Trusted Containers

FABIO VALLONE

Outline

01 – Introduction

02 – Intel Cloud Integrity Technology (CIT)

03 – Core Os & rkt

04 – SECURED

05 – Comparison

Introduction

What is Trusted Computing?

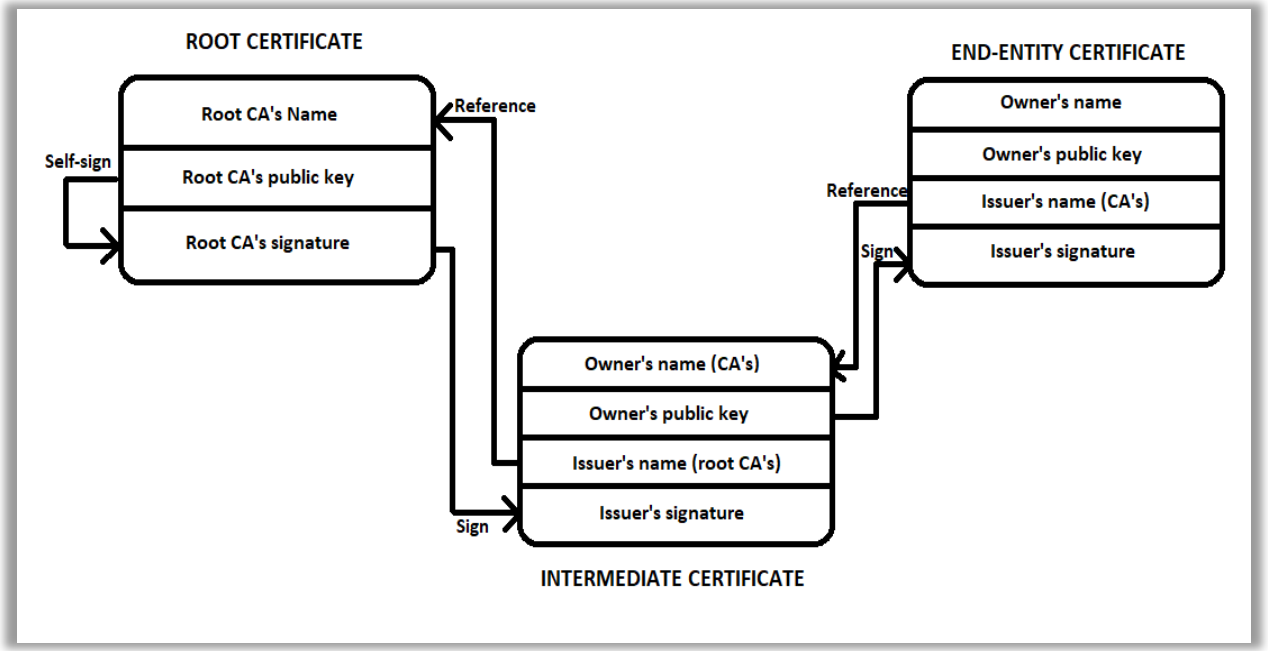
According to the definition given by **IETF** in **RFC4949,** a trusted system is:

> *"$ trusted system 1. (I) /information system/ A system that operates as expected, according to design and policy, doing what is required -- despite environmental disruption, human user and operator errors, and attacks by hostile parties -- and not doing other things [NRC98]. (See: trust level, trusted process. Compare: trustworthy.)"* [1]

[1] https://tools.ietf.org/html/rfc4949

Introduction

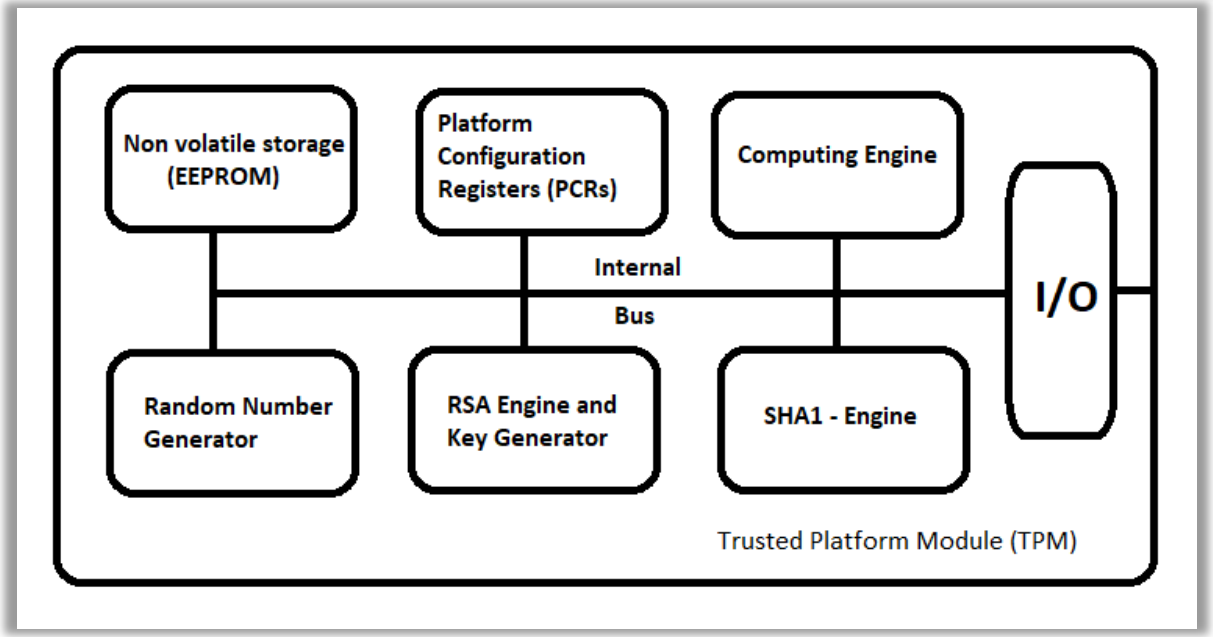
Chain of trust



In Computer Security a **chain of trust** is a technique that allow us to validate the last element by **validating each element from the bottom to the top**. Each component will validate the next component.
In this way the trust is transferred from the bottom level to the upper one. This technique is used during the validation of a certificate: In order to validate a certificate, we must follow all the **Certification Authority (CA)** chain until we found the root one. The certificate of the root CA is called the **Root of Trust** and it is **trusted by default**.

