Docker

Container

24.0.6

Integration Guide

CryptoServer HSM

SecurityServer V4.60.0.0

utimaco®

Imprint

Copyright 2026	Utimaco IS GmbH
	Krefelder Straße 220
	52070 Aachen
	Germany
Phone	AMERICAS +1-844-UTIMACO (+1 844-884-6226)
	EMEA +49 800-627-3081
	APAC +81 800-919-1301
Internet	https://support.hsm.utimaco.com/
e-mail	support@utimaco.com
Document Version	1.0.0
Date	2026-07-28
Status	PUBLISHED
Document No.	IG-2026-0086

Table of Contents

1 Introduction

This guide is part of the information and support provided by Utimaco. Additional documentation produced to support your Utimaco SecurityServer product can be found in the document directory of the Utimaco SecurityServer product bundle.

All Utimaco SecurityServer product documentation is available from Utimaco's website at https://utimaco.com/.

1.1 About This Guide

This guide describes how to enable HSM integration with Docker. The instructions in this document have been thoroughly tested and provide a straightforward integration process. There may be other untested ways to achieve interoperability.

1.2 Target Audience for This Guide

This guide is intended for using Docker and HSM administrators.

1.3 Document Conventions

The following conventions are used in this guide:

Convention	Use	Example
Bold	Items of the Graphical User Interface (GUI), e.g., menu options	Select **Details** and click on **Properties** button
`Monospaced`	Code that is given for explanation or as an example, file paths	`certreq.exe -new request.inf IISCertRequest.csr`
Italic	References and important terms	Operating system listed in *Tested Versions*

Table 1: Document conventions

We use special icons to highlight the most important notes and information.

Here you will find important safety information that should be followed.

Here you will find additional notes or supplementary information.

This message marks the result expected after the successful execution of an instruction.

1.4 Abbreviations

The following abbreviations are used in this guide:

Abbreviation	Meaning
API	Application Programming Interface
CD	Compact Disc
CSADM	CryptoServer Command-line Administration Tool
CSAR	Cloud Service Architecture
CNG	Cryptography API Next Generation
CSP	Cryptographic Service Provider
GUI	Graphical User Interface

Abbreviation	Meaning
HSM	Hardware Security Module
ID	Identity
IP	Internet Protocol
JCE	Java Cryptography Extension
LAN	Local area network
PCIe	PCI Express Interface
PKCS#11	Public-Key Cryptography Standard #11
Pseudo TTY	Pseudo Terminals
SDK	Software Development Kit
STDERR	Standard Error
STDOUT	Standard Output
TCP	Transmission Control Protocol
URL	Uniform Resource Locator

Table 2: List of abbreviations

2 Overview

2.1 Docker Minimal Client

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

Utimaco offers a fully functional HSM Software Simulator (SecurityServer Simulator). The SecurityServer Simulator package comes with 100% functional runtime, including all administration and configuration toolsets. The Utimaco SecurityServer Simulator facilitates evaluation, development and integration testing without purchase, delivery, or installation of hardware. For a development teams but even more so for partners' and customers' development teams, the SecurityServer Simulator serves in the process of (application) development. Customers that use the SecurityServer Software Development Kit (SDK) to develop their own firmware module can use the SecurityServer Simulator for testing and validation. Where hardware security modules are integrated into existing IT infrastructure, multiple users can test their developments and corresponding interfaces on the SecurityServer Simulator without affecting production. The SecurityServer Simulator can be used to integrate the HSM with third party applications that provide standardized cryptographic API. These include PKCS#11, CSP/CNG and JCE. The SecurityServer Simulator can be used in plug & play deployments for evaluation of different configuration options, application settings, as well as load-balancing or highavailability scenarios. This guide shows how to build an image for the SecurityServer Simulator. With this Docker image it is simple to start and stop a single instance of a

SecurityServer Simulator or run several SecurityServer Simulator simultaneously as a SecurityServer cluster. Development teams can easily deploy custom firmware modules to the SecurityServer Simulator container without the need to revert the current firmware module configuration manually right after integration or unit testing.

