**OxM Fast Track (FTx)**

**At-Scale Validation Test Content**

**Revision v1.0**

**Intel Taiwan At-Scale Validation Team**

**Legal Disclaimer**

Notice: This document contains information on products in the design phase of development. The information here is subject to change without notice. Do not finalize a design with this information.  
Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at Intel.com, or from the OEM or retailer..  
No computer system can be absolutely secure. Intel does not assume any liability for lost or stolen data or systems or any damages resulting from such losses.  
You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free  
license to any patent claim thereafter drafted which includes subject matter disclosed herein.  
No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.  
The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.  
This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest  
Intel product specifications and roadmaps.  
Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from  
course of performance, course of dealing, or usage in trade.  
Cost reduction scenarios described are intended as examples of how a given Intel- based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances  
will vary. Intel does not guarantee any costs or cost reduction.  
Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or  
configuration may affect your actual performance.  
Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.  
Intel and the Intel logo are trademarks of Intel Corporation in the U. S. and/or other countries.  
\*Other names and brands may be claimed as the property of others.  
Copyright © 2020, Intel Corporation. All Rights Reserved

**ECCN EAR99**

**Intel Confidential. NDA is required for non-Intel distribution**

Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Lean more at intel.com, or from the OEM or retailer.

Software are workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit [www.intel.com/benchmarks](http://www.intel.com/benchmarks).

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Copyright© 2019-2020, Intel Corporation. All rights reserved. Intel, the Intel logo, Pentium, Xeon, Core, VTune, OpenVINO, Cilk, are trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

Contents

[**Revision Control** 5](#_Toc157004474)

[**Document Reviewers/Approvers** 5](#_Toc157004475)

[**1.** **FTx Validation Requirement** 6](#_Toc157004476)

[**1.1** **Test Overview** 6](#_Toc157004477)

[**1.2** **Test Content Summary** 7](#_Toc157004478)

[**1.2.1** **Cycling Test** 8](#_Toc157004479)

[**1.2.2** **Rank Margin Tool (RMT) Validation – Low-Level Data Gathering Tool** 9](#_Toc157004480)

[**1.2.3** **Intel IO Margin Tool (IOMT) Validation – Low-Level Data Gathering Tool** 11](#_Toc157004481)

[**1.2.4** **Memtest/Advance Memtest (enabled in BIOS)** 14](#_Toc157004482)

[**1.2.5** **System Health Check (SHC) Test Tools** 15](#_Toc157004483)

[**1.2.6** **MLC** 20](#_Toc157004484)

[**1.2.7** **iPerf** 21](#_Toc157004485)

[**1.2.8** **FIO** 22](#_Toc157004486)

[**1.2.9** **Stressapptest** 23](#_Toc157004487)

[**1.2.10** **Prime95 (mprime)** 24](#_Toc157004488)

[**1.2.11** **Stress-ng** 25](#_Toc157004489)

[**1.2.12** **SPEC CPU** 26](#_Toc157004490)

[**1.2.13** **RAS - FISHER** 27](#_Toc157004491)

[**1.2.14** **Virtualization** 30](#_Toc157004492)

[**1.2.15** **Software Guard Extensions (SGX)** 34](#_Toc157004493)

[**1.2.16** **Intel Trust Domain Extensions [Intel TDX]** 38](#_Toc157004494)

[**1.2.17** **Eigen** 38](#_Toc157004495)

[**1.2.18** **NSS** 39](#_Toc157004496)

[**1.2.19** **Cloud-hypervisor** 42](#_Toc157004497)

[**1.2.20** **Unixbench** 45](#_Toc157004498)

[**1.2.21** **Firecracker** 46](#_Toc157004499)

[**1.2.22** **Povray** 51](#_Toc157004500)

# **Revision Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Modifier** | **Date** | **Change Summary** |
| V1.0 | ASV TWN | 12/08/2023 | Initial release |
|  |  |  |  |
|  |  |  |  |

# **Document Reviewers/Approvers**

|  |  |  |
| --- | --- | --- |
| **Area/Function** | **Representative** | **Role** |
| Intel ASV TWN Manager | Lin, River | Reviewer |
| Intel ASV Director | Gonzalez, Ricardo A | Approver |

# **FTx Validation Requirement**

This section provides the test overview of each test phase and the summary for each test contents that Intel requests.

## **Test Overview**

The test overview provides the high-level strategy and the reasons of why FTx validation program focuses on the specific test scope, and it acts as a baseline test of FTx validation program that helps OxM understand more detail to explore their own test contents.

**Phase1**

Phase1 of FTx Validation is a type of acceptance test suite run to ensure the SUTs are stable enough for following test contents. Phase1 consists of a set of Burn-in pipeline including cycling test, memory margin test and stress test for Intel silicon.

**Phase2**

Phase2 of FTx Validation is an extension test of Phase1 to ensure the SUTs’ silicon and I/Os stability with longer test duration on 2 test pipelines – Core Pipeline and Platform Content Pipeline that outline the various well-known stressors for CPUs, memories, storage devices and networking devices. The pass rate >90% must be met which is aligned with phase exit criteria. Otherwise, a regression test is required for pre-sightings fixing and verifying.

**Phase3**

Phase3 of FTx Validation includes elements of RAS, virtualization and security in order to ensure SUTs’ availability and reliability across different feature of CPU architectures. The purpose is to validate the specific test contents Intel defines work properly under various test methodologies. Each test content has particular test requirement, steps and criteria in follow sections.

**Extensive Phase**

Extensive Phase of FTx Validation is to perform hybrid test series. These series are like "assembly line" that OxM can reorder all the test contents and add additional well-known 3rd party test tools. Intel provides the reference pipelines named Combo1 to Combo5 for OxM executing to check whether the system works properly under system heavy load conditions. The OxM is responsible for completing all test pipelines Combo1 to Combo5 and providing the test concept, specific test content and test combination for their CSP customers based on what they’ve learned in FTx validation program from this phase.

The High-level concept of Extensive Phase is to:

* System level stress test with heavy loading by running various stress tools in-parallel
* Perform specific test contents with real-world scenarios from open source and proprietary tools
* Customized test contents by OxM
* Hybrid different stress ratio

## **Test Content Summary**

This section indicates the baseline test requirements of FTx validation Intel requests to help the OxM learn the test purpose, test scope and test procedures of each test content. The test methodologies and parameters can be adjusted depending on OxM’s product configuration and system specification after getting aligned with Intel ASV Team.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase1** | | | |
| **Pipeline** | **Test Content** | | **Time** |
| Burn-in Pipeline | Cycling Test | DC Warm Cycling + Sandstone | 25 cycles |
| DC Cold Cycling + Sandstone | 25 cycles |
| RMT and/or IOMT – 5-10 cy in cold cycling | | 10 cycles |
| Memtest (with boot cycles) and/or ​Advance Memtest (enabled in BIOS) | | every cycle |
| Stress (SHC) | Sandstone-release | 1 hour |
| Sandstone-chaos | 2.5 hours |
| TSL | 30 minutes |
| imunch | 1 hour |
| **Phase2** | | | |
| **Pipeline** | **Test Content** | | **Time** |
| Core Pipeline | Sandstone-latest or Sandstone-release | | 6 hours |
| Sandstone-chaos | | 12 hours |
| TSL (DOL Breadth, DOL Depth) | | 12 hours |
| Platform Content Pipeline | MLC | | 1.5 hours |
| iPerf | | 7.5 hours |
| FIO | | 1.5 hours |
| Stressapptest | | 4.5 hours |
| Prime95 | | 4 hours |
| Stressng | | 4 hours |
| SPEC CPU | | 24 hours |
| **Phase3** | | | |
| **Pipeline** | **Test Contenty** | | **Time** |
| Misc. Pipeline | RAS-Stress | | 16 hours |
| Virtualization | | 12 hours |
| Security Flows | | 12 hours |
| **Extensive Phase** | | | |
| **Pipeline** | **Test Content** | | **Time** |
| Combo1 | Eigen + nss +fio + unixbench + sandstone | | 24 hours |
| Combo2 | Firecracker + nss + fio + unixbench + Sandstone-chaos | | 24 hours |
| Combo3 | Nss + fio + cloud-hypervisor+ Povray + sandstone | | 24 hours |
| Combo4 | FIO + MLC + Sandstone | | 24 hours |
| Combo5 | FIO + iPerf + Sandstone | | 24 hours |
| OxM Customized Pipeline | OxM creates their own test contents | | 48 hours |
| Regression Test | Reserve regression test if necessary | | 120 hours |

### **Cycling Test**

In FTx validation, 2 different of power cycling test are required to ensure system stability through system power on and off – DC warm cycling and DC cold cycling.

**System Requirements**

* Tested machine need IPMI command from in-band.
* Tested machine need Redfish Rest API from out-of-band.

**Steps and Procedures**

**Building the container image**

1. Create the Dockerfile for setting up the base image and install ipmitool and curl by package manager (e.g., APT, YUM, DNF …).
2. Setup the environment in the image.
3. Add run-specific.sh to the image.
4. Build the container image base on Dockerfile.
5. Check the docker image on a standalone system as well as on cluster.

**Run script**

1. Execute sandstone test by following the steps in section [System Health Check (SHC) Test Tools](#_System_Health_Check) for 30 mins
2. Perform system DC warm cycling/DC cold cycling after sandstone test completed
3. Repeat steps 1&2 for 25 cycles

**Run Time**

25 cycles

**Setup Time**

1 hour

**Expected Result**

check “REBOOT COUNT:” key word with “qpool-\*” of KIBANA.

### **Rank Margin Tool (RMT) Validation – Low-Level Data Gathering Tool**

The Low-Level Data Gathering Tool for RMT Intel ASV Team recommended is a software that run on the target UUT itself, and it is intended to provide the OxM with risk guidance for the electrical robustness of their platform interconnects. This tool is used to retrieve previously generated margin data from the Rank Margin Tool (RMT) and covers the retrieving of memory margin data from the BDAT.