Introduction

Trusted Platform Module

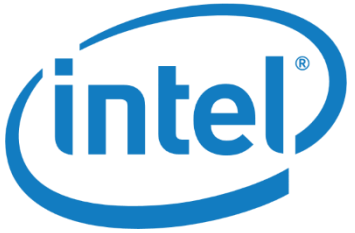


The **Trusted Platform Module (TPM)** is a **tamper proof Hardware module** that it is often used as the **root of trust** in a Trusted Computing (TC) environment. The important component in a TC environment are:

- **PCR**: Platform Configuration Register, they are protected memory regions where it is possible to store some data
- **Cryptographic engine**: Used for cryptographic operations. In v1 it is available a SHA1-engine, while v2 needs a SHA2-engine

Intel CIT

Overview



Intel **Cloud Integrity Technology (CIT)** is a platform developed by Intel with the aim to provide an attestation framework for all the Cloud components. It is the successor of the Open Attestation framework. It extends the **hardware root of trust** up to the last software layer. It is based on the proprietary Intel **Trusted Execution Technology (TXT)** and needs a compatible Intel processor in order to work.

Intel CIT

Features

The **key features** of the Intel **Cloud Computing Technology** are:

GeoLocation

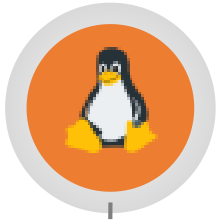
Support for Trusted Location and Boundary Control

Standard VM

Support for most of the standard virtualization technology, like ESX Citrix-Xen, KVM and Hyper-v

Docker

Direct support for light virtualization technology like Docker

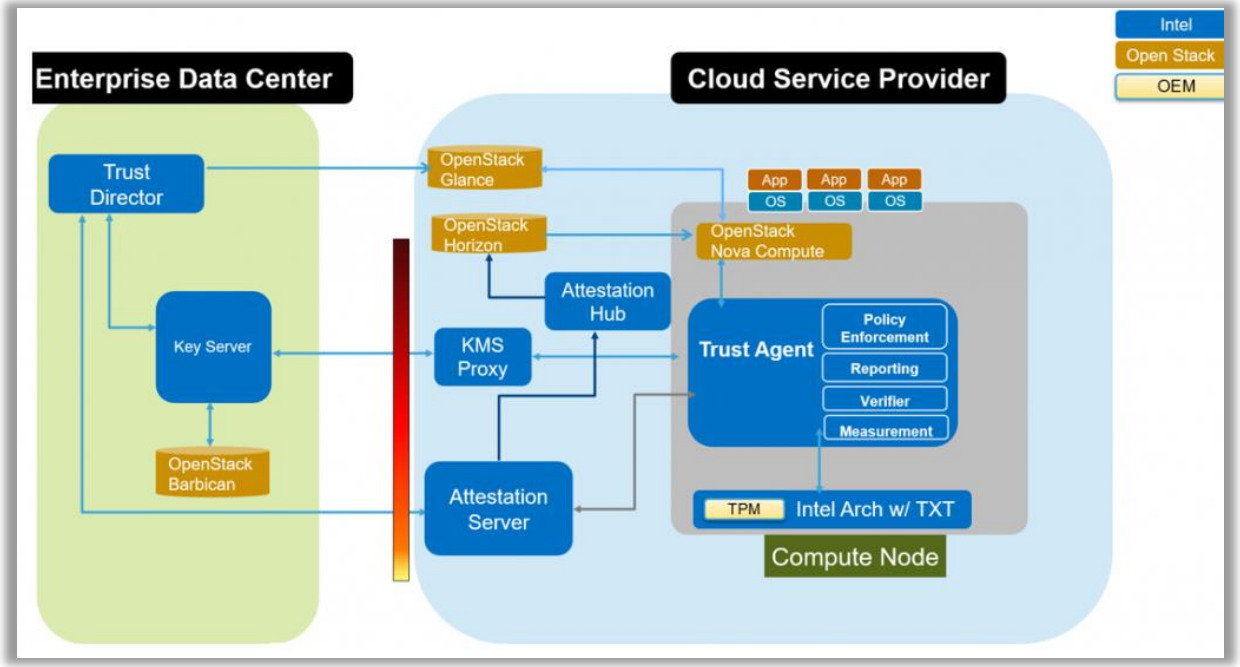


Multiple OS

It doesn’t require a particular Os in order to run, since it provides the client for the major Os available

Intel CIT

Architecture



https://01.org/opencit

The main components are:

- **Attestation Server:** Is the server used to attest the registered nodes
- **Trust Director:** Used to encrypt the generated VMs images
- **Trust Agent:** Host application that enables the RA process for the client
- **Key Broker Service:** Provides and retains encryption/decryption keys for the images launched by the VMs
- **KMS Proxy:** Used to manage the launch request of encrypted images

Intel CIT

VM encryption

With the Intel CIT is possible to **encrypt VM images** so that we can **enforce** that the image can be **run only by a verified host**. In particular the step required for run an encrypted image are:

- The host **request** the **key** to the **KMS Proxy** and provide its **Attestation Identity Key (AIK)** taken from its TPM
- The **KMS Proxy** will check its integrity status with the **Attestation Server** using the AIK as the host Identity
- If the Verification is ok, the **decryption key is bound to the AIK** and is sent back to the host
- The host **unwrap the key using its TPM** and runs the decrypted image

Intel CIT

Image integrity & signature

In the **Intel Cloud Integrity Technology** image integrity is enforced by **measuring all the files that are inside the image** that needs to be loaded, and by comparing those measure to the good one saved in a whitelist. This operation is done **every time the image is about to be loaded in memory**. No run time integrity check is available.

Since it supports a wide variety of VM engines, **the image signature feature is delegated to the engine of choice**.

Intel CIT

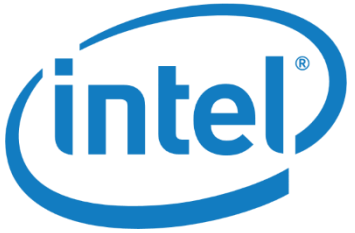
Boundary Control

https://01.org/opencit

With the Intel Cloud Integrity Technology platform it is possible to set **Boundary out of which data cannot be transferred**. For example, this technology, can be used to **prevent** that a particular **VM image** can be executed in a particular country or location. In case of host located in multiple location, it is possible to tell to Intel CIT that a particular workload must be done on a trusted location. All this features use the **GeoTag** information available in the Intel TXT architecture.