2.2 Utimaco SecurityServer HSM

SecurityServer is a hardware security module developed by Utimaco IS GmbH. SecurityServer is a physically protected specialized computer unit designed to perform sensitive cryptographic tasks and to securely manage as well as store cryptographic keys and data. It can be used as a universal, independent security component for heterogeneous computer systems.

3 Integration Requirements and Prerequisites

Ensure that the system environment you will be using meets the following hardware and software requirements.

This guide assumes that the user has already installed and configured required Software.

3.1 Tested Versions

The integrations that have been successfully tested with the Utimaco HSM with Docker Container.

Base Operating System	Docker Version	Utimaco Security Server Version	Utimaco HSM
RHEL 8	24.0.6	SecurityServer V4.60.0.0.	SecurityServer Simulator V4.60.0.2

Table 3: List of tested versions

3.2 Software Requirements

Software	Software Requirements
HSM Interfaces	SecurityServer PKCS#11 Provider
Utimaco SecurityServer Software	V4.60.0.0
Docker Image Version	24.0.6

Table 4: List of software requirements

Set up an account on the Utimaco support portal and request download access at the following URL https://support.hsm.utimaco.com/.

3.3 Prerequisites

Before you begin, please ensure that you have:

- Installed/set up the operating system listed in *Tested Versions*.
- Installed/set up the SecurityServer listed in *Tested Versions*.
- Familiarized yourself with the docker documentation and setup process and have the Utimaco documentation available.
- Admin access to install docker packages.

4 Install Docker Engine

To install docker engine on your Linux operating system we do refer to the docker documentation website. (Refer Link, *Install Docker Engine on CentOS | Docker Documentation*).

1. Install the latest version of Docker Engine, containerd using below mentioned command.

›_ Console

```
# yum install docker-ce docker-ce-cli containerd.io
```

2. After installation of the docker engine, you can check whether the docker engine is running and set up correctly by starting a hello-world container.

›_ Console

```
# docker run --rm hello-world && docker image rm hello-world
```

```
[root@DockerSS4 bin]# docker run --rm hello-world && docker image rm hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figure 1 : Running Docker command output

If you are running the above and it results in an error, please verify that either you are running the commands as root or your account is a member of the docker group.

5 Creating Docker Image

Docker containers are based on a Docker image. Those images consist of multiple immutable layers of data. The immutable layers are generated on build time. One image can be based on another image. By starting a container, a slim read/write-layer is generated on top of the static image which holds all runtime data. Changing files in the underlying layers will start a copy-on-write action with the new file in the r/w-layer overlaying the older one. Due to this, multiple containers can be based on a single image with minimal footprint. A Docker image is built by using a so called Dockerfile.

5.1 Creating the Dockerfile

A Dockerfile contains plain text instructions on how to build a docker image.

1. Create a directory cssim to the appropriate location and change the directory to `cssim`.

›_ Console

```
# mkdir ~/cssim

# cd ~/cssim
```

2. Create a new file Dockerfile to build our simulator image.

›_ Console

```
# vi Dockerfile
```

3. Copy the following lines into the Dockerfile.

For Redhat UBI Based Docker Image:

Dockerfile

```
FROM redhat/ubi8:latest
LABEL version="4.60.0.0"
maintainer = "support@utimaco.com"
ARG SimulatorPort=3001
RUN dnf install glibc.i686 glibc-devel.i686 libstdc++-devel.i686 -y
WORKDIR /simulator/bin
COPY sim5_linux /simulator
RUN chmod u+x ./bl_sim5
ENV SDK_PORT=$SimulatorPort
EXPOSE $SimulatorPort
STOPSIGNAL SIGINT
ENTRYPOINT ["./bl_sim5"]
CMD ["-o","-h"]
```

For Debian Based Docker Image:

Dockerfile