**System Requirements**

The Low-Level Data Gathering Tool supports the following configurations:

* Target CPU: ICX and SPR CPUs
* Operating System/Kernel: Linux OS installed with kernel 4.15.0 or later version
* Python 3.6.8, 64-bit is required
* Docker for running LLD tool inside a container

**Prerequisites**

* Contact your Intel account Field Application Engineer (FAE) representative for legal license signature process.
* LLD tool is for customer engineering evaluation purpose under Non-Disclosure Agreement (NDA).
* The software license ‘Internal Use License Agreement’ is required at no cost to customer for use tools in production revenue product/platform.
* Once license agreement is signed, the LLD tool will be available thru Intel® Resource & Design Center (RDC).
* LLD Failure logs should be sent along with processor. (i.e., crash dump, generated logs, the number of processor tested/failed, and other info that Intel may request to help Intel find/understand the root cause.)
* Required BIOS Knob Settings
* ICX-SP:

bdatEn=1, AttemptFastBootCold=0, AttemptFastBoot=0, EnableBiosSsaRMT=1, LegacyRmt=0, BiosSsaPerBitMargining=1, EnableBiosSsaRMTonFCB=0, RmtOnColdFastBoot=0

* SPR:

bdatEn=1, EnableRMT=1, RMTPerBitMargining=1, RMTBacksideMargining=1, AttemptFastBoot=0, AttemptFastBootCold=0, EnableRMTonFCB=0, AllowedSocketsInParallel=1

**Steps and Procedures**

Tool Installation

1. Using the latest BKC with the supported CPU on target platform
2. Download the LLD tool from RDC and copy it to any path of the UUT with Linux installed
3. The installation process and tool execution must be ran by root user, the command “sudo su” is required
4. Go to the tool folder where you stored, then execute \*.run file as root
   * ./LLD\_IPSS\_x.x.x\_Linux.run
5. The installation process will ask for accepting Intel license. Press “y” to move forward
6. After the installation is done. The folder names “ipss” must be shown in current directory by command:
   * ls -al

Tool Execution

1. Before running the tool, use the command “source activate” next to activate the Python virtual environment and IPSS commands for access to different functions.
   * linux$ pwd

/usr/local/ipss/bin

* linux$ source activate

(ipss) linux$

1. Run command to read BDAT looking for margin results:
   * sudo ipss runcommand envemory readBdat -t localhost
2. Find margin results, \*\_Margin\_Results.json, on following path: /usr/local/ipss/lib/python3.6/site-packages/evToolsTests/localTestsRepository/evmemory/results/readBdat. Run command to send margin results to Kibana:
   * python3 -m elasticsearch\_loader --ca-certs ca.crt --with-retry --es-host "**Error! Hyperlink reference not valid.**" --http-auth <username>:<password> --index "margin" json --lines \*\_Margin\_Results.json

Building the Container Image

1. Create the Dockerfile for setting up the base OS image based on Tool Installation steps 1 to 6
2. Setup the environment in the image
3. Add run.sh and run-specific.sh to the image
4. Build the container image based on Dockerfile
5. Check the docker image on a standalone system as well as on a cluster
6. Execute docker command which is the same as 2nd step of tool execution section
   * #docker run –privileged -v $(PATH of tool folder)/ipss runcommand evmemory readBdat -t localhost
7. If all required BIOS knobs are set correctly, you can find the output results in the folder you ran

**Run Time**

10 cycles

**Setup Time**

4 hours

**Expected Results**

1. The JSON margin result file will use the format named “results”. The test logs will be stored in the same folder as \*.tsv file under the run date.
   * /usr/local/ipss/lib/python3.6/site-packages/evtoolsTests/localTestsRepository/evmemory/results/readBdat/
2. Check the margin result manually to see if margin data can be met the criteria. If not, a pre-sighting needs to be submitted.
3. The test logs can be used for debugging purpose as needed.

### **Intel IO Margin Tool (IOMT) Validation – Low-Level Data Gathering Tool**

The Low-Level Data Gathering Tool for IOMT Intel ASV Team recommended is a software that run on the target SUT. It is intended to provide users a guide for the electrical robustness of the interconnect of their SUTs. This tool is used to retrieve high speed interface margin data.

**System Requirements**

The Intel IO Margin Tool supports the following configurations:

* Target CPU: ICX and SPR CPUs
* Linux OS installed with kernel 5.0 or later
* Python 3.6.8, 64-bit is required
* Docker engine installed on the SUTs
* At least 50 GB of free space

**Prerequisites**

* Contact your Intel account Field Application Engineer (FAE) representative for legal license signature process.
* IOMT is for customer engineering evaluation purpose under Non-Disclosure Agreement (NDA).
* The software license ‘Internal Use License Agreement’ is required at no cost to customer for use tools in production revenue product/platform.
* Once license agreement is signed, the IOMT tool will be available thru Intel® Resource & Design Center (RDC).
* IOMT Failure logs should be sent along with processor. (i.e., crash dump, generated logs, the number of processor tested/failed, and other info that Intel may request to help Intel find/understand the root cause.)
* Required BIOS Knob Settings
* ICX-SP:  
  All HSIO Margining
* EDKII Menu > Socket Configuration > IIO Configuration > IIO DFX Configuration > EV DFX Features > Set to: Enable

PCIe/DMI Margining

* EDKII Menu > Socket Configuration-> IIO Configuration -> Socket 0 Configuration -> Port 0/DMI -> PCI-E ASPM Support = Disable
* EDKII Menu > Socket Configuration > IIO Configuration > IIO-PCIE Express Global Options > PCI-E ASPM Support > Set to: No

PCH Uplink Margining

* EDKII Menu > Socket Configuration-> IIO Configuration -> Socket x Configuration -> Port xx -> PCI-E Port = Yes

PCH Margining

* EDKII Menu > Socket Configuration -> Uncore Configuration -> Uncore General Configuration -> Link L0p Enable -> Set to: Disable
* EDKII Menu > Socket Configuration -> Uncore Configuration -> Uncore General Configuration -> Link L1 Enable -> Set to: Disable

CPU PCIe/DMI/UPI Margining

* EDKII Menu > Platform Configuration > System Event Log > IIO Error Enabling > IIO/PCH Global Error Support > Set to: Disable

PCH DMI Margining

* EDKII Menu > Platform Configuration > PCH Configuration > PCI Express Configuration > PCH DMI ASPM > Set to: Disable

PCH SATA Margining

* EDKII Menu > Platform Configuration > PCH Configuration > PCH SATA Configuration > SATA test mode > Enable
* EDKII Menu > Platform Configuration > PCH Configuration > PCH sSATA Configuration > SATA test mode > Enable

PCH HSIO Margining

* EDKII Menu > Platform Configuration > PCH Configuration > PCH DFX Configuration > Reveal PCH P2SB device > Set to: Enable
* EDKII Menu > Platform Configuration > PCH Configuration > PCH DFX Configuration > Unlock PCH P2SB device > Set to: Enable
* SPR:

EV DFX Features is needed.

* Socket Configuration -> IIO Configuration -> IIO DFX Configuration -> EV DFX Features -> Enable
* Platform Configuration -> PCH-IO Configuration -> Enable I/O margining -> Enable

Enable UPI Port 3

* Socket Configuration -> Uncore Configuration -> Uncore Per Socket Configuration -> CPU x -> CPU x UPI Port x -> Link Disable = No

If low power management features are enabled:

* Socket Configuration -> Uncore Configuration -> Uncore General Configuration -> Link L0p Enable = Disable
* Socket Configuration -> Uncore Configuration -> Uncore General Configuration -> Link L1 Enable = Disable
* Socket Configuration -> IIO Configuration -> PCI-E ASPM Support (Global) = No
* Socket Configuration -> IIO Configuration -> Socket 0 Configuration -> Port 0/DMI -> PCI-E ASPM Support = Disable
* Platform Configuration -> PCH-IO Configuration -> PCI Express Configuration -> DMI Link ASPM Control = Disable
* Platform Configuration -> PCH-IO Configuration -> PCI Express Configuration -> PCI Express Root Port x -> ASPM = Disable
* Platform Configuration -> PCH-IO Configuration -> PCI Express Configuration -> PCI Express Root Port x -> L1 Substates = Disable

 Avoid seeing system crashes:

* Platform Configuration -> System Event Log -> IIO Error Enabling -> IIO/PCH Global Error Support =  ->Disable

 Windows Server 2019 OS to boot properly

* Socket Configuration -> IIO Configuration -> Intel? VT for Directed I/O (Vt-d) -> Intel VT for Directed I/O -> Disable

 PCH-SATA SQCT

* Platform Configuration -> PCH-IO Configuration -> SATA and RST Configuration -> Controller X SATA And RST Configuration -> SATA Test Mode -> Enabled

**Steps and Procedures**

Build and run the container image

1. Download the IOMT tool from RDC and copy it to any path of the UUT with Linux installed
2. Install the following packages by using the package manager tool (e.g., APT, YUM, DNF) in the host OS:

build-essential software-properties-common debconf-utils libgtk2.0-0 libnotify-dev libgconf-2-4 libnss3 libxss1 libasound2 libxtst6 apt-transport-https ca-certificates wget gcc make zlib1g-dev sudo dmidecode vim unzip python3-pip kmod dos2unix

1. Copy the XXX\_IPSS\_1.X.XX\_Linux.run and the Dockerfile to the same folder.
2. Build the container image by using the following command:   
   docker build --build-arg installer="XXX\_IPSS\_1.X.XX\_Linux" -t iomt-as-container:1.0 .

Tool Execution

1. Run the container image by using:  
   docker run --privileged -v /usr/local/ipss/results:/usr/local/ipss/lib/python3.6/site-packages/evToolsTests/localTestsRepository/iomargin/results iomt-as-container:1.0 <Parameters>   
   Parameters could be the followings:
   * For ICX-SP:  
     <Interface>: ICX-PCIe, ICX-DMI, LBG-DMI, LBG-PCIe, LBG-SATA, LBG-USB
   * For SPR:  
     <Interface>: SPR-PCIe, SPR-DMI, EBG-DMI, EBG-PCIe, EBG-SATA, EBG-USB
   * -r <Number of repetitions>
2. Evaluate the docker image on a standalone system as well as on a cluster
3. Find margin results and run command to send margin results to Kibana:
   * python3 -m elasticsearch\_loader --ca-certs ca.crt --with-retry --es-host "**Error! Hyperlink reference not valid.**" --http-auth <username>:<password> --index "iomt" json --lines <filename>.json

**Run Time**

10 cycles

**Setup Time**

4 hours

**Expected Results**

1. The JSON margin result file will use the format named “results”. The test logs will be stored in the same folder as \*.tsv file under the run date.
   * /usr/local/ipss/results
2. Check the margin result manually to see if margin data can be met the criteria. If not, a pre-sighting needs to be submitted.
3. The test logs can be used for debugging purpose as needed.

### **Memtest/Advance Memtest (enabled in BIOS)**

The BIOS hand some duties to initialize very specific peripheral devices. The system memory initialization is one of the principal diagnostic tests of BIOS during POST which is called Memtest/Advance Memtest.

**System Requirements**

The memtest/advance memtest supports the following configurations:

* Target CPU: ICX and SPR CPUs
* 1 DPC at least

**Prerequisites**

The AMT must be enabled in BIOS. The settings are found in the BIOS, accessed by pressing F2 during system boot.

* Advanced > Memory Configuration > Attempt Fast Boot [Disabled]
* Advanced > Memory Configuration > Attempt Fast Cold Boot [Disabled]
* Advanced > Memory Configuration > MRC Promote Warnings [Disabled]
* Advanced > Memory Configuration > Promote Warnings [Disabled]
* **Advanced > Memory Configuration > MemTest [Enabled]**
* **Server Management > FRB-2 Enable [Disabled]**

The fast boot must be disabled to run AMT. If fast boot is enabled, it will bypass the AMT initialization. Disabling FRB-2 allows the test to complete without triggering the watchdog timeout.