Intel CIT

Intel TXT



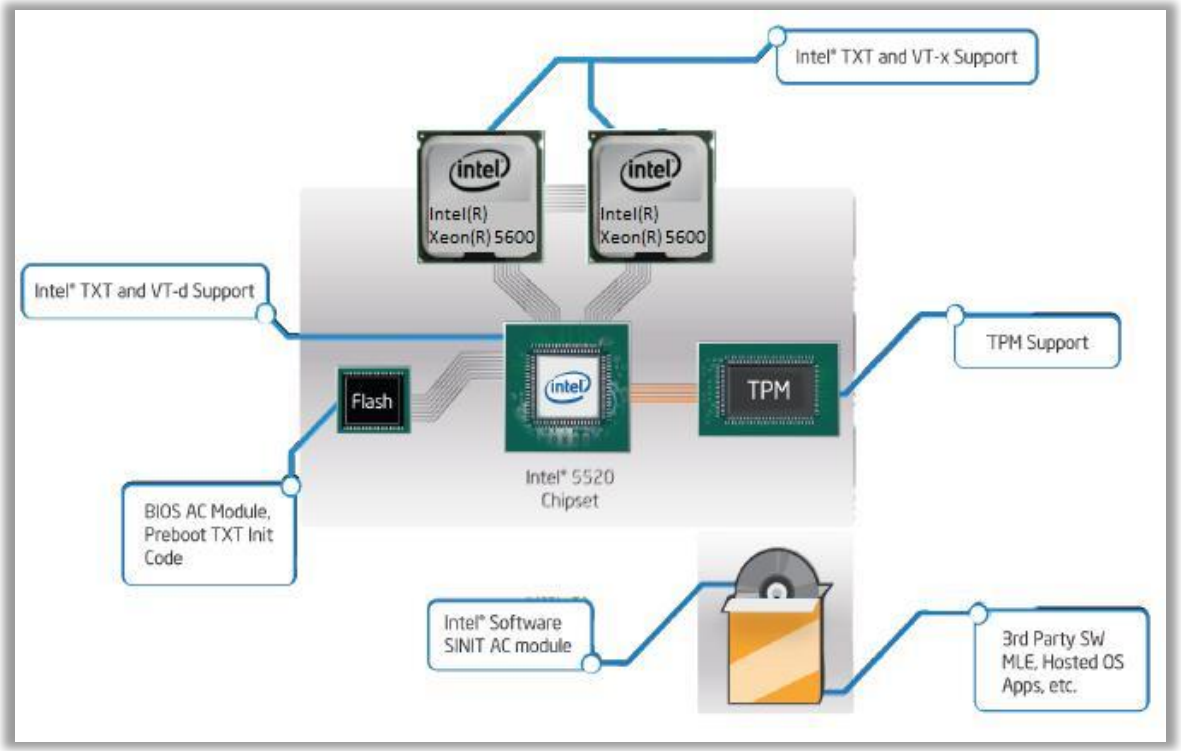
Intel **Trusted Execution Technology (TXT)** is an **Hardware architecture** presented by Intel in order to **validate platform trustworthiness** during the **boot** and **launch** of software. In order to work the system must be equipped with a **Trusted Platform Module (TPM) v1.2** on which the Intel TXT can extends the measure done. It is used as the **Root of Trust** for the Intel Cloud Integrity Technology platform.

Intel CIT

Intel TXT

It requires the following components:

- **Intel VT-d:** Intel proprietary on-chip Virtualization technology with support for directed I/O
- **TPM v1.2:** security chip compatible with the standard v1.2 proposed by the Trusted Computing Group (TCG)
- **ACM enabled BIOS:** Bios with the support for Authenticate Code Module technology



https://software.intel.com/en-us/blogs/2012/09/25/how-to-enable-an-intel-trusted-execution-technology-capable-server

Intel CIT

Intel TXT – ACM

The **Authenticated Code Module (ACM)** is an **hardware module digitally signed** that is provided by the chipset manufacturer. When establishing a new chain of trust, it is the **first** that is **measured** by the processor. When its signature and integrity are successfully validated, **it will** then proceed to **validate** the first module of **the BIOS**. All this measure are then extended to the PCR0 of the **Trusted Platform Module (TPM)** and they are considered the **root of trust.**

Core Os & rkt

Introduction

Container Linux is a lightweight, virtualization based Operating System developed by **Core Os**. All the applications are required to run on different **containers**. At the beginning it supports only **Docker** as container engine. In 2014, Core Os released a new container engine called **Rocket (rkt)**, that is designed with security and efficiency in mind.

Core Os & rkt

Container Linux - Features

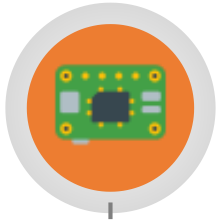
The **key features** of the **Container Linux OS** are:

Containers

It is designed to run only container, so no overhead due to unused software processes

Scalability

Greater scalability due to easy management done through specific designed and easy to use tools



Hardware

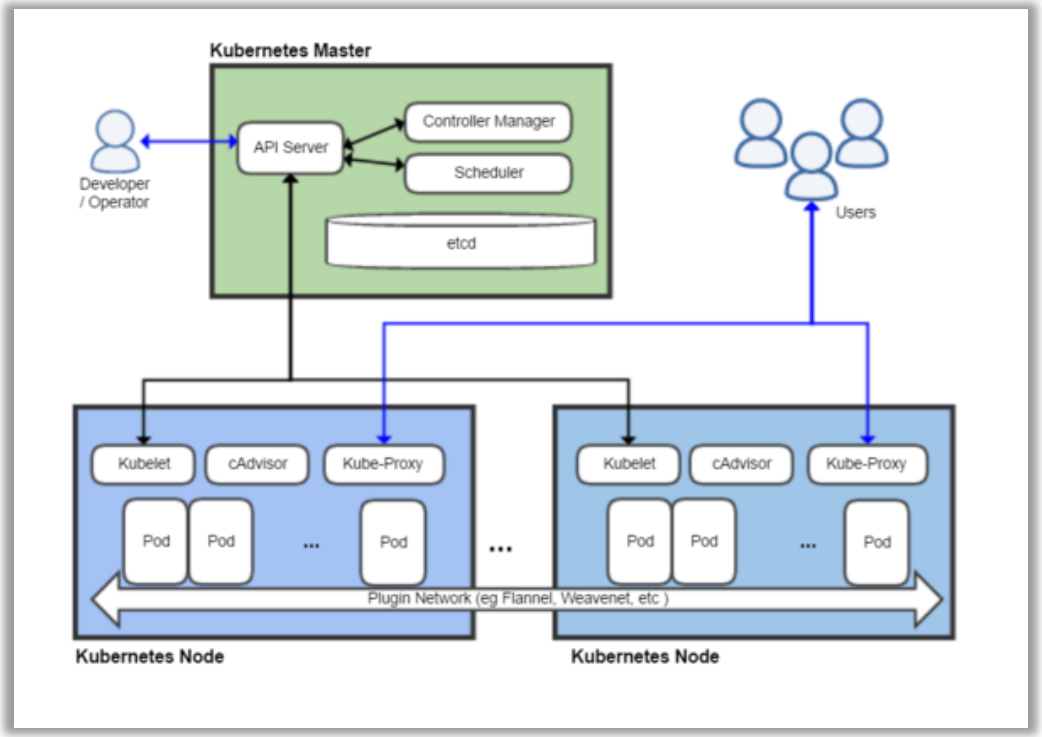
Doesn’t require particular hardware except for a TPM to use the Trusted Computing features