```
FROM debian:latest
LABEL version="4.60.0.0"
maintainer = "support@utimaco.com"
ARG SimulatorPort=3001
RUN apt-get update; \
apt-get install -y gcc-multilib
WORKDIR /simulator/bin
COPY sim5_linux /simulator
RUN chmod u+x ./bl_sim5
ENV SDK_PORT=$SimulatorPort
EXPOSE $SimulatorPort
STOPSIGNAL SIGINT
ENTRYPOINT ["./bl_sim5"]
CMD ["-o","-h"]
```

Change the SecurityServer Simulator version="4.60.0.0" appropriately.

As you can see from redhat/ubi8:latest that we have derived our image from an official redhat for ubi8 image. Those images are specifically built for containers as they do not contain unnecessary files. Due to the SecurityServer Simulator being an x86 application you need to install some x86 libraries. Similar is the case for Debian image. To connect to the SecurityServer

Simulator we will expose a TCP/IP port. The default port 3001 can be changed by supplying a parameter during the build process of our image. The Dockerfile moreover requires the SecurityServer Simulator files to be present in the project directory.

4. Copy the SecurityServer Simulator files from your product CD. They are located in `<cd>/Software/Linux/Simulator`. Copy the `sim5_linux` folder to the project folder `~/cssim`.

›_ Console

```
# cp -r /media/cdrom/Software/Linux/Simulator/sim5_linux ~/cssim
```

5. After this initial setup your project folder should look like this below.

›_ Console

```
# tree
```

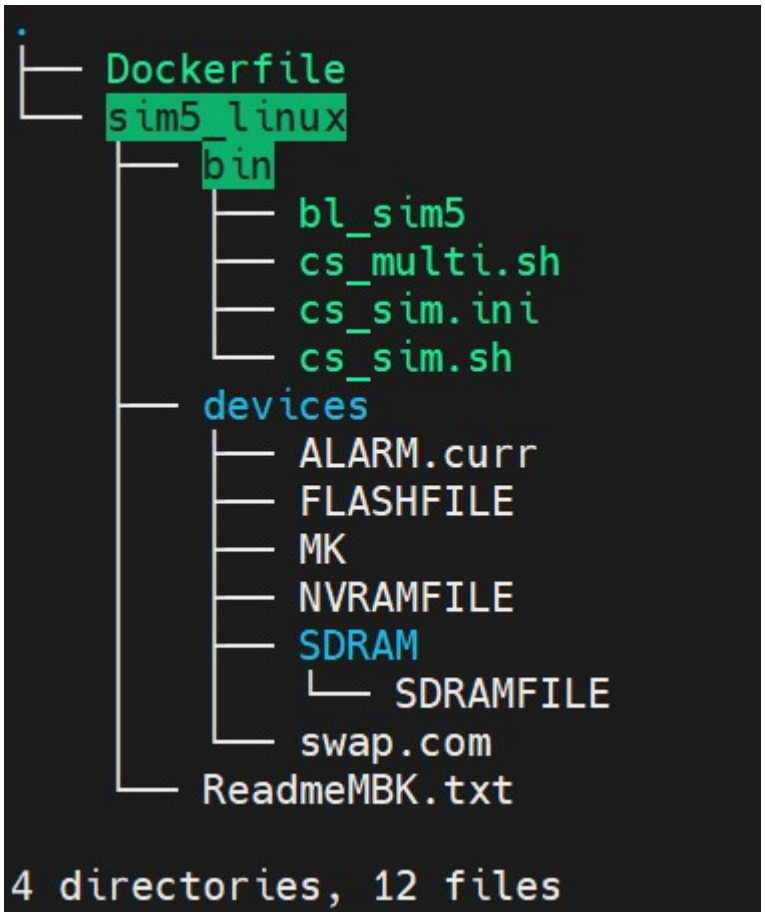

Figure 2 : Tree command output

5.2 Building the Docker Image

To start containers, the user can now build a Docker image that contains the SecurityServer Simulator.

1. Use the docker build command inside of the project directory, created in the previous paragraph. Tag the image with the appropriate version of the SecurityServer Simulator. In this document we use version 4.60.0.0 as it represents the current product CD version.

›_ Console