**Steps and Procedures**

MemTest Loops sets the number of times to loop through the memory test. The default value is [1], set the value to [0] will lead MemTest running continuously.

**Run Time**

Run AMT in every system boot cycle

**Setup Time**

0.5 hour

**Expected Results**

When AMT is active, the BIOS POST codes show additional iterations of

0xB0 Detect DIMM population

And repeated iterations of

0xB7 Train DDR4 ranks

0xB0 Detect DIMM population

0xb9 Hardware memory test and init

0xB1 Set DDR4 frequency

When AMT completes all test loops, the POST code changes to

0xBF MRC is done

The system boot time increases around 5 minutes per cycle with AMT which is expected, the boot time is system configuration dependent. If System boot time does not increase, it may be considered AMT is not running normally.

### **System Health Check (SHC) Test Tools**

Tests of SHC find errors by comparing expected results to computed results or comparing results of multiple iterations.  Test logs provide a clear indication of pass or fail (Generated by SHC Wrapper). Detailed error logs are generated if a failure is detected. Updates will be provided as new tests are developed or new processors are introduced. The SHC contains different tests for different purposes including Sandstone, TSL and imunch. Each program can be executed individually or run a series of tests by calling SHC.

* iMunch – Intel designed test-tool aimed at executing randomized verification of floating point (FP) functions. Designed to find Silent Data Errors (SDEs) within vector execution units in a CPU core.
* Sandstone – Intel-designed test framework and content library designed to exercise a wide variety of CPU functions from a software developer’s perspective. Designed to exercise a wide variety of CPU functions, in some cases operating like datacenter workloads.
* TSL – Test Seed Loader - CPU testing and evaluation tool emphasis on floating-point and other arithmetic instructions as well as IA data-paths. Used to detect data miscompare (DMC) issues with an emphasis on floating-point and other arithmetic instructions as well as IA data paths.
* SHC – SHC is a command line tool intended to unify the execution of a set of test tools that can be used to verify  
  processor operation and detect defective parts. SHC behaves as a wrapper for test content which allows the  
  users. Test logs provides a clear indication of PASS/FAIL.

**System Requirements**

* **Supported Operating Systems & Kernel (From SHC Kit v 4.1.9 User Guide – UserGuide.md)**
  + **Ubuntu 18.04/20.04 64-bit**
  + **CentOS 7.7 or newer version**
* **Linux OS Kernel (Refer to latest version UserGuide.md in SHC Kit)**
  + **SHC has been evaluated on those Linux kernel listed in user guide.**
* **Make sure ‘dmidecode’ is installed to obtain DIMM topology**

**Prerequisites**

* **Contact your Intel account Field Application Engineer (FAE) representative for legal license signature process.**
* **SHC is for customer engineering evaluation purpose under Non-Disclosure Agreement (NDA).**
* **The software license ‘Internal Use License Agreement’ is required at no cost to customer for use tools in production revenue product/platform.**
* **Once license agreement is signed, the SHC tools will be available thru Intel® Resource & Design Center (RDC).**
* **SHC Failure logs should be sent along with processor. (i.e., crash dump, generated logs, the number of processor tested/failed, and other info that Intel may request to help Intel find/understand the root cause.)**

**Steps and Procedures**

**Building the SHC Container**

* Download the SHC binary and copy the shc\_unified\_hvm.vx.x.x.tar.xz archive file to a local drive on the Linux system where you intend to build your container
* Clone the ftx-prt-int-cluster repo from *Github website intel provides* to the same Linux system where you will build your containers.
* Update build.sh updating the repository and container name:tag. It is necessary to modify the proper direction in the build.sh where the SHC binary is stored.
* Run ./build.sh passing in the path to the archive file in step 1

**iMunch**

This tool detects processor issues. It runs for a given amount of time (-t #) on all available threads or all threads on a single physical core (-c #)

Commends:

./imunch -t 60 -r 10 -n 10 –iterateCores --checksum

-t: Test time in seconds. (Default 60s)  
-l: Limit per-thread/instruction mismatch report. (Default 1)  
-q: Quiet, minimize diagnostic information displayed.  
-h : Displays usage information and exits

Example for running imunch using SHC container by passing the following parameters:

*docker run -e SANDSTONE\_BIN="/shc-tools/iMunch/run-test.sh" -e SANDSTONE\_ARGS="/tests/imunch 1800" ftx-shc-test:latest*

Sample imunch.base:

*env.JENKINS\_TEST\_TAG="ftx-shc-test:latest"*

*env.JENKINS\_TEST\_LABEL="ftx-shc-test"*

*env.JENKINS\_NODE\_LABEL="ftx-shc-test"*

*env.JENKINS\_TEST\_BINARY=*[*\\/shc-tools\\/iMunch\\/run-test.sh*](file://shc-tools///iMunch///run-test.sh)

*env.JENKINS\_TEST\_ARGS=""*

*env.DRAGON\_TYPE\_ARG=""*

*env.DRAGON\_ARGS\_ARG=""*

*env.RUN\_DRAGON\_ARG=true*

*env.EXTRA\_ARGS\_ARG=""*

*env.JOB\_RUNTIME=5400*

*env.ITERATIONS=2*

**Sandstone**

Sandstone is an Intel-designed test framework and content library designed to test Intel CPU. It is designed to exercise a wide variety of CPU functions from a software developer’s perspective, operating in some cases more like datacenter workloads. Sandstone is fully configurable for a variety of timing and execution modes. It can execute individual tests on specific CPUs and CPU cores or, every core. Tests can be run for many variations of timing and verbosity. Machine Check Exceptions (MCE) and thermal events (Linux-only) – Sandstone periodically checks whether the Linux kernel has detected and logged an MCE or a thermal event. If detected, warnings are generated in the output:  
# WARNING: Machine check exception detected  
# WARNING: Thermal events detected.

Commands:

./sandstone --disable=@locks\_cross\_cacheline --retest-on-failure=0 -vv -e [sandstone\_availabletests\*] -t 5000 (\*sandstone\_availabletests command: sandstone -l)

--retest : When a test fails, automatically re-run the test to help determine if the failure is consistent.  
--temperature-threshold : Specify a minimum temp that the CPU must come down to between tests.  
--dump-cpu-info : Print available CPU info: e.g., package ID, core ID, thread ID, microcode PPIN.  
--test-list-file : Specify the tests to run in a text file. This will run the tests in the order they appear in the  
file also allows vary individual the durations.  
-t <test-time>: Specify the execution time per test for the program in ms. (Default 1000ms)  
-o: Specify output log file’s location  
--disable [test\_name]: disable test. (i.e., ‘read\_eeprom’)  
--help: Output test common commands / options

Example for running Sandstone-release using SHC container by passing the following parameters:

*docker run -e SANDSTONE\_BIN="sandstone" -e SANDSTONE\_ARGS="-vv --beta -T 36000" ftx-shc-test:latest*

Sample sandstone-release.base:

*env.JENKINS\_TEST\_TAG="ftx-shc-test:latest"*

*env.JENKINS\_TEST\_LABEL="ftx-shc-test"*

*env.JENKINS\_NODE\_LABEL="ftx-shc-test"*

*env.JENKINS\_TEST\_BINARY="\\/shc-tools\\/sandstone\\/sandstone"*

*env.JENKINS\_TEST\_ARGS="-vv --beta -T 3600000"*

*env.JOB\_RUNTIME=3600*

Example for running Sandstone-combo (also referred to as sandstone-chaos) using SHC container by passing the following parameters:

*docker run -e SANDSTONE\_BIN="/tests/scripts/run.sh" -e SANDSTONE\_ARGS="" ftx-shc-test:latest*

Sample sandstone-combo.base:

*env.JENKINS\_TEST\_TAG="ftx-shc-test:latest"*

*env.JENKINS\_TEST\_LABEL="ftx-shc-test"*

*env.JENKINS\_NODE\_LABEL="ftx-shc-test"*

*env.JENKINS\_TEST\_BINARY="\\/tests\\/scripts\\/run.sh"*

*env.JENKINS\_TEST\_ARGS=""*

*env.JOB\_RUNTIME=4200*

*env.ITERATIONS=2*

**TSL (Test Seed Loader) – Previous called Dragon on Linux (DoL)**

TSL is an application to validate proper CPU functionality by executing object files (known as “Dragon seeds”) that implement different algorithms to exercise CPU’s IP. TSL has the following characteristics:

* Stand-alone command-line application written in the C++ programming language.
* Does not require elevated privileges and presents results via standard output or local disk files.
* By default, it generates a text-based test report upon completion.
* Multiple command line parameters to modify uses

Commends:

./tslengine -F ./icx\_seeds.txt -n 4 -T2 -a 20 -r 10

-F [--ordered-list-file] : Runs an ordered list of seeds/tests contained in this file path.  
-n [--report] : Runs a seed (1-100) times, no matter if it passes/fails.  
-T [--time] : Display total execution time and time to failure information. T2: print time to failure at end of execution line.  
-a [--alarm] : minimum timeout value in seconds.  
-r [--retry] : Runs a seed (1~100) times if it fails.  
-h [--help] : Display usage options

Example for running TSL using SHC container by passing the following parameters:

*docker run -e SANDSTONE\_BIN="/shc-tools/TSL/tslengine" -e SANDSTONE\_ARGS="-d /shc-tools/TSL/icx\_seeds.txt -a 30 -O" ftx-shc-test:latest*

Sample imunch.base:

*env.JENKINS\_TEST\_TAG="ftx-shc-test:latest"*

*env.JENKINS\_TEST\_LABEL="ftx-shc-test"*

*env.JENKINS\_NODE\_LABEL="ftx-shc-test"*

*env.JENKINS\_TEST\_BINARY="\\/shc-tools\\/TSL\\/tslengine"*

*env.JENKINS\_TEST\_ARGS="-d \\/shc-tools\\/TSL\\/seeds\\/ICX\\/Breadth -a 30 -O "*

*env.TEST\_RUNTIME="10h"*

*env.JOB\_RUNTIME=36000*

**SHC**

Combines iMunch, TSL, and Sandstone test commands into one program with no user selectable options. Test logs provides a clear indication of PASS/FAIL. Detail info will be shared in Module 4.4 System Test Result/Log in this document.

Commands:

./run.sh

(The default option is pre-configured on the entry script *run.sh* located at the top level of the un-tared file)

or

./shc -r TestFlow\_Pre\_Base.txt

(Run recipe mode. Complete test list contented in text file ‘TestFlow\_Pre\_base.txt’)

-a : to store the artifacts in the SHC report directory.

-r : Recipe mode, execute test from a recipe file.  
-c : Execute tests from command line.  
-h : Display help menu.