Secure

Provide all the security features including the support for Trusted Computing through Kubernetes

Core Os & rkt

Container Linux – Kubernetes



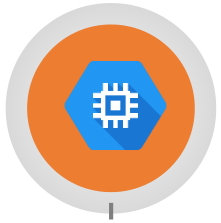
https://en.wikipedia.org/wiki/Kubernetes

Kubernetes is an open-source platform developed by Google in order to **automatically manage a cluster of container nodes**. It is structured with a **Client-Server architecture**. The basic unit is the **pod** that corresponds to a set of containers that needs to run on the same location. All the nodes are managed by the **Master** by talking to the **kubelet**, that is an application that runs on the host and manage all the pods locally.

Core Os & rkt

Rocket – Features

The **key features** of the **Rocket** container engine are:



Non-root

All the operations done by rkt doesn't require root access, granting a safer way to manage containers

Security

It offers integrated support for SELinux, Distributed Trusted Computing through TPM and Intel Clear Container

Modular

All the features are developed in different binaries, so it is possible to add or remove functionality on the fly

Docker Support

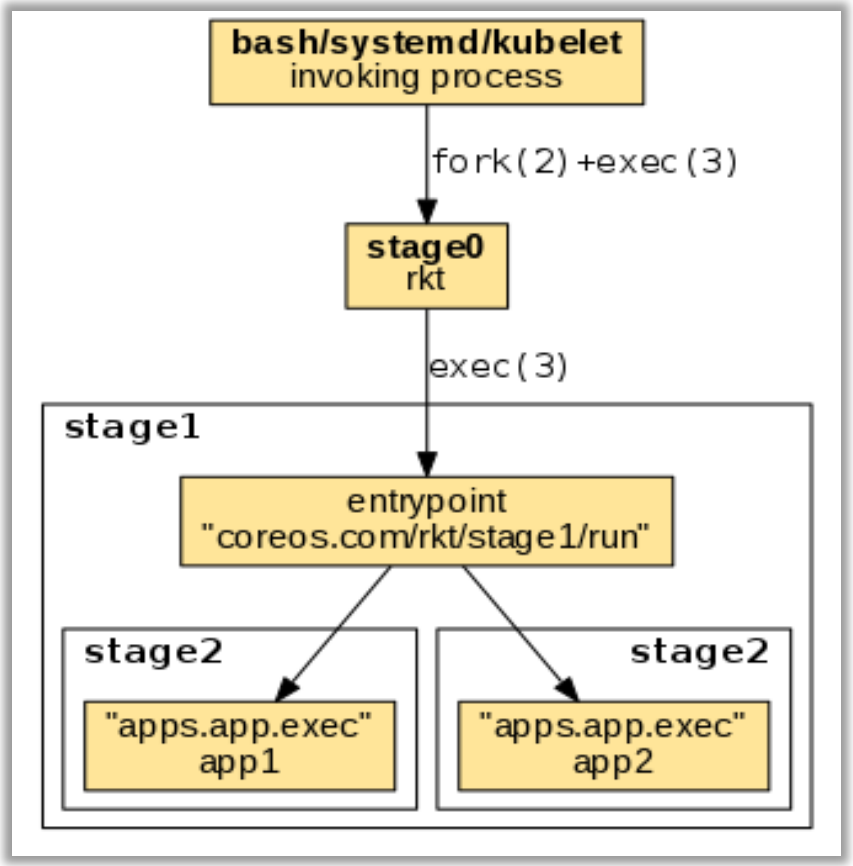
It is possible to convert a docker image or a dockerfile directly to an App Container for Rocket

Core Os & rkt

Rocket – Architecture

The basic unit in Rocket is the **pod**. Rocket **doesn't have a daemon** but each time a new container is started, it will create a new process divided in different **stages**:

- **Stage 0:** It will run the rkt binary and do some preparation task like generating the fs for the pod or making the pod UUID
- **Stage 1:** Read the Pod manifest and create the necessary isolation level
- **Stage 2:** Run the actual application chosen by the user



https://coreos.com/rkt/docs/latest/devel/architecture.html

Core Os & rkt

Rocket – Isolation

For each **pod** to be launched, it is possible to define on his **manifest** the appropriate isolation level. The available values are:

- **Fly:** It is the lowest possible isolation level. It is implemented as a simple chroot environment.
- **Systemd/nspawn:** It is the medium isolation level. It is implemented using a combination of cgroups and namespaces functionality using systemd or systemd-nspawn. This options correspond to the isolation level offered by Docker.
- **kvm:** It is a full isolated environment implemented through the Kernel Virtualization Module (kvm) of the Unix kernel.