```
# cd ~/cssim

# docker build -t simulator:4.60.0.0 .
```

```
[root@DockerSS4 cssim]# docker build -t simulator:4.60.0.0 .
[+] Building 33.7s (10/10) FINISHED                                                                                  docker:default
 => [internal] load build definition from Dockerfile                                                                          0.2s
 => => transferring dockerfile: 458B                                                                                          0.0s
 => [internal] load .dockerignore                                                                                             0.2s
 => => transferring context: 2B                                                                                               0.0s
 => [internal] load metadata for docker.io/redhat/ubi8:latest                                                                 1.7s
 => [1/5] FROM docker.io/redhat/ubi8:latest@sha256:a7143118671dfc61aca46e8ab9e488500495a3c4c73a69577ca9386564614c13           6.4s
 => => resolve docker.io/redhat/ubi8:latest@sha256:a7143118671dfc61aca46e8ab9e488500495a3c4c73a69577ca9386564614c13           0.1s
 => => sha256:a7143118671dfc61aca46e8ab9e488500495a3c4c73a69577ca9386564614c13 1.47kB / 1.47kB                                0.0s
 => => sha256:0a342233b8a501dc2e46b943ad75bedb396ff6bc27dfc02665fd2014ebd87f8d 429B / 429B                                    0.0s
 => => sha256:817f060b4672f886292b297d96d2288dec751013210f35a4c89cd9499866e7a5 6.42kB / 6.42kB                                0.0s
 => => sha256:0fa65fe5c23e8b1745b1f39aa3735f2f3ce77cad9e470bfbb1468cb45a886bbe 79.32MB / 79.32MB                              2.7s
 => => extracting sha256:0fa65fe5c23e8b1745b1f39aa3735f2f3ce77cad9e470bfbb1468cb45a886bbe                                     3.0s
 => [internal] load build context                                                                                             0.9s
 => => transferring context: 69.35MB                                                                                          0.7s
 => [2/5] RUN dnf install glibc.i686 glibc-devel.i686 libstdc++-devel.i686 -y                                                22.1s
 => [3/5] WORKDIR /simulator/bin                                                                                              0.2s
 => [4/5] COPY sim5_linux /simulator                                                                                          1.2s
 => [5/5] RUN chmod u+x ./bl_sim5                                                                                             0.5s
 => exporting to image                                                                                                        1.1s
 => => exporting layers                                                                                                       1.1s
 => => writing image sha256:7f457cd7aa48080b001cb32b1ff615773d0de05e3872141aaf53b3ced2aad3c6                                  0.0s
 => => naming to docker.io/library/simulator:4.60.0.0                                                                         0.0s
[root@DockerSS4 cssim]#
```

Figure 3 : Docker build command output for Redhat ubi8 image