Example for running the full SHC tests using the SHC container by passing the following parameters:

*docker run -e SANDSTONE\_BIN="/shc-tools/TSL/tslengine" -e SANDSTONE\_ARGS="-d /shc-tools/TSL/icx\_seeds.txt -a 30 -O" ftx-shc-test:latest*

Sample imunch.base:

*env.JENKINS\_TEST\_TAG="ftx-shc-test:latest"*

*env.JENKINS\_TEST\_LABEL="ftx-shc-test"*

*env.JENKINS\_NODE\_LABEL="ftx-shc-test"*

*env.JENKINS\_TEST\_BINARY="run.sh"*

*env.JENKINS\_TEST\_ARGS=""*

*env.DRAGON\_TYPE\_ARG=""*

*env.DRAGON\_ARGS\_ARG=""*

*env.RUN\_DRAGON\_ARG=true*

*env.EXTRA\_ARGS\_ARG=""*

*env.JOB\_RUNTIME=5400*

*env.ITERATIONS=2*

**Run Time**

* Phase1 (Total 5 hours)
  + Sandstone-release: 1 hour
  + Sandstone-chaos: 2.5 hours
  + TSL: 0.5 hour
  + Imunch: 1 hour
* Phase2 (Total 30 hours)
  + Sandstone-release: 6 hours
  + Sandstone-chaos: 12 hours
  + TSL (DOL Breadth, DOL Depth): 12 hours

**Setup Time**

* Phase1: 4 hours
* Phase2: 2 hours

**Expected Results**

Test information captured includes exit codes, standard output, standard error output, TAP failure string search results, start/stop time of each test and artifacts saved in “report.txt / .json”.

SHC summarizes all test results in a section titled OVERALL RESULTS which will either say PASSED, FAILED or INCONCLUSIVE.

* PASSED: No detectable errors or failures were found
* FAILED: A failure string was found in standard output or standard error output of the test.
* INCONCLUSIVE: A non-zero exit code or non-empty stderr was returned by the test

SHC generates one test report (‘shc\_report’) containing all information for tests executed. Test report can be found on the files report.txt & report.json both containing the same information and are generated by default

* shc\_report\_YYYY-MM-DD-hh-mm-ss
* report.json / report.txt
* Tool directories – Inside of each, there’s sub-directory(es) for execution ID (Sub-TEST ID).
* Expected empty folder if not error occurred in testing. Number of folders depends on test list pre-defined in recipe file (i.e., ‘TestFlow\_Pre\_Base.txt’)
* topology.xml

Both ‘shc\_report’ and ‘topology.xml’ are required to be attached/shared with test failed unit and return to Intel through FACR process.

### **MLC**

is a tool used to measure memory latencies and b/w, and how they change with increasing load on the system. It also provides several options for more fine-grained investigation where b/w and latencies from a specific set of cores to caches or memory can be measured as well. Use as a stressor in FTx for system stability evaluation.

**Prerequisites**

* MLC supports running on Linux in cluster environments
* GNU C library (glibc/pthread) must be present on the system
* Root privileges are required to run this tool as the tool modifies the H/W prefetch control MSR to enable/disable prefetchers for latency and b/w measurements.
* MSR driver from the OS should be loaded. This can typically be done with 'modprobe msr' command if it is not already included.

**System Requirement**

* Major Linux Distro with GNU C library(glibc/pthread)

**Steps and Procedures**

**Building the container image**

1. Download the tool binary
2. Copy the tool binary to any directory on the system
3. Check the library dependency on the system
4. Add the run content to the scripts (run.sh and/or run-specific.sh)
5. Write a Dockerfile based on step 1 to 4
6. Build the container images based on the Dockerfile from step 5
7. Evaluate the docker image on a standalone system as well as on a cluster

**Run script**

The concept of running MLC is to use MLC to measure (1) the system memory access time in different patterns (e.g., local memory, remote memory) in different NUMA configs. (2) loaded memory latency (3) max system memory bandwidth.

mlc --loaded\_latency -T -d0 –t180 -u -b6m -I

mlc --loaded\_latency -T -d0 –t180 -I

mlc --loaded\_latency -T -d0 -o/local.txt –t180

mlc --loaded\_latency -T -d0 -o/remote.txt –t180 -I

mlc --loaded\_latency -T -d0 –t180 -u -b6m -I –Z(avx-512)

mlc --max\_bandwidth

**Run Time**

1.5 hours but can be extended based on needed.

**Setup Time**

2 hours

**Expected Results**

* Memory latency in different NUMA configs
* Memory bandwidth in different NUMA configs
* CPU Interconnect bandwidth
* Loaded memory latency

### **iPerf**

a tool for active measurements of the maximum achievable bandwidth on IP networks. It supports tuning of various parameters related to timing, buffers and protocols (TCP, UDP, SCTP with IPv4 and IPv6). For each test it reports the bandwidth, loss, and other parameters. Use as a stressor in FTx for system stability evaluation.

**Prerequisites** iPerf supports running on Linux in cluster environments

* iPerf supports running on Linux in cluster environments
* GNU libraries (libstdc++/libm/libgcc/libpthread/libc) must be present on the system
* A table storing SUT names and IP addressing and can be referenced by SUTs through NFS in runtime

**System Requirement**

* Major Linux Distro with GNU libraries (libstdc++/libm/libgcc/libpthread/libc)
* Network connectivity

**Steps and Procedures**

**Building the container image**

* 1. Install the tool and its related dependency in the Linux container image by using package managers (e.g., APT, YUM, DNF)
  2. Setup the environment in the image
  3. Add the run content (run.sh and run-specific.sh) to the image
  4. Write a Dockerfile based on step 1 to 3
  5. Build the container image
  6. Evaluate the docker image on a standalone system as well as on a cluster

**Run script**

* SUT kicks off client and server iPerf respectively. Send traffic to next SUT to simulate a ring traffic among all SUTs. Example: SUT-01 sends traffic to SUT-02. SUT-02 sends traffic to SUT-03, etc.
* iperf -c "$mypeer" -t 60 -d "$@" &
* Option: send traffic to all SUTs and received traffic from all SUTs

**Run Time**

7.5 hours but can be adjusted based on needed.

**Setup Time**

2 hours

**Expected Results**

SUTs output the corresponding iPerf results with reasonable network throughput values without any error message.

### **FIO**

A tool that would be able to simulate a given I/O workload. Use as a stressor in FTx for system stability evaluation.

**Prerequisites**

* FIO supports running on different Linux distros in cluster environments
* Tons of libraries depend on what IO engine has been used with FIO

**System Requirement**

* Major Linux Distro
* Storage devices on the SUTs

**Steps and Procedures**

**Building the container image**

* 1. Install the tool and its related dependency in the Linux container image by using package managers (e.g., APT, YUM, DNF)
  2. Setup the environment in the image
  3. Add the run content (run.sh and run-specific.sh) to the image
  4. Write a Dockerfile based on step 1 to 3
  5. Build the container image
  6. Evaluate the docker image on a standalone system as well as on a cluster

**Run script**

1. Stress every NVMe disks in a SUT sequentially/in parallel
2. Need to find out unmount disks on a SUT and stress it directly
3. fio $files --name=gen4-bandwidth-workload --bs=1m,4m --numjobs=16 --iodepth=128 --direct=1 --buffered=0 --ioengine=libaio --output-format=json+ --runtime=300

**Run Time**

1.5 hours but can be extended based on needed.

**Setup Time**

2 hours

**Expected Results**

SUTs output the corresponding FIO results with reasonable storage device throughput values without any error message.

### **Stressapptest**

Stressful Application Test (or stressapptest, its unix name) is a memory interface test. It tries to maximize randomized traffic to memory from processor and I/O, with the intent of creating a realistic high load situation in order to test the existing hardware devices in a computer. It has been used at Google for some time. Use as a stressor in FTx for system stability evaluation.

**Prerequisites**

* Stressapptest supports running on Linux in cluster environments
* GNU libraries (libstdc++/libm/libc/libpthread/libaio/librt) must be present on the system

**System Requirement**

* Major Linux Distro with GNU libraries

**Steps and Procedures**

**Building the container image**

* 1. Install the tool and its related dependency in the Linux container image by using package managers(e.g., APT, YUM, DNF)
  2. Setup the environment in the image
  3. Add the run content (run.sh and run-specific.sh) to the image
  4. Write a Dockerfile based on step 1 to 3
  5. Build the container image
  6. Evaluate the docker image on a standalone system as well as on a cluster

**Run script**

* Concept is to use up almost all HW resource within a SUT
* HW thread number and ram size should be adjusted accordingly
* stressapptest -s 20(time) -M 256(ram) -m 8(thread) -W
  + suggest using 80~90% HW thread
  + suggest using 90% memory space

**Run Time**

4.5 hours but can be extended based on needed.

**Setup Time**

2 hours

**Expected Results**

Can successfully run the tool in a given time without any error message from the SUTs

### **Prime95 (mprime)**

Prime95 is a tool for stress / torture testing a CPU. The software makes heavy use of the processor's integer and floating point instructions, it feeds the processor a consistent and verifiable workload to test the stability of the CPU and the L1/L2/L3 processor cache. Additionally, it uses all of the cores of a multi-CPU / multi-core system to ensure a high-load stress test environment. Use as a stressor in FTx for system stability evaluation.

**Prerequisites**

* Prime 95 supports running on major Linux distributions.
* GNU libraries (libstdc++/libm/libc/libpthread/librt/libdl/libgmp/libgcc\_s) must be present on the system

**System Requirement**

Major Linux Distro with GNU libraries

**Steps and Procedures**

**Building the container image**

1. Download the source code of the tool
2. Build the binary for the tool
3. Copy/install the tool binary to a specific folder in a system
4. Check the library dependency on the system
5. Add the run content to the scripts (run.sh and/or run-specific.sh)
6. Write a Dockerfile based on step 1 to 5 for making the tool can run in a container image
7. Build the container images based on the Dockerfile from step 6
8. Evaluate the docker image on a standalone system as well as on a cluster

**Run script**

* + Options -> Torture Test
  + Using all HW threads
  + Using large FFTs might consumes all RAM capacity. Try to use small/smallest FFTs instead.

**Run Time**

4 hours but can be extended based on needed.

**Setup Time**

2 hours

**Expected Results**

Can successfully run the tool in a given time without any error message from the SUTs

### **Stress-ng**

Stress-ng is designed to exercise various physical subsystems of a computer as well as the various operating system kernel interfaces. stress-ng also has a wide range of CPU specific stress tests that exercise floating point, integer, bit manipulation and control flow. The tool has a wide range of different stress mechanisms (known as "stressors") and a full description of these is included in the man page. Use as a stressor in FTx for system stability evaluation.