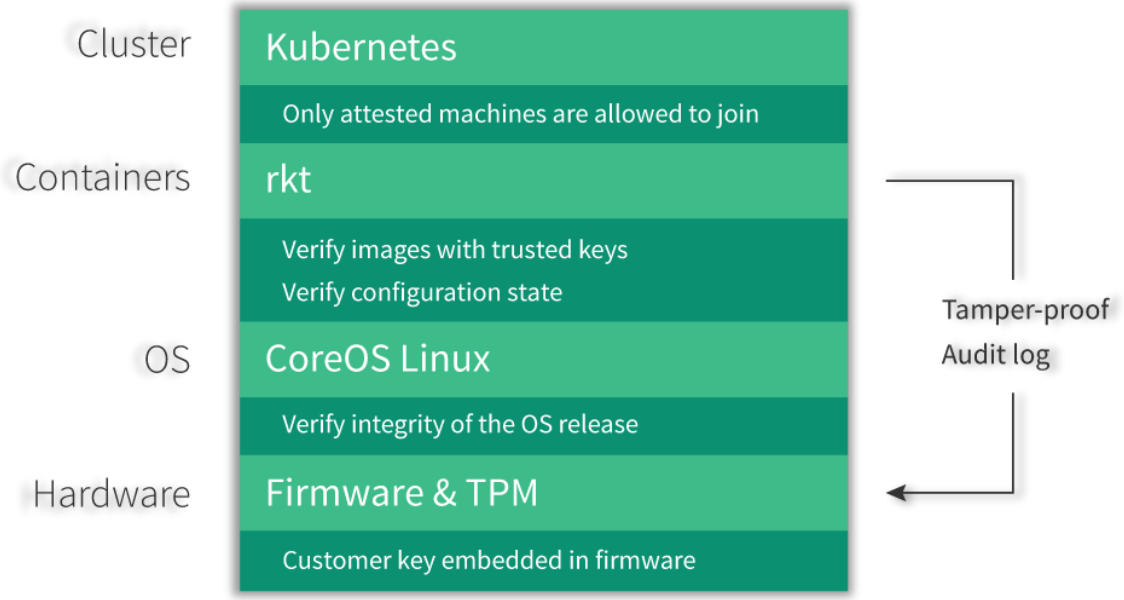
Core Os & rkt

Rocket – Intel
Clear Container

By choosing the highest isolation level **(kvm) Core Os will use** a technology called **Intel Clear Containers**. It was initially **developed by Intel** with the Intel-VT standard as a requirement, but **now is supported** also **by Core Os** in the rkt container engine with no hardware requirement. The basic idea is to **optimize a standard hypervisor** (qemu in the Intel version, kvm in Core Os) **in order to reduce its memory footprint and startup time**. This is done by optimizing both the kernel and the filesystem and by exploiting the direct access feature available in Unix kernel v4.0+. The result is a Virtual Machine that share all the isolation benefit of an hypervisor, but it is close to the startup time and memory utilization of a container.

Core Os & rkt

Core Os – Trusted Computing



https://coreos.com/blog/coreos-trusted-computing.html

Core Os provide **Trusted Computing** by using **Kubernetes**. It will allow a node to join the cluster if and only if its integrity status is verified. The verification of a node start from its boot phase by verifying the integrity status of the hardware and of all the software up to the OS.
Each VM image contains a **signature** that is checked with its configuration state by rkt when it must be loaded into memory.

Core Os & rkt

Image integrity & signature

The default image format used by **Rocket** is **Appc**, and it will include an image signature useful for checking the image integrity at **load time**. This feature is enabled by default and will allow rkt to run only verified images. A similar feature is available with **Docker** by using the **Docker Content Trust (DCT).** No other integrity check mechanism are available.

SECURED

Introduction

SECURED is a European funded reaserch project with the aim to create a **trusted** and **secure execution environment** by taking all the security application into the **Network Edge Device (NED).** Since the NED is a critical node, it must be trusted and this is done by using a modified version of the **Open Attestation Toolkit (OAT)** framework and the **Integrity Module Architecture (IMA)** available in the linux kernel.

SECURED

Features

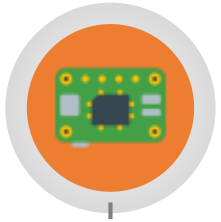
The **key features** of the **SECURED** platform are:

Docker

It enables the Remote Attestation process on Docker

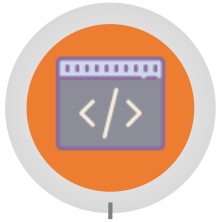
Scalability

The solution is designed by taking into account large scale of operations



Hardware

Doesn’t require particular hardware except for a TPM v1.2

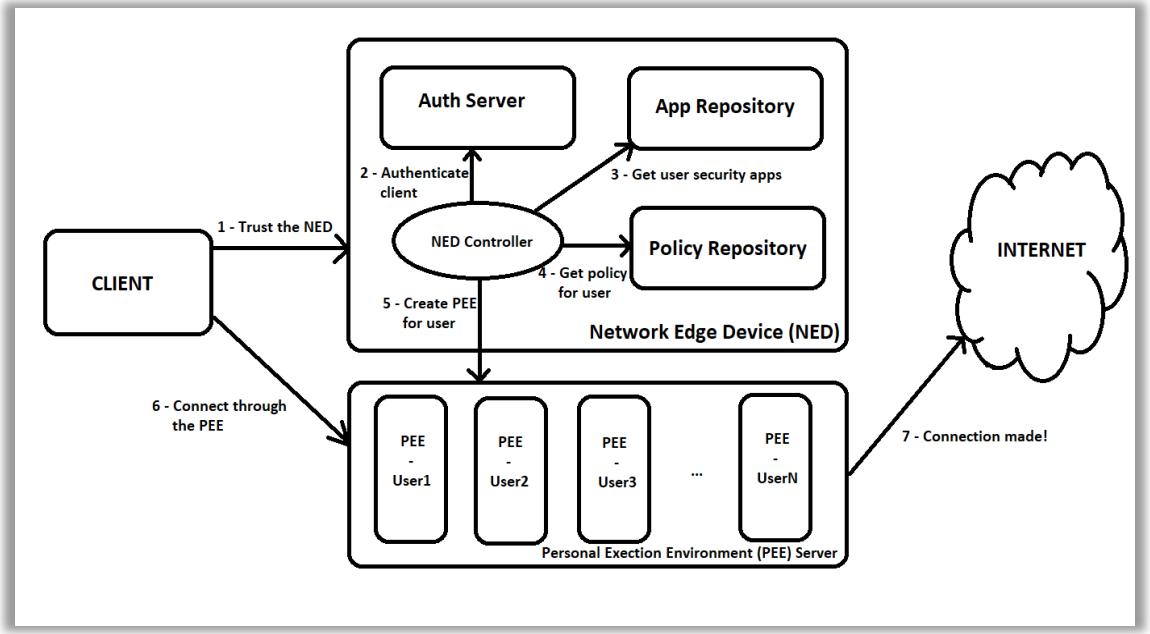


Open Source

All the code is publicly available through github

SECURED

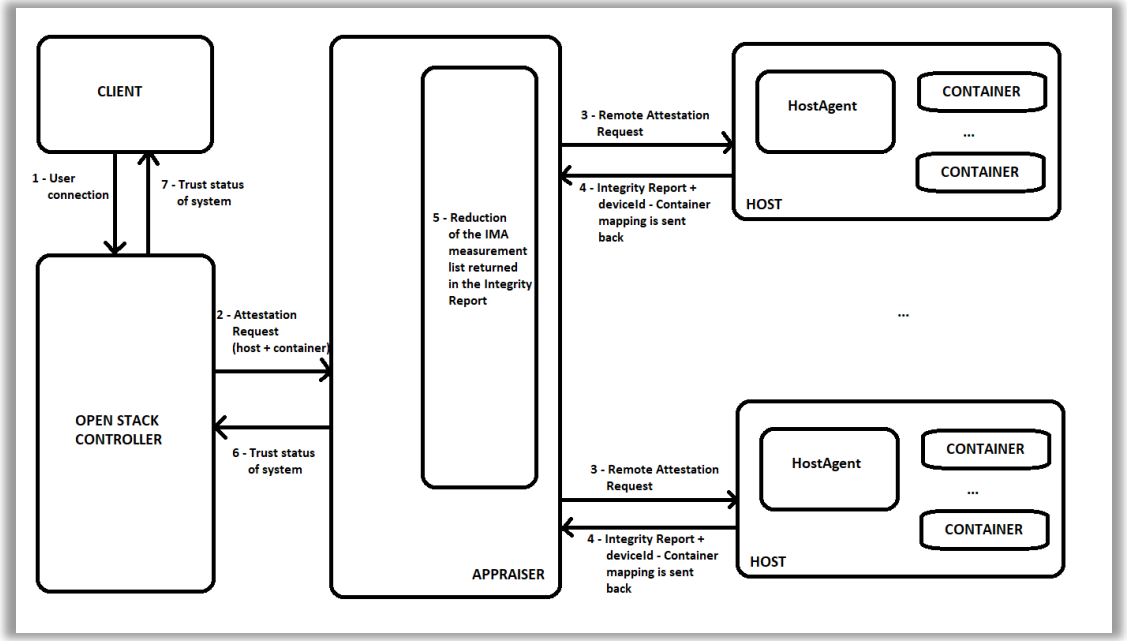
Architecture



The key module of the **SECURED** project is the **Network Edge Device (NED)**. It allows to **authenticate** a user and then to create a **Personal Execution Environment (PEE)** from which the user can **connect safely** to the network. Each PEE consists of the set of **Personal Security Applications (PSA)** needed by the user and runs using the **Network Function Virtualization (NFV)** technology inside a **Docker** container.

SECURED

Remote Attestation



Since the **NED** is a critical module, it must be **trusted**. For that reason a **Remote Attestation (RA)** procedures is put in place. The main components are:

- **HostAgent**: Is the client software that runs on each host and create the **IR**
- **Appraiser**: Is the central server that makes the RA request and check the returned IR with the **whitelist database**.

SECURED

Image Integrity & signature

Image signature can be enabled in docker and it is **handled by the Docker Content Trust (DCT)** mechanism.

Image integrity is enforced at **boot time** with the **DCT** mechanism and at **run time** by **measuring all the files whenever they are loaded into memory**. At any time the Appraiser can request an IR to the host and check that the files loaded into memory are all contained in the whitelist.

SECURED

Docker Content Trust

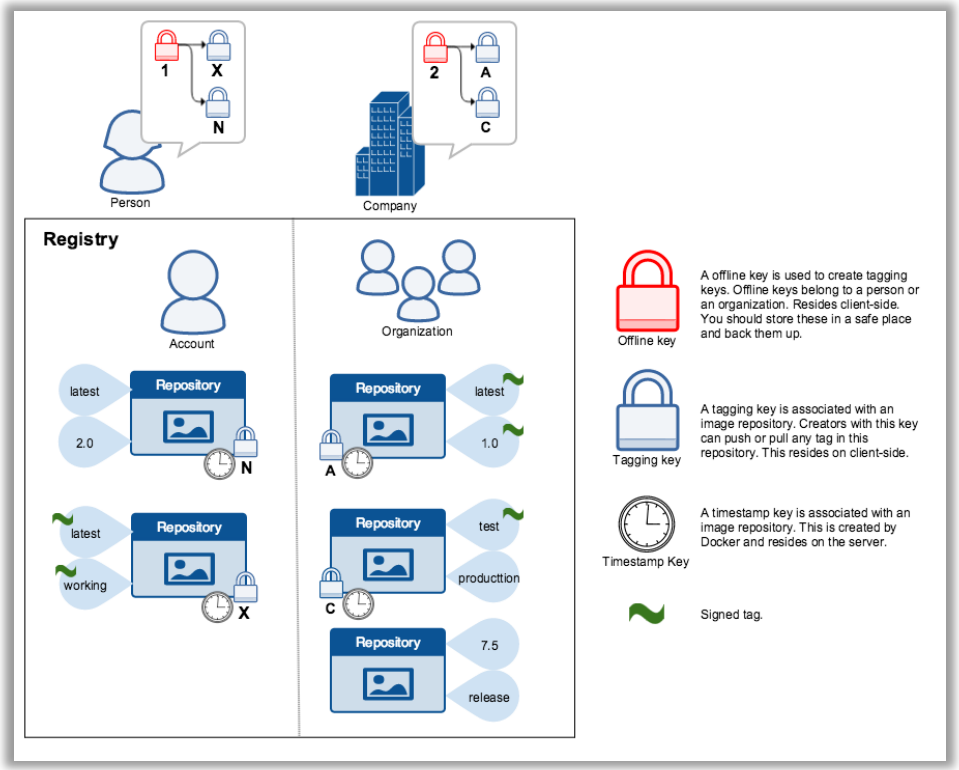


Image signing can be enabled by using the **Docker Content Trust (DCT)** features. It will force docker to run only signed images. The main keys involved in the process are:

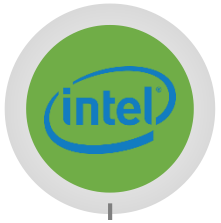
- **Root Key:** root key used to generate the others keys
- **Repository key:** used to sign a particular tag
- **Timestamp key:** used to sign the timestamp of an image in order to guarantee freshness to it

https://docs.docker.com/engine/security/trust/content_trust/#content-trust-operations-and-keys