```
[root@DockerSS4 cssim]# docker build -t simulator:4.60.0.0 .
[+] Building 55.5s (10/10) FINISHED                                                                                  docker:default
 => [internal] load build definition from Dockerfile                                                                          0.2s
 => => transferring dockerfile: 439B                                                                                          0.0s
 => [internal] load .dockerignore                                                                                             0.2s
 => => transferring context: 2B                                                                                               0.0s
 => [internal] load metadata for docker.io/library/debian:latest                                                              2.8s
 => [1/5] FROM docker.io/library/debian:latest@sha256:7d3e8810c96a6a278c218eb8e7f01efaec9d65f50c54aae37421dc3cbeba6535       4.8s
 => => resolve docker.io/library/debian:latest@sha256:7d3e8810c96a6a278c218eb8e7f01efaec9d65f50c54aae37421dc3cbeba6535       0.1s
 => => sha256:7d3e8810c96a6a278c218eb8e7f01efaec9d65f50c54aae37421dc3cbeba6535 1.85kB / 1.85kB                                0.0s
 => => sha256:22cc4de537485807b7efe6f4c942d7460c4482852f49434f9c022c044c545a90 529B / 529B                                    0.0s
 => => sha256:676aedd4776fe4617839f2517c46c44e8da9091d1c81a01f315101df51a2097f 1.46kB / 1.46kB                                0.0s
 => => sha256:0a9573503463fd3166b5b1f34a7720dac28609e98d731e50e7068f92e79b8545 49.58MB / 49.58MB                              2.3s
 => => extracting sha256:0a9573503463fd3166b5b1f34a7720dac28609e98d731e50e7068f92e79b8545                                     1.8s
 => [internal] load build context                                                                                             0.5s
 => => transferring context: 69.35MB                                                                                          0.3s
 => [2/5] RUN apt-get update; apt-get install -y gcc-multilib                                                                43.2s
 => [3/5] WORKDIR /simulator/bin                                                                                              0.2s
 => [4/5] COPY sim5_linux /simulator                                                                                          1.2s
 => [5/5] RUN chmod u+x ./bl_sim5                                                                                             0.5s
 => exporting to image                                                                                                        2.3s
 => => exporting layers                                                                                                       2.3s
 => => writing image sha256:daa166b5d8ef58db8d417c2dae1498a8001458aad3a08aec0af37464e923b302                                  0.0s
 => => naming to docker.io/library/simulator:4.60.0.0                                                                         0.0s
[root@DockerSS4 cssim]#
```

Figure 4 : Docker build command output for Debian image

If you receive an error message on build you may check that you have setup the project folder as described in the previous paragraph. Also, you may check the connection to the yum repository as it is required to install the necessary additional packages.

2. Verify that the Docker image is created successfully using below command.

›_ Console

```
# docker image ls
```

```
[root@DockerSS4 cssim]# docker image ls
REPOSITORY    TAG         IMAGE ID       CREATED          SIZE
simulator     4.60.0.0    7f457cd7aa48   35 seconds ago   372MB
hello-world   latest      9c7a54a9a43c   5 months ago     13.3kB
[root@DockerSS4 cssim]#
```

Figure 5 : Listing docker images

6 Using the SecurityServer Simulator Image

Now that you have created successfully a docker image for the SecurityServer Simulator, you can start a SecurityServer Simulator container. After you have started the container, you can then verify the functionality of the SecurityServer Simulator.

6.1 Launching a SecurityServer Simulator Container

Start an instance of SecurityServer Simulator docker image as a named container.

›_ Console

```
# docker run -dt --name cssim460 simulator:4.60.0.0
```

```
[root@DockerSS4 cssim]# docker run -dt --name cssim460 simulator:4.60.0.0
0af910db8ad69661e1353c86246aaf06be9dd6adf13210ce3d879296d3faeceb
[root@DockerSS4 cssim]#
```

Figure 6 : Docker run command output

Let us break down the command.

- docker run: will create and start a container in the same step.
- -dt: will run the container in the background (detached) and allocate a pseudo TTY for the process.
- --name cssim: will make the container accessible by the name in addition to the container ID.
- simulator:4.60.0.0 : tells docker to use the image tagged with 4.60.0.0.

6.2 Checking the Running Status of the Container

›_ Console

```
# docker container list
```

```
[root@DockerSS4 cssim]# docker container list
CONTAINER ID   IMAGE                COMMAND             CREATED         STATUS         PORTS      NAMES
ce6c7aa23882   simulator:4.60.0.0   "./bl_sim5 -o -h"   5 minutes ago   Up 5 minutes   3001/tcp   cssim460
[root@DockerSS4 cssim]#
```

Figure 7 : Listing of docker container

6.3 Viewing the Container Output

When the container was started with a pseudo TTY allocated, you can view the output of the SecurityServer Simulator. Therefore, we use the docker logs command. It shows all the output of the container process made to STDOUT and STDERR. You could also directly attach to the container although this is not recommended.

A newly created simulator container without modifications will have an output similar to the next console output. If the build was successful, you now have a docker image of the SecurityServer Simulator available to your local docker installation. You can view the image with the docker image command.

›_ Console