**Prerequisites**

* Stress-ng supports running on Linux in cluster environments
* GNU libraries (libm/libc/libpthread/libaio/librt/libbsd/libcrypt/libsctp/libdl/libJudy/libapparmor/libIPSec\_MB) must be present on the system

**System Requirement**

* Major Linux Distro with GNU libraries
* Have internet connection with the build environment

**Steps and Procedures**

**Building the container image**

* 1. Install the tool and its related dependency in the Linux container image by using package managers (e.g., APT, YUM, DNF)
  2. Setup the environment in the image
  3. Add the run content (run.sh and run-specific.sh) to the image
  4. Write a Dockerfile based on step 1 to 3
  5. Build the container image
  6. Evaluate the docker image on a standalone system as well as on a cluster

**Run script**

* + Concept is to use up almost all hw resource within a SUT
  + Example: stress-ng --cpu 8 --io 4 --vm 2 --vm-bytes 128M --fork 4 --timeout 10s
  + Normally distribute all HW threads into following workload types
    - -cpu - CPU intensive
    - -cpu-cache - stress CPU instruction and/or data caches
    - -memory - stack, heap, memory mapping, shared memory stressors
    - -vm - Virtual Memory stressor (paging and memory)

**Run Time**

4.5 hours but can be extended based on needed.

**Setup Time**

2 hours

**Expected Results**

Can successfully run the tool in a given time without any error message from the SUTs

### **SPEC CPU**

A benchmark tool contains suites including SPECrate 2017 Integer and SPECrate 2017 Floating Point to measure the system throughput or work per unit of time. Use as a stressor in FTx for system stability evaluation.

**Prerequisites**

* SPEC CPU supports running on Linux in cluster environments
* 32bit runtimes libgcc/libstdc++/glibc
* glibc 2.28+
* GNU LD version 2.27+

**System Requirement**

* License to download latest version SPEC CPU from SPEC.org
* Contact Intel FAE for downloading Intel-build SPEC CPU Binary
* 1.5 GB RAM per rate copy or process launched
* Minimize non-benchmark overhead. Consider dropping to runlevel 3 (noGUI command line with networking)

**Steps and Procedures**

**Building the container image**

1. Download SPEC CPU ISO and mount it
2. Install SPEC CPU to a dedicated folder (e.g, /cpu2017.installed)
3. Untar Intel binary to the dedicated folder
4. Check/install the library dependency on the system
5. Create the Dockerfile for setting up the base OS image based on step 1 to 4
6. Build the container image based on Dockerfile
7. Evaluate the docker image on a standalone system as well as on a cluster

**Run script**

* Use the script provided in Intel-build binary to stress all cores in a SUT.
* Target is to use up all HW threads in int rate run and fp rate run
* Example: reportable-icxxxx-lin-core-avx512-rate-smt-on-xxxxx.sh

**Run Time**

Finish at least 1 cycle of INT rate run and FP rate run

**Setup Time**

2 hours

**Expected Results**

* Can successfully finish the INT rate run and FP rate run without any error message from the SUTs.
* Reasonable INT rate and FP rate scores

### **RAS - FISHER**

Fisher is a tool to help validate the behavior of the RAS features on a server platform. The tool will invoke a workload defined by the user, then inject an error in the physical address occupied by the workload using Linux’s APEI EINJ module and finally verify the error messages logged by the OS. At scale testing of RAS features utilizing Fisher focuses on the system’s handling of memory correctable and memory uncorrectable but non-fatal errors. Different workloads along with different fisher command parameters can be utilized to target various scenarios and instruction sets for a more complete test coverage.

**Prerequisites**

* Download and install Fisher tool available thru **Intel® Resource & Design Center (RDC) as Fault Injection Software Hook Pkg.**
* Installation of the workloads mentioned previously in this document including but not limited to:
  + Sandstone, FIO, Stress-ng, Stressapptest, mprime, ai-benchmark, openssl, povray, y-cruncher, stamp-rtm
* Install mcelog as the Linux error collector (https://mcelog.org/installation.html)
* Setup BIOS knobs and Linux kernel according to Fisher user guide

**System Requirement**

* Setup system configuration as required based on workloads mentioned above
* Recommended to use Linux kernel version 5.11 or above

**Steps and Procedures**

**Building the container image**

1. Install workloads mentioned above in the base OS image
2. Untar Fisher package into base OS image
3. In directory where fisher is located, create one test script for each test case
4. Create a run-all test script calling all test scripts, preferably in a randomized order.
5. Create Dockerfile based on steps 1-4
6. Build the container image based on Dockerfile
7. Evaluate the docker image on a standalone system as well as on a cluster\*

\*Note: when running the ras docker image, several directories need to be mounted and run in privileged mode for fisher to work properly.

Example: *docker run --rm -it --name fisher* ***-v /lib/modules:/lib/modules -v /var/log/journal:/var/log/journal -v /etc/machine-id:/etc/machine-id -v /sys:/sys -v /dev/shm:/dev/shm --privileged*** *ftx-test:ras-latest /tests/scripts/run-all.sh*