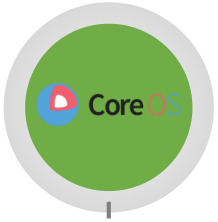
Comparison

Features

Image signature:



Since it supports a wide variety of VM, image signing is delegated to the VM engine used



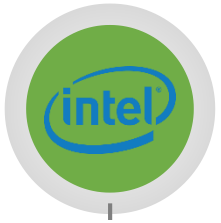
All the image run by rkt are signed by default. By using docker it is possible to sign images using Docker Content Trust

Images can be signed by using the Docker Content Trust feature

Comparison

Features

Image Integrity verification:



Supported only on KVM and Docker. Measure done file by file for all the files found inside the image at load time. No Integrity check at run time

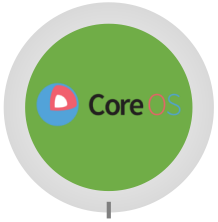


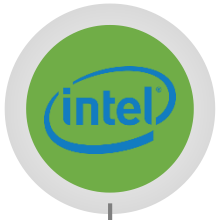
Image integrity is checked only with the image signature at load time. No Integrity check at run time

Integrity checked by exploiting DCT and by measuring all the files for each container when they are loaded into memory

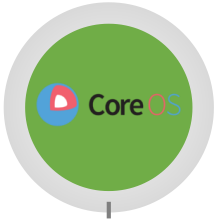
Comparison

Features

Virtualization support:



It support all type of VMs ranging from the standard to the light ones



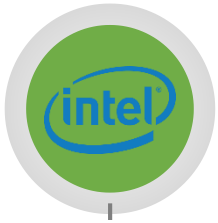
It is based on container, so it support only Docker and the proprietary Rocket container engine

It support only Docker

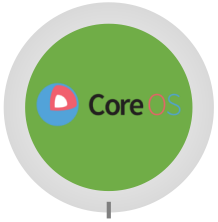
Comparison

Features

Hardware required:



It requires not only a TPM v1.2 but also a compatible Intel processor with the Intel TXT technology



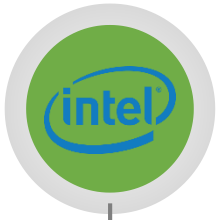
It doesn’t require a particular hardware except for a TPM v1.2 for using the Trusted Computing features

It only requires a TPM v1.2 on the host machines

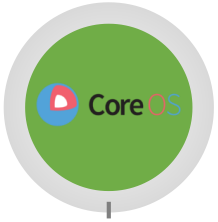
Comparison

Features

Remote Attestation Customization possibility:



Very little configuration options for the RA process



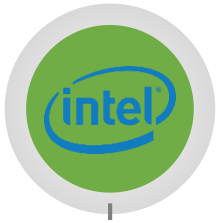
It is not possible to configure the RA process

It allows to select different types of analysis, in order to customize the depth of trustworthiness

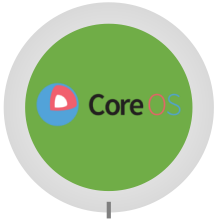
Comparison

Features

Chain of trust:



Starts from the hardware Intel TXT chip, measuring also the BIOS phase



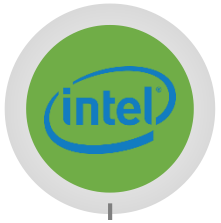
Starts from the standard TPM after the boot phase

Starts from the standard TPM after the boot phase

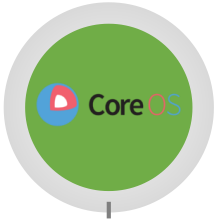
Comparison

Features

Installation process:



Easy to install and operate since all is done through installer



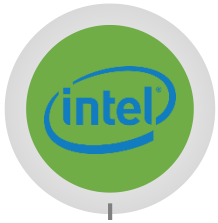
Easy to install and operate since all is done through installer

Quite complex since it requires the compilation of a custom version of the kernel, docker and OAT

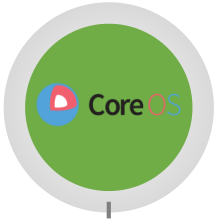
Comparison

Features

Extensibility:



Not possible unless patching the source code available through the Open CIT project