```
# docker logs cssim460
```

```
[root@DockerSS4 cssim]# docker logs cssim460
Utimaco CryptoServer Simulator HSD process started


23.10.12 09:06:38 SMOS SDK Ver. 5.6.10.0 (Sep  5 2023) started [0]
23.10.12 09:06:38 Compiler Ver. 4.8.5
23.10.12 09:06:38 CPU clock frequency: 1000000000
23.10.12 09:06:38 Devices directory: '/simulator/devices'
23.10.12 09:06:38 Sensory Controller Ver. 2.0.0.42 [0/0]
23.10.12 09:06:38 Real Random Number Generator initialized with:
  RESEED_INTERVAL = 1000
  PREDICTION_RESISTANCE = 0
  REALRANDOM_SHARE = 3
23.10.12 09:06:38 Pseudo Random Number Generator initialized with:
  RESEED_INTERVAL = 1000
  PREDICTION_RESISTANCE = 0
  REALRANDOM_SHARE = 0
23.10.12 09:06:38 Load module 'adm.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/adm.so) returned: -1
23.10.12 09:06:38 Load module 'cmds.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/cmds.so) returned: -1
23.10.12 09:06:38 Load module 'crypt.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/crypt.so) returned: -1
23.10.12 09:06:38 Load module 'cxi.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/cxi.so) returned: -1
23.10.12 09:06:38 Load module 'db.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/db.so) returned: -1
23.10.12 09:06:38 Load module 'mbk.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/mbk.so) returned: -1
23.10.12 09:06:38 Load module 'ntp.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/ntp.so) returned: -1
23.10.12 09:06:38 Load module 'oscca.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/oscca.so) returned: -1
```

Figure 8 : Container logs

```
23.10.12 09:06:38 Load module 'pp.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/pp.so) returned: -1
23.10.12 09:06:38 Load module 'sc.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/sc.so) returned: -1
23.10.12 09:06:38 Load module 'stun.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/stun.so) returned: -1
23.10.12 09:06:38 Load module 'util.msc' from FLASHFILE
23.10.12 09:06:38 remove(/simulator/devices/SDRAM/util.so) returned: -1
23.10.12 09:06:38 module 0x83 (CMDS) initialized successfully
23.10.12 09:06:38 module 0x86 (UTIL) initialized successfully
23.10.12 09:06:38 module 0x9f (CRYPT) initialized successfully
23.10.12 09:06:38 module 0x88 (DB) initialized successfully
23.10.12 09:06:38 module 0x8a (STUN) initialized successfully
23.10.12 09:06:38 PP: Setting PIN pad type to AUTO1
23.10.12 09:06:38 module 0x82 (PP) initialized successfully
23.10.12 09:06:38 module 0x85 (SC) initialized successfully
23.10.12 09:06:38 module 0x96 (MBK) initialized successfully
23.10.12 09:06:38 module 0x69 (MBK_EI) initialized successfully
23.10.12 09:06:38 module 0x68 (CXI) initialized successfully
23.10.12 09:06:38 module 0xa1 (OSCCA) initialized successfully
23.10.12 09:06:38 module 0x87 (ADM) initialized successfully
23.10.12 09:06:38 module 0x9a (NTP) initialized successfully
[root@DockerSS4 cssim]#
```

Figure 9 : Container logs

6.4 Get the IP address of SecurityServer Simulator

When the container is running properly you can connect to this simulator using the csadm console application. To do this you need to get the current IP address of the running container.

1. To determine the IP address, use the docker inspect command.

›_ Console

```
# docker inspect cssim460 | grep -m 1 \"IPAddress\"
```

```
[root@DockerSS4 cssim]#
[root@DockerSS4 cssim]# docker inspect cssim460 | grep -m 1 \"IPAddress\"
            "IPAddress": "172.17.0.2",
[root@DockerSS4 cssim]#
```

Figure 10 : Docker inspect command output

2. Now you can connect to the SecurityServer Simulator via this IP address using csadm. You can run a csadm GetState command to ensure the basic connection is successful and retrieve some status information from the SecurityServer Simulator.

›_ Console