**Example Test Cases**

The following test cases are provided as an example of a test suite meant to cover various workloads and scenarios. Test cases and test parameters can be modified as needed by project.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item** | **RAS Workload** | **Runtime (min)** | **Workload Comments** | **Tool** | **Error Type** |
| 1 | fisher --workload="ai-benchmark" --workload="ai-benchmark" --workload="ai-benchmark" --workload="ai-benchmark" \  --workload="ai-benchmark" --workload="ai-benchmark" --workload="ai-benchmark" --workload="ai-benchmark" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DRd --inspect-rate=200us --runtime=20m $@ | 20 | Poison Stress - Artificial Intelligence with Oversuscription Intent: AI benchmark focused on training and inference in an oversubscripted environment | AI Benchmark | uce-non-fatal |
| 2 | fisher --workload="imunch" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DRd --runtime=20m $@ | 20 | Poison Number Crunch Intent: Expectation is to recover system when crossed with a verity of math and vector operations | imunch | uce-non-fatal |
| 3 | fisher --workload="sllr64 -t\$(shuf -i1-$(nproc) -n1) -d -q\"\$(shuf -i10000-800000 -n1)\*\$(shuf -i10000-800000 -n1)^\$(shuf -i1-200000 -n1)-1\"" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DRd --runtime=20m $@ | 20 | Lucas Lehmer Riesel Primality Tests Intent: Poison consumption under random primality tests with different thread counts | sllr64 | uce-non-fatal |
| 4 | fisher --workload="mprime -t" \  --injection-type=memory-correctable:10,memory-uncorrectable-non-fatal \  --match=DRd --runtime=20m $@ | 20 | Uncorrectable and Correctable Memory Mix Intent: Fast memory correctable injection with low probability of memory uncorrectable running memory benchmarks with AVX-512 instructions | mprime | correctable:10, uce-non-fatal |
| 5 | fisher --workload="mprime -t" \  --workload="mlc" \  --workload="stressapptest -M 8192 -F -s 7200 --max\_errors 1" \  --workload="stressapptest -M 8192 -F -s 7200 --max\_errors 1" \  --workload="stressapptest -M 8192 -W -C --cc\_test -s 7200 --max\_errors 1" \  --workload="openssl speed -multi $(nproc) -seconds 1" \  --injection-type=memory-uncorrectable-non-fatal \  --match='DRd\_^(?!mov$)' --runtime=20m $@ | 20 | Poison Supreme Intent: Full stack stressing workload execution while uncorrectable error injection, parallel stress includes: CPU torture, floating-point, integer arithmetic instructions, memory copy, multi-sized memory chunk, cryptographic algorithms, performance tests | mix | uce-non-fatal |
| 6 | fisher --workload="mprime -t" \  --workload="sandstone --total-time=2h -o /dev/null" \  --workload="stressapptest -W -C --cc\_test -s 7200 --max\_errors 1" \  --injection-type=memory-correctable \  --match=DRd --runtime=20m $@ | 20 | DDR5 ECC Correctable Stress Multi Intent: Memory Correctable stress with different memory pressure schemes | mix | correctable |
| 7 | fisher --workload="openssl speed -multi $(nproc) -seconds 1" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DRd --runtime=20m $@ | 20 | OpenSSL Intent: Cryptography workload with poison stress | openssl | uce-non-fatal |
| 8 | fisher --workload="povray --benchmark <<< EOF" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DRd --inspect-rate=200us --workload-exit-code=0,2 \  --injection-delay-max=3s --runtime=20m $@ | 20 | POV-Ray Rendering Intent: Poison handling on ray tracing heavily parallel cacheline data sharing | povray | uce-non-fatal |
| 9 | fisher --workload="sandstone -e \\*aes\\* -T 72h -v -o /dev/null" \  --injection-type=memory-uncorrectable-non-fatal \  --match='DRd\_^(?!mov)' --inspect-rate=200us \  --workload-exit-code=0,1 --runtime=20m $@ | 20 | AES Poison Intent: Varying strength Advanced Encryption Standard encryption and decryption tests poison consumption | sandstone | uce-non-fatal |
| 10 | fisher --workload="sandstone -e \\*amx\\* -T 72h -v -o /dev/null" \  --injection-type=memory-uncorrectable-non-fatal \  --match='DRd\_^(?!mov)' --inspect-rate=200us \  --workload-exit-code=0,1 --runtime=20m $@ | 20 | Advanced Matrix Extensions Poison Intent: Poison crossed with different microbenchmarks using AMX instruction family | sandstone | uce-non-fatal |
| 11 | fisher --workload="sandstone -e \\*string\\* -T 72h -v -o /dev/null" \  --workload="sandstone -e \\*avx\\* -T 72h -v -o /dev/null" \  --injection-type=memory-uncorrectable-non-fatal \  --match='DRd\_^(?!mov$)' --inspect-rate=200us \  --workload-exit-code=0,1 --runtime=20m $@ | 20 | Strings + AVX + Poison Intent: Fast Strings crossed with AVX licensing and DCU memory poison consumption | sandstone | uce-non-fatal |
| 12 | fisher --workload="sandstone --5min --disable=\\*lock\\* -v -o /dev/null" \  --injection-type=memory-uncorrectable-non-fatal \  --match='DRd\_^(?!mov$)' --inspect-rate=200us \  --workload-exit-code=0,1 --runtime=20m $@ | 20 | Sandstone Breadth Poison Intent: Poison stress on sandstone micros with a random coverage-driven sweep of subtests | sandstone | uce-non-fatal |
| 13 | fisher --workload="sandstone -e \\*cmpxchg\\* -T 72h -v -o /dev/null" \  --injection-type=memory-uncorrectable-non-fatal \  --match='\*\_xchg' --inspect-rate=200us --no-split-cacheline \  --strict-inspect --workload-exit-code=0,1 --runtime=20m $@ | 20 | CAS + Poison Intent: Poison test focusing on atomic data exchanges of different lengths | sandstone | uce-non-fatal |
| 14 | fisher --workload="sandstone -e \\*rdrand\\* -T 2m -v -o /dev/null" \  --workload="sandstone --5min --disable=\\*lock\\* -v -o /dev/null" \  --injection-type=memory-uncorrectable-non-fatal \  --match='DRd\_^(?!mov)' --inspect-rate=200us \  --workload-exit-code=0,1 --runtime=20m $@ | 20 | DRNG Poison Intent: Memory tests with poison where subprograms are run with a mix of hardware entropy generation | sandstone | uce-non-fatal |
| 15 | fisher --workload="sandstone -e \\*string\\* -T 72h -v -o /dev/null" \  --injection-type=memory-uncorrectable-non-fatal \  --match='DRd\_^(?!mov$)' --inspect-rate=200us \  --workload-exit-code=0,1 --runtime=20m $@ | 20 | Strings + Poison Intent: Poison test for REP + {STOS, CMPS, SCAS} instructions, moving forward and backwards | sandstone | uce-non-fatal |
| 16 | fisher --workload="sandstone -e \\*hle\\* -T 72h -v -o /dev/null" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DWr\_lock --workload-exit-code=0,1 \  --inspect-rate=200us --runtime=20m $@ | 20 | Poison + TSX HLE Intent: Exercise the TSX load and store instructions with sandstone workload using HLE programming model while injecting in xaquire atomic writes an uncorrectable error | sandstone | uce-non-fatal |
| 17 | fisher --workload="sandstone -e \\*x87\\* -T 72h -v -o /dev/null" \  --injection-type=memory-uncorrectable-non-fatal \  --match=\\* --inspect-rate=200us \  --workload-exit-code=0,1 --runtime=20m $@ | 20 | x87 + Poison Intent: Poison test focusing on x87 math extensions | sandstone | uce-non-fatal |
| 18 | fisher --workload="stamp-rtm-genome -g 65536 -s 256 -n 67108864 -t \$((2\*\*(\$(shuf -i0-12 -n1))))" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DRd --match=DWr --inspect-rate=200us \  --runtime=20m $@ | 20 | Poison + TSX RTM Genome Sequencing Intent: Exercise the TSX load and store instructions while injecting an uncorrectable error in the critical sections | stamp-rtm | uce-non-fatal |
| 19 | fisher --workload="stamp-rtm-vacation -c \$((2\*\*(\$(shuf -i0-12 -n1))))" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DRd --match=DWr --inspect-rate=200us \  --runtime=20m $@ | 20 | Poison + TSX RTM Database Intent: Exercise the TSX load and store instructions while injecting an uncorrectable error in the critical sections | stamp-rtm | uce-non-fatal |
| 20 | fisher --workload="stream" \  --injection-type=memory-correctable:10,memory-uncorrectable-non-fatal \  --match='DRd\_^(?!mov$)' --runtime=20m $@ | 20 | Memory ECC Correctable + Patrol Scrub Uncorrectables + AVX512 (10:1 Ratio) Intent: Memory Benchmark using avx512 instructions with fast correctable injections and uncorrectable for patrol scrub detection | stream | uce-non-fatal |
| 21 | fisher --workload="stream" \  --injection-type=memory-correctable:10,memory-uncorrectable-non-fatal \  --match=PreF --runtime=20m $@ | 20 | Uncorrectable and Correctable Memory Mix - Explicit Prefetch Intent: Fast memory correctable injection with low probability of memory uncorrectable running memory benchmarks with AVX-512 instructions | stream | correctable:10, uce-non-fatal |
| 22 | fisher --workload="stress-ng --stream 0 --stream-madvise hugepage --verify --timeout 0" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DRd --workload-exit-code=0,2 \  --injection-einj-trigger --os-verification-rules=os-verification-trigger --strict-inspect \  --runtime=20m $@ | 20 | 2 MiB HugePages + Patrol Scrub Intent: Advising mmap from kernel to allocate pages larger than default to 2MiB, intent is to verify proper kernel actions when accessing poisoned data under this mmap scheme, leaving patrol scrub to detect the error | stress-ng | uce-non-fatal |
| 23 | fisher --workload="stress-ng --cpu 0 --cpu-method all --verify --timeout 0" \  --injection-type=memory-uncorrectable-non-fatal \  --match=CRd --workload-exit-code=0,2 --runtime=20m $@ | 20 | Code Poison Consumption MT Intent: IFU recovery flow stress on multi thread function calls and code reads | stress-ng | uce-non-fatal |
| 24 | fisher --workload="stress-ng --cpu \$(shuf -i1-$(nproc) -n1) --cpu-method callfunc --verify --timeout 0" \  --injection-type=memory-uncorrectable-non-fatal \  --match=CRd\_call --workload-exit-code=0,2 --runtime=20m $@ | 20 | Code Poison Recursion Intent: IFU recovery flow on recursive function calls | stress-ng | uce-non-fatal |
| 25 | fisher --workload="stress-ng --icache 0 --verify --timeout 0" \  --injection-type=memory-uncorrectable-non-fatal \  --match=CRd --workload-exit-code=0,2 --runtime=20m $@ | 20 | Code Poison SMC Intent: Self modifying code causing threads cache to reload poisoned code | stress-ng | uce-non-fatal |
| 26 | fisher --workload="stress-ng --cpu 1 --cpu-method all --verify --timeout 0" \  --injection-type=memory-uncorrectable-non-fatal \  --match=CRd --workload-exit-code=0,2 \  --timeout=800ms --runtime=20m $@ | 20 | Code Poison Consumption ST Intent: IFU recovery flow test on single thread function calls and code reads | stress-ng | uce-non-fatal |
| 27 | fisher --workload="stress-ng --mcontend \$(shuf -i1-$(nproc) -n1) --verify --timeout 0" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DRd --workload-exit-code=0,2 --runtime=20m $@ | 20 | Data Poison Memory Contention Intent: Memory ordering & contention stress with poison consumption | stress-ng | uce-non-fatal |
| 28 | fisher --workload="stress-ng --atomic \$(shuf -i1-$(nproc) -n1) --verify --timeout 0" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DWr --no-split-cacheline --strict-inspect \  --workload-exit-code=0,2 --runtime=20m $@ | 20 | Poison Atomics Cache Locks Intent: Atomic instruction stress with DCU memory poison consumption where all threads sweep | stress-ng | uce-non-fatal |
| 29 | fisher --workload="stress-ng --class cpu,cpu-cache,memory -x opcode,sigsegv,lockbus,atomic,misaligned,rmap --sequential \$(shuf -i1-$(nproc) -n1) --verify --timeout 4" \  --injection-type=memory-uncorrectable-non-fatal \  --match='DRd\_^(?!mov)' --workload-exit-code=0,2 --runtime=20m $@ | 20 | Mix: stress-ng Intent: Uncorrectable error injection while multiple stress scenarios ranging from cpu and memory stressors in parallel with variants such: unsigned integer math operations, libc string functions, floating point exceptions, instruction cache reloads, tlb shootdowns | stress-ng | uce-non-fatal |
| 30 | fisher --workload="stressapptest -W -C --cc\_test -s 7200 --max\_errors 1" \  --injection-type=memory-correctable \  --match=DRd --runtime=20m $@ | 20 | DDR5 ECC Correctable Stress Base Intent: Memory stressful traffic with memory correctable errors mixed with cache coherency stress and data verification from the workload | stressapptest | correctable |
| 31 | fisher --workload="stressapptest -M 2048 -H 2048 -p 131072 -W -s 7200 --max\_errors 1 --channel\_hash 0x20000" \  --injection-type=memory-uncorrectable-non-fatal \  --match=DWr --injection-delay-max=3s --runtime=20m $@ | 20 | Memory Partial Writes Intent: Simulates a heavy memory intensive workload with data verification crossed with a pool of memory trashers generating write merging stalls where poison is hit in the read underfill - high probability of partial writes but not guaranteed | stressapptest | uce-non-fatal |
| 32 | fisher --workload="y-cruncher pause:-1 skip-warnings colors:0 stress -M:1G -D:4" \  --match=DRd --injection-type=memory-uncorrectable-non-fatal \  --workload-exit-code=0,135 --inspect-rate=200us --runtime=20m $@ | 20 | Pi Calculation Intent: Number crunching stress over different number calculation algorithms running each for a short duration | y-cruncher | uce-non-fatal |

**Run Time**

Finish at least one run of all test cases (+10hrs)

**Setup Time**

4 hours

**Expected Results**

* Each test case should complete with > 90% PASS rate at the end of the test

[05:29:54.813] I: fisher: stats: PASS = 98 - FAIL = 4 ç 96% pass rate

[05:29:54.813] I: fisher: caught signal: SIGQUIT

[05:29:54.813] I: fisher: total run time: 0:05:00.813491

* Tests should not end with a fatal exit code.

[04:33:09.935] E: fisher: fatal: exit code = 134 ç means tool encountered unexpected error and could not complete the test

* Systems should not hang or reboot

### **Virtualization**

Virtualization is the way of running multiple operating systems on a computer system simultaneously. To the applications and users running on top of the virtualized machine, it appears as if they have their own dedicated machine, where the operating system, libraries, system resources and other programs are unique to the guest virtualized system. To sum up, virtualization allows users to use the server systems more efficiently. The content here is to simulate a real world virtualization scenario on SUTs.

**Prerequisites**

* Enable Intel VT-x and VT-d options in Setup menu

**System Requirement**

* Ensure that you have a Linux distribution that is with following package installed. Be sure to use package managers (e.g., APT, YUM, DNF) to install the package as well as install all dependent package
  + qemu-system-x86\_64

**Steps and Procedures**

**Building a base VM image**

1. Download a preferable base image from one of the Linux distros. Images for each distro can be referred here: <https://docs.openstack.org/image-guide/obtain-images.html>
2. Modify the downloaded image and make it capable of running FIO, Stress-ng, MLC with infinite loop
   1. Additionally, making all stressors in phase2 are capable of running in the base image
3. Modify the base image to let outside system can login with best known methods. For example, using a pre-created SSH key-pair, or using Cloud-init when creating VMs.
4. Save the modified image as a master image for creating the docker image. The base image will be a template for creating VMs.

**Run script for use in the container image in runtime**

1. Detect the system CPU HW thread number then minus 20. It is the available CPUs for VMs utilization($a).
2. Detect the system free memory space then minus 2048 MB, It is the available RAM for VMs utilization($b).
3. $b/$a is the size that RAM space for each VM.
4. Create VM based on the base image and system config from $a and $b. Let each VM run a different pre-installed workload. For example, VM1 runs FIO, VM2 runs Stress-ng, VM3 runs, MLC, VM4 runs FIO, etc.
5. Make the whole script can run to exceed the predefined run time.
6. Parsing logs from every VM and send it to Kibana for further analysis

Code example of launching VMs:

launch\_cmd="qemu-system-x86\_64 -machine pc,accel=kvm,kernel\_irqchip -cpu host -m $gRAM -smp 1 -vga none -nographic -device virtio-net-pci,netdev=net0 -netdev user,id=net0,hostfwd=tcp::$vm\_instance-:22 -drive file="$IMAGE",if=virtio,aio=threads -drive file="$CLOUDINIT",if=virtio,format=raw"

code example of logging output from VMs:

tmux new-session -d -s ${session} && tmux send-keys -t $session:0 "$launch\_cmd | sed "s/^/[vm-$vm\_instance] /" &>> /tests/vm\_logs/vm-$vm\_instance.log" C-m

**Build the container image**

1. Install the tool and its related dependency in the Linux container image by using package managers (e.g., APT, YUM, DNF)
2. Setup the environment in the image
3. Copy the run content (run.sh/run-specific.sh, the base VM image) to the container image
4. Write a Dockerfile based on step 1 to 3
5. Build the container image
6. Evaluate the docker image on a standalone system as well as on a cluster

**Run Time**

We restrict test time to 12 hours.