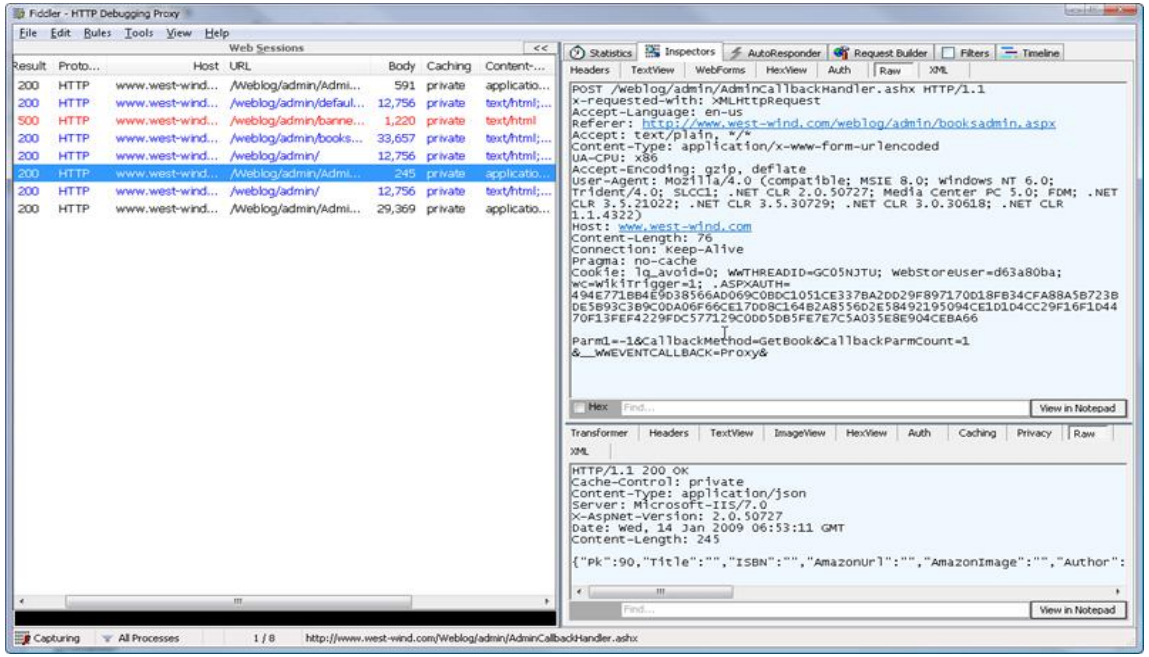


It is possible to develop new plugins using the REST API provided by Core Os

Not much unless patching the source code

Comparison

Fiddler



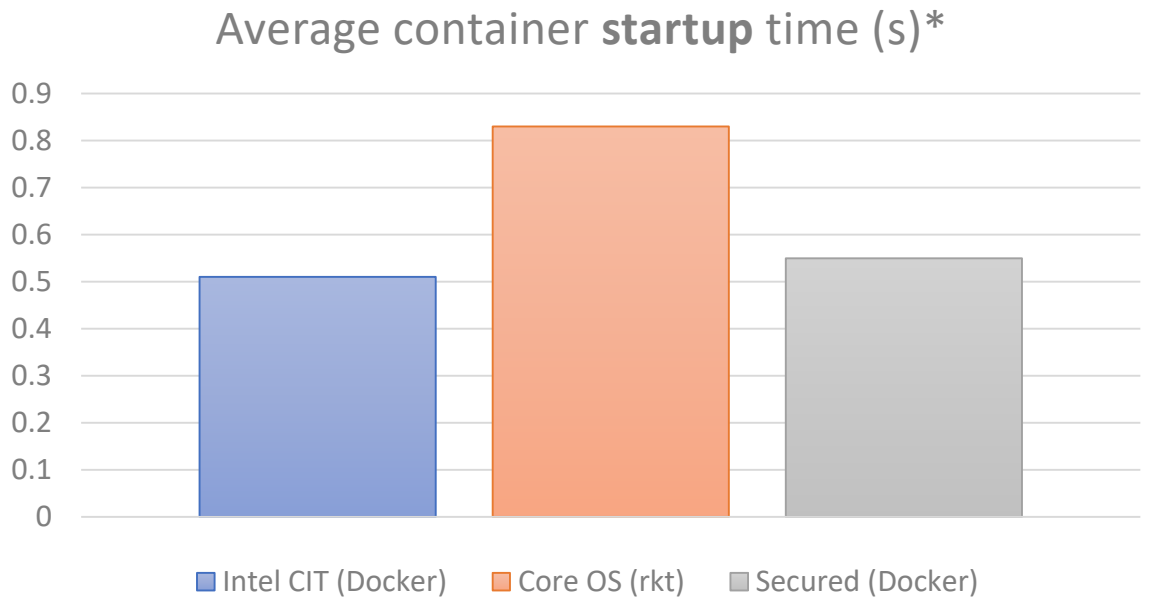
Fiddler is a **web debugging tool** that is able to log all the HTTP/HTTPS traffic that transit on the machine that is installed. In particular it can be used for:

- **Web traffic logging** and monitoring
- **Web application debugging** through manipulation of cookies, headers, cache
- **Calculate** and optimize the **loading times of web pages**
- **Security Testing** through decryption of data and Man In the Middle technique

Comparison

Performance

The **Intel CIT** technology and the **SECURED** project are very **comparable** in terms of container **startup time**, since they are all based on docker. Instead **Core OS** is **slower** since **rkt** needs to load everything every time a new container is started.



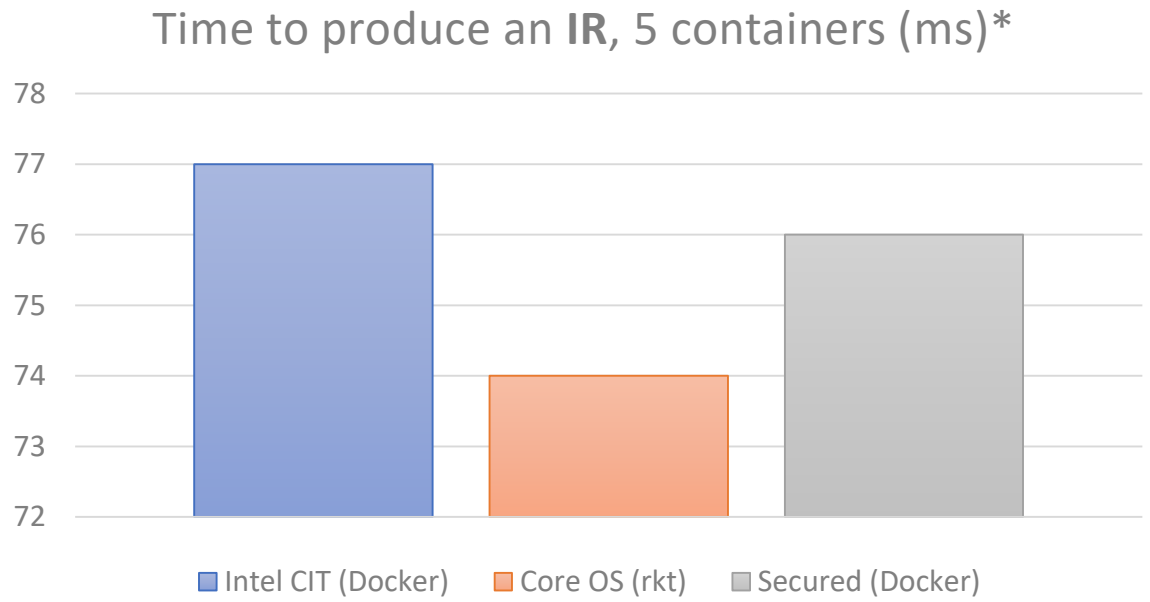
*All the test are done on the following testbench:

- Processor: Core i5 4670 @ 3.8 Ghz
- RAM: 8gb DDR3 @ 1600Mhz
- HDD: 1Tb WD @ 6Gb/s
- GPU: Nvidia GTX 570

Comparison

Performance

The time are taken by putting **fiddler** on the **Appraiser** and measuring the time **from** the **RA request** until the **IR** is **received.** Despite the difference in architecture, **all the solutions performs similarly** when they had to create a new IR report.



*All the test are done on the following testbench:

- Processor: Core i5 4670 @ 3.8 Ghz
- RAM: 8gb DDR3 @ 1600Mhz
- HDD: 1Tb WD @ 6Gb/s
- GPU: Nvidia GTX 570

Thank you for your attention!

Contacts

Fabio Vallone – fabio.vallone@studenti.polito.it