```
#cd /opt/utimaco/bin/

#./csadm Dev=3001@172.17.0.2 GetInfo
```

```
[root@DockerSS4 bin]# ./csadm Dev=3001@172.17.0.2 GetInfo
model 5
slot  sim-linux
batt  ok
[root@DockerSS4 bin]#
```

Figure 11 : CSADM command output

6.5 Connecting to the Container

When the container was created in above steps, you can connect to the container using the docker command along with the ID of that container.

›_ Console

```
# docker exec -i -t cssim460 sh

# pwd

/simulator/bin

# ls -ltr
```

```
[root@DockerSS4 bin]# docker exec -it cssim460 sh
sh-4.4# pwd
/simulator/bin
sh-4.4# ls -ltr
total 128
-rwxrwxrwx. 1 root root     70 Oct 11 12:18 cs_sim.sh
-rwxrwxrwx. 1 root root      0 Oct 11 12:18 cs_sim.ini
-rwxrwxrwx. 1 root root   1448 Oct 11 12:18 cs_multi.sh
-rwxrwxrwx. 1 root root 120792 Oct 11 12:18 bl_sim5
sh-4.4#
```

Figure 12 : Connecting the container

6.6 Stop the SecurityServer Simulator Container

When the container is no longer needed, it can be removed via the docker container `rm` command.

›_ Console

```
# docker container stop cssim460
```

```
[root@DockerSS4 bin]# docker container stop cssim460
cssim460
[root@DockerSS4 bin]#
```

Figure 13 : Stopping the container

7 Troubleshooting

Error	Diagnosis
docker run -dt --rm --name cssim simulator:4.60.0.0 Unable to find image 'simulator:4.60.0.0' locally docker: Error response from daemon: pull access denied for simulator, repository does not exist or may require 'docker login': denied: requested access to the resource is denied. See 'docker run --help'.	Make sure that the image with the specified tag exist. Run docker images to list the images with tag value.
./p11tool2 slot=0 LoginUser=ask PubKeyAttr=CKA_LABEL="TestRSAKey" PrvKeyAttr=CKA_LABEL="TestRSAKey" GenerateKeyPair=RSA LoginUser= failed: 12.10.2023 06:04:14 src/p11adm_R2.c[280] p11_open_session: C_OpenSession returned Error 0x00000005 (CKR_GENERAL_ERROR)	Make sure that simulator is running.

Table 5: List of errors and their diagnoses

8 Further Information

This document forms a part of the information and support which is provided by Utimaco IS GmbH. Additional documentation can be found on the product CD in the Documentation directory.

All SecurityServer product documentation is also available at the Utimaco IS GmbH website: *https://utimaco.com/*.

9 References

Reference	Title/Company	Document No.
[CSADMIN]	CryptoServer – csadm Manual/Utimaco IS GmbH	2009-0003
[CSTrSh]	CryptoServer Troubleshooting/Utimaco IS GmbH	M011-0008-en

Table 6: List of references

10 Contact and Support Information

You can reach us from Monday to Friday, 09.00 a.m. to 05.00 p.m., Central European Time (CET).

Utimaco IS GmbH
Krefelder Str. 220
52070 Aachen
Germany

RMA Query

If you need to send the device back to Utimaco IS GmbH, please open a new RMA case (Return Merchandise Authorization). We request that you use the following web address. RMA cases cannot be opened by email or phone.

https://support.hsm.utimaco.com/support/rma/new

Other Support Queries

- Mail (preferred contact method)
  support@utimaco.com
  Attach the diagnostic information to your email.
- Web portal
  https://support.hsm.utimaco.com/support/cases/new/
  The diagnostic information will be requested in our response if necessary.
- By phone
  AMERICAS +1-844-UTIMACO (+1 844-884-6226)
  EMEA +49 800-627-3081
  APAC +81 800-919-1301
  The diagnostic information will be requested in our response if necessary.