**Setup Time**

Estimated 6 hours

**Expected Results**

Can run the test in 12 hours with seeing any warning/error messages from SUTs.

**Virtualization – Kata**

Kata Containers is an open source project and community working to build a standard implementation of lightweight Virtual Machines (VMs) that feel and perform like containers, but provide the workload isolation and security advantages of VMs.

Kata Container allows us to run the test workload inside a lightweight VM to verify that content that works and identify issues on baremetal can also reproduce the same inside a VM. In FTx we use it to the stress the virtualization part of the system with TDX.

**Prerequisites**

* Linux kernel version 4.8 or higher.
* Kernel module vhost\_vsock is loaded.
* On the Eagle Stream / Sapphire Rapids BIOS, the following configuration is required TDX.
  + Disable Limit CPU PA to 46 bits

EDKII → Socket Configuration → Uncore Configuration → Uncore General Configuration → Limit CPU PA to 46 bits: <Disable>

* + Disable 2LM Memory Map (otherwise the Intel® TDX knob will be grayedout status)

EDKII → Socket Configuration → Memory Configuration → Memory Map → Volatile Memory Mode : <1LM>

* + Disable UMA Based Clustering (Sub-Numa Cluster [SNC] allowed but no hemisphere mode/Quad mode)
  + EDKII → Socket Configuration → Common RefCode Configuration → UMABased Clustering: <Disable (All2All)>
  + Enable Safer Mode Extensions (SMX), Virtual Machine Extension (VMX)

EDKII → Socket Configuration → Processor Configuration → Enable SMX : <Enable>

EDKII → Socket Configuration → Processor Configuration → Enable VMX : <Enable>

* + Enable Intel® SGX, Intel® TME, TME-MT, Intel® TDX

EDKII → Socket Configuration → Processor Configuration

* + - Total Memory Encryption (TME) → <Enable>
    - Total Memory Encryption Multi-Tenent (TME-MT) → <Enable>
    - Trust Domain Extension (TDX) → <Enable>
    - Software Guard Extension (SGX) → <Enable>
  + Adjust the Intel® TDX split key bits number

EDKII → Socket Configuration → Processor Configuration → Intel TME-MT Intel TDX key split <example:3> (non-zero value)

**System Requirement**

* Target CPU: ICX and SPR CPUs
* Operating System:
  + Debian 9
  + Ubuntu 18.04
  + CentOS 8
  + Fedora 34
  + SUSE Linux Enterprise (SLE) 15 SP2

**Steps and Procedures**

**Building Kata installation image**

1. **git clone** https://github.com/fidencio/kata-containers --branch topic/mvp-2023ww01-at-scale --single-branch
2. cd kata-containers && make kata-tarball *This will build all the needed packages including custom patches for the runtime and kata-deploy*
3. cp kata-static.tar.xz tools/packaging/kata-deploy/
4. docker build -t kata-deploy-bare:3.2.0-alpha0.2 -f tools/packaging/kata-deploy/Dockerfile tools/packaging/kata-deploy/
5. docker tag kata-deploy-bare:3.2.0-alpha0.1 prt-registry.sova.intel.com/kata-deploy-bare:3.2.0-alpha0.2
6. docker push prt-registry.sova.intel.com/kata-deploy-bare:3.2.0-alpha0.2 *This will pull kata prebuilt binaries from github, replace the shim with the patched version and add the TDX guest kernel to the package.*

**Run Kata installation container to setup Kata in SUT**

1. Label the nodes where you need to run the test with **localadmin.io-kata-node=true**
2. Kubectl apply -f kata-containers/tools/packaging/kata-deploy/kata-rbac/base/kata-rbac.yaml
3. Update kata-containers/tools/packaging/kata-deploy/kata-deploy/base/kata-deploy.yaml as below:
   1. Add nodeSelector:

|  |
| --- |
| spec:  nodeSelector:  localadmin.io-kata-node: "true" |

* 1. Modify image:

|  |
| --- |
| Image: <registry address>/kata-deploy-bare:2.3.2 |

1. Kubectl apply -f kata-containers/tools/packaging/kata-deploy/kata-deploy/base/kata-deploy.yaml
2. Remove “overHead” section in kata-containers/tools/packaging/kata-deploy/runtimeclasses/kata-runtimeClasses.yaml.
3. Kubectl apply -f kata-containers/tools/packaging/kata-deploy/runtimeclasses/kata-runtimeClasses.yaml

**Run Test Contents:**

Run standstone in separate pipeline for Kata container runtime

**Run Time**

We restrict test time to 12 hours.

**Setup Time**

Estimated 6 hours

**Expected Results**

Can run the test in 12 hours with seeing any warning/error messages from SUTs.

### **Software Guard Extensions (SGX)**

Intel® Software Guard Extensions (Intel® SGX) is an Intel technology for application developers seeking to protect select code and data from disclosure or modification. It allows user applications to run in an isolated enclave. This enclave protects user code and data from privileged mode attackers.

For this test, the purposes are:

* Verify SGX is enabled in target processor.
* Use sandstone as test content to validate that application can run well in enclave.

**Prerequisites**

* SGX-capable CPU (On servers, ICX and newer)
* Enable Intel® Software Guard Extensions (Intel® SGX) BIOS in Setup menu
  + Navigate to EDKII\*->Socket Configuration->Processor Configuration-> Intel® Total Memory Encryption (Intel® TME) = Enable
  + Intel® SGX Factory Reset = Disable
  + Intel® Guard Extensions (SGX) SW = Enable
  + Intel® SGX Package Info In-Band Access = Disable
  + Intel® SGX PRMRR Size = {Choose one}
  + Intel® SGX QoS = {Enable, Disable}
  + Select Owner EPOCH input type = New Random
  + SGXLEPUBKEYHASHx Write = Enable
  + Intel® SGX Debug Mode = Disable
  + Intel® SGX Auto MP Registration Agent = Enable
  + SGX registration server = {sbx, prx}
  + NUMA Mode = Enable (NUMA must be enabled)
  + UMA-Based clustering = Disable (Must be disabled)
  + SNC Mode = {Disable (1 PRMRR/Skt), SNC2 (2 PRMRR/Skt)}
  + Memory Mirroring = OFF (Must be off)
  + DDR ECC Enable = Enable (Must be enabled)
* SGX PSW and SGX SDK
* Intel SGX driver must be built in the Linux kernel
  + We recommend to use Linux kernel version 5.11 or higher: starting from this version, Linux has the FSGSBASE functionality as well as the Intel SGX driver built-in.
* Gramine: <https://gramine.readthedocs.io/en/latest/quickstart.html>
* Test contents: sandstone (>= v90, within SHC package v4.5.5. Please refer to SHC section regarding how to download SHC package from Intel RDC)

**System Requirement**

* Ensure that you have one of the following required operating systems:
  + Ubuntu\* 20.04 LTS Desktop 64bits
  + Ubuntu\* 20.04 LTS Server 64bits
  + Red Hat Enterprise Linux Server release 8.2 64bits
  + CentOS 8.2 64bits
* Required libraries. See
  + <https://github.com/intel/linux-sgx#build-the-intelr-sgx-sdk-and-intelr-sgx-psw-package>
  + <https://gramine.readthedocs.io/en/latest/devel/building.html>
* At least 1 DIMM per channel

**Steps and Procedures**

**Create Dockerfile based on below steps**

* Install SGX PSW
  + echo 'deb [arch=amd64] https://download.01.org/intel-sgx/sgx\_repo/ubuntu bionic main' | tee /etc/apt/sources.list.d/intel-sgx.list
  + wget <https://download.01.org/intel-sgx/sgx_repo/ubuntu/intel-sgx-deb.key>
  + apt-key add ./intel-sgx-deb.key
  + apt-get update && apt-get install -y libsgx-urts libsgx-epid
* Install SGX SDK
  + curl –OL <https://download.01.org/intel-sgx/latest/dcap-latest/linux/distro/ubuntu20.04-server/sgx_linux_x64_sdk_2.22.100.3.bin>chmod +x sgx\_linux\_x64\_sdk\_2.22.100.3.bin
  + echo -e "no\n/opt/intel" | ./sgx\_linux\_x64\_sdk\_2.22.100.3.bin
* Install Gramine: Follow steps in <https://gramine.readthedocs.io/en/latest/devel/building.html>
  + git clone <https://github.com/gramineproject/gramine.git>
  + cd gramine
  + openssl genrsa -3 -out enclave-key.pem 3072
  + python3 -m pip install 'meson>=0.55' 'toml>=0.10'
  + mkdir -p /usr/include/asm && wget -O /usr/include/asm/sgx.h <https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/plain/arch/x86/include/uapi/asm/sgx.h>
  + meson setup build/ --buildtype=release -Ddirect=enabled -Dsgx=enabled && ninja -C build/ && ninja -C build/ install
* Gramine preprocess:
  + openssl genrsa -3 -out enclave-key.pem 3072
  + Create a file named “[sandstone.manifest.template](https://gitlab.devtools.intel.com/cluster-central/cluster-content/-/blob/master/sgx/sgx-graphene/sandstone/sandstone.manifest.template)”.

