime extensions for HW specific support		
Flatpak System API access (portals)		
Flatpak audio and video routing		
Creation of Flatpaks		

(Industrial) IOT scenario

Another area in which Apertis focuses is industrial IOT. In a sense these devices are close to fixed function devices in that they've got little to no ability to support user interaction. However these are not fixed function devices, these devices are targeted at running "edge" workloads with the ability to dynamically manage which devices run specific workloads.

Typically these devices collect data on the edge either directly by containing various sensors (e.g. cameras, power measurement, temperature etc) or indirectly via other devices on a local network (which could be fixed function apertis devices). The role of these devices typically is to process all these inputs in some fashion and either act on them or relay it to a cloud infrastructure.

To allow workloads to be flexible they will be executed as containerized workloads; These containers are distributed and run using the open container initiative (OCI) standards. Apertis will include an open-source workload orchestrater to interact with a management system to orchestrate deployments.

Another role an edge device can take is as an orchestrator for deploying software updates or workloads to secondary devices. For example an accompanying microcontroller running a real time operating system or a separate network-attached system which is not (or cannot be) directly connected to the internet. The orchestrater is able to manage and push updates to these devices.

feature	documentation link	status
OCI container orchestration		
Remote firmware deployment		

feature	documentation link	status
Remote workload deployment		
OCI container management		

Focus area container runtime

Like Flatpaks OCI images to be run on a device are separate from the main system to decouple the two. OCI images are the standard way of distributing cloud workload for network services which is a good and natural fit for the workloads on edge devices. This allows the Apertis device to take advantage of common workflows developers are used to for building images targeting cloud deployments.

The other benefit of using OCI standards is the potential to use/consider different CRI-O[19] implementation, including implementations using hypervisor technology to allow for an even bigger separation of host and container then typically available on namespace based container systems.

feature	documentation link	status
Guidelines for OCI container deployments		
OCI container building		
OCI container integration with system API		

SDK scenario

The final area is the SDK environment; While not really a product as such, ease of development is a critical aspect for the success of Apertis. To enable smooth development Apertis provides a pre-build virtual machine for development (which in a sense is yet another device). This is preinstalled with all the tooling required to locally build all aspects of the Apertis universe such as:

- Build Debian packages for target devices
- Build full system images
- Build Flatpak applications
- Build OCI images

Apart from building, deployment and debugging should be as easy as possible. To support that, tooling is included to directly provision the various artifacts to local devices (depending on the device installation of course).

feature	documentation link	status
Debian package building (devroot, sysroot)	sysroots and devroots[20]	Implemented

[19] https://cri-o.io/

feature	documentation link	status
System image building (debos)	Image building[21]	Implemented
Flatpak image building		
OCI image building		
SDK development environment (IDE)	SDK[22]	Implemented
Local Device deployment/debugging	ADE[23]	Implemented: Requires Update

[20] https://em.pages.apertis.org/apertis-website/architecture/sysroots-and-devroots/
[21] https://em.pages.apertis.org/apertis-website/guides/image_building/
[22] https://em.pages.apertis.org/apertis-website/guides/virtualbox/
[23] https://em.pages.apertis.org/apertis-website/guides/ade/