Example of file contents:

|  |
| --- |
| # Python3 manifest example  loader.preload = "file:{{ gramine.libos }}"  libos.entrypoint = "{{ entrypoint }}"  loader.log\_level = "{{ log\_level }}"  loader.env.LD\_LIBRARY\_PATH = "/lib:/lib:{{ arch\_libdir }}:/usr/lib:/usr/{{ arch\_libdir }}"  loader.insecure\_\_use\_cmdline\_argv = true  sys.enable\_sigterm\_injection = true  fs.mount.lib.type = "chroot"  fs.mount.lib.path = "/lib"  fs.mount.lib.uri = "file:{{ gramine.runtimedir() }}"  fs.mount.lib2.type = "chroot"  fs.mount.lib2.path = "{{ arch\_libdir }}"  fs.mount.lib2.uri = "file:{{ arch\_libdir }}"  fs.mount.usr.type = "chroot"  fs.mount.usr.path = "/usr"  fs.mount.usr.uri = "file:/usr"  fs.mount.pyhome.type = "chroot"  fs.mount.pyhome.path = "{{ python.stdlib }}"  fs.mount.pyhome.uri = "file:{{ python.stdlib }}"  fs.mount.pydisthome.type = "chroot"  fs.mount.pydisthome.path = "{{ python.distlib }}"  fs.mount.pydisthome.uri = "file:{{ python.distlib }}"  fs.mount.tmp.type = "chroot"  fs.mount.tmp.path = "/tmp"  fs.mount.tmp.uri = "file:/tmp"  fs.mount.etc.type = "chroot"**[**  fs.mount.etc.path = "/etc"  fs.mount.etc.uri = "file:/etc"  sgx.debug = true  sgx.nonpie\_binary = true  sgx.enclave\_size = "64G"  #sys.stack.size = "2M"  sgx.thread\_num = 256  sgx.trusted\_files = [  "file:{{ entrypoint }}",  "file:{{ gramine.runtimedir() }}/",  "file:{{ arch\_libdir }}/",  "file:/usr/{{ arch\_libdir }}/",  "file:{{ python.stdlib }}/",  "file:{{ python.distlib }}/",  "file:/etc/mime.types",  "file:/usr/bin/gdb",  "file:/usr/share/gdb/",  ]  sgx.allowed\_files = [  "file:/etc/nsswitch.conf",  "file:/etc/ethers",  "file:/etc/hosts",  "file:/etc/group",  "file:/etc/passwd",  "file:/etc/gai.conf",  "file:/etc/host.conf",  "file:/etc/resolv.conf",  "file:/tmp",  "file://tests/scripts/"  ] |

* + gramine-manifest -Dlog\_level=error -Darch\_libdir=/lib/$(gcc -dumpmachine) -Dentrypoint=sandstone [sandstone.manifest.template](https://gitlab.devtools.intel.com/cluster-central/cluster-content/-/blob/master/sgx/sgx-graphene/sandstone/sandstone.manifest.template) > [sandstone.manifest](https://gitlab.devtools.intel.com/cluster-central/cluster-content/-/blob/master/sgx/sgx-graphene/sandstone/sandstone.manifest.template)
  + gramine-sgx-sign --key enclave-key.pem --manifest sandstone.manifest --output sandstone.manifest.sgx
  + gramine-sgx-get-token --output sandstone.token --sig sandstone.sig

**Run scripts:**

* Execute “gramine-sgx sandstone -vv --on-crash backtrace+core -T 12h --strict-runtime --output-log=/tmp/sandstone.log”

**Run Time**

We restrict test time to 12 hours.

**Setup Time**

Estimated 6 hours.

**Expected Results**

* Each test items of sandstone should pass (ok)
* Overall results: “exit: pass”

### **Intel Trust Domain Extensions [Intel TDX]**

Intel® Trust Domain Extensions (Intel® TDX) is Intel's newest confidential computing technology. This hardware-based trusted execution environment (TEE) facilitates the deployment of trust domains (TD), which are hardware-isolated virtual machines (VM) designed to protect sensitive data and applications from unauthorized access. Intel TDX uses hardware extensions for managing and encrypting memory and protects both the confidentiality and integrity of the TD CPU state from non-SEAM mode. Intel TDX ensures data integrity, confidentiality, and authenticity, which empowers engineers and tech professionals to create and maintain secure systems, enhancing trust in virtualized environments.

For this test, the purposes are:

* Verify TDX is enabled in target SUTs.
* Create VMs that run with TDX.

**Prerequisites**

To run TDX, please follow the documents from Intel RDC to enable TDX in SUTs

* Intel® Trust Domain Extension (Intel® TDX) Enabling and Validation
* Emerald Rapids Intel® Trust Domain Extension (Intel® TDX) Enablement and Validation Guide

Following setup must be met to enable TDX in SUTs

* Hardware: Jumper config
* TDX capable host Linux kernel
* TDX kernel parameters
* TDX BIOS knobs

**System Requirement**

DIMM Population should follow Intel® Software Guard Extensions (Intel® SGX) POR (1DPC or 2DPC)

**Steps and Procedures**

**Create Dockerfile based on below steps**

**Run scripts:**

**Run Time**

We restrict test time to 12 hours.

**Setup Time**

Estimated 6 hours.

**Expected Results**

* Each test items of sandstone should pass (ok)
* Overall results: “exit: pass”

### **Eigen**

Eigen is an open source, high-level C++ library of template headers for linear algebra, matrix and vector operations, geometrical transformations, numerical solvers and related algorithms.

In FTx we use Eigen to stress CPU computing capability in Combo1 test with other test contents in parallel.

.

**Prerequisites**

N/A

**System Requirement**

* Ensure that you have one of the following required operating systems:
  + Ubuntu\* 20.04 LTS Desktop 64bits

**Steps and Procedures**

**Create Dockerfile based on below steps:**

* + Install dependencies and tools:
    1. sudo apt update
    2. sudo apt install gcc g++ cmake make git
  + git clone --single-branch --depth=1 --branch "3.4.0" <https://gitlab.com/libeigen/eigen.git>
  + mkdir eigen/build && cd $\_
  + cmake .. -DCMAKE\_BUILD\_TYPE=Release

**Run script:**

Run below command in while loop:

make check -j `nproc`

**Run Time:** 86400 seconds

**Setup Time:** 1 hours

**Expected Results:** No failed test case.

### **NSS**

Network Security Services (NSS) is a set of libraries designed to support cross-platform development of security-enabled client and server applications. Applications built with NSS can support SSL v3, TLS, PKCS #5, PKCS #7, PKCS #11, PKCS #12, S/MIME, X.509 v3 certificates, and other security standards.

In FTx we use NSS to stress network security capability in Combo1, Combo2, and Combo3 tests with other test contents in parallel.

Network Security Services (NSS) is a set of libraries designed to support cross-platform development of security-enabled client and server applications. Applications built with NSS can support SSL v3, TLS, PKCS #5, PKCS #7, PKCS #11, PKCS #12, S/MIME, X.509 v3 certificates, and other security standards.

In FTx we use NSS to stress network security capability in Combo2 test with other test contents in parallel.

**Prerequisites**

N/A

**System Requirement**

* Ensure that you have one of the following required operating systems:
  + Ubuntu\* 20.04 LTS Desktop 64bits

**Steps and Procedures**

**Create Dockerfile based on below steps:**

1. Install dependencies and tools:
   * 1. sudo apt update
     2. sudo apt-get install -y gcc g++ iputils-ping libnspr4 libp11-3 libsqlite3-0 make nano python3 python-is-python3 zlib1g build-essential curl gyp libnspr4-dev libp11-dev libsqlite3-dev ninja-build zlib1g-dev
     3. python3 -m pip install gyp-next
2. Get source code:
   * 1. curl -L -O <https://ftp.mozilla.org/pub/security/nss/releases/NSS_3_77_RTM/src/nss-3.77.tar.gz>
     2. tar -xvf nss-3.77.tar.gz
3. cd nss-3.77/nss && ./build.sh -g -o -t x64 --gcc --system-nspr --system-sqlite --enable-libpkix
4. install -m755 -Dt /usr/local/bin /tests/nss-3.77/dist/Release/bin/\*
5. install -m644 -Dt /usr/local/lib /tests/nss-3.77/dist/Release/lib/\*

**Run script:**

In scripts:

hn="$(hostname)"

dn="local"

ip="$(hostname -i | awk '{ print $1 }')"

echo -e "\n# FTx-nss\n${ip} ${hn}.${dn}\n" >> /etc/hosts

Run below command in while loop:

BUILT\_OPT=1 HOST=$hn DOMSUF=$dn USE\_64=1 ./tests/all.sh

**Run Time:** 86400 seconds

**Setup Time:** 1 hours

**Expected Results:** No failed test case.

Tests summary:

--------------

Passed: 81976

Failed: 0

Failed with core: 0

ASan failures: 0

Unknown status: 0

### **Cloud-hypervisor**

Cloud Hypervisor is an open-source Virtual Machine Monitor (VMM) that runs on top of [KVM](https://www.kernel.org/doc/Documentation/virtual/kvm/api.txt) hypervisor and Microsoft Hypervisor (MSHV). It focuses on exclusively running modern, cloud workloads, on top of a limited set of hardware architectures and platforms. Cloud workloads refers to those that are usually run by customers inside a cloud provider. Here cloud workloads refer to modern operating systems with most I/O handled by paravirtualised devices (i.e. virtio), no requirement for legacy devices, and 64-bit CPUs.

**Prerequisites and System Requirement**

* Modern Linux distributions
* Linux package: git, build-essential, apt-utils,
* Rust toolchain

**Steps and Procedures**

**Build the container image (Dockerfile)**

1. Use Ubuntu 20.04 LTS as the base image for the container image and update its packages to latest version.   
   FROM ubuntu:20.04

RUN apt-get update

RUN apt-get -yq upgrade

1. Install latest Docker Engine in Ubuntu  
   RUN apt-get install -y apt-utils ca-certificates curl gnupg apt-transport-https lsb-release

RUN curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

RUN echo deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

RUN apt-get update

RUN DEBIAN\_FRONTEND=noninteractive TZ=Asia/UTC+8 apt-get -y install tzdata

RUN DEBIAN\_FRONTEND=noninteractive apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin

1. Install required packages for Cloud-hypervisor  
   RUN DEBIAN\_FRONTEND=noninteractive apt-get install -y flex build-essential bison musl-tools libelf-dev libseccomp-dev libcap-ng-dev git
2. Install Cloud-hypervisor  
   RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y

RUN echo 'source $HOME/.cargo/env' >> $HOME/.bashrc

ENV PATH="/root/.cargo/bin:${PATH}"

RUN rustup target add x86\_64-unknown-linux-musl

WORKDIR "/root"

RUN git clone https://github.com/cloud-hypervisor/cloud-hypervisor.git

WORKDIR "/root/cloud-hypervisor"

RUN cargo build --release

RUN setcap cap\_net\_admin+ep ./target/release/cloud-hypervisor

RUN cargo build --release --target=x86\_64-unknown-linux-musl --all

RUN service docker start

1. Setup up run\_specific.sh in container image. Please refer to the following section for the content of run\_specific.sh

ADD run\_specific.sh /root/cloud-hypervisor/run\_specific.sh

RUN chmod +x /root/cloud-hypervisor/run\_specific.sh

1. Setup entrypoint(run\_specific.sh) for container image  
   ENTRYPOINT ["/root/cloud-hypervisor/run\_specific.sh"]

**Run script:**

#!/bin/sh

trap 'exit' TERM

date

echo "test-start-indicator"

./scripts/run\_unit\_tests.sh

if [ $? -eq 0 ]; then

date

echo "TEST PASSED"

else

date

echo "TEST FAILED"

fi

echo "test-end-indicator"

touch /**tmp**/testdone

while true;

do

sleep 10

done

**Run Time:** 1 hours

**Setup Time:** 4 hours

**Expected Results: Complete pass all cloud-hypervisor test items without failed items.**

Running unittests (target/x86\_64-unknown-linux-gnu/debug/deps/vmm-3a68af0f7a9da866)

running 33 tests

test config::tests::test\_console\_parsing ... ok

test config::tests::test\_config\_validation ... ok

test config::tests::test\_device\_parsing ... ok

test config::tests::test\_cpu\_parsing ... ok

test config::tests::test\_mem\_parsing ... ok

test config::tests::test\_disk\_parsing ... ok

test config::tests::test\_net\_parsing ... ok

test config::tests::test\_option\_parser ... ok

test config::tests::test\_parse\_rng ... ok

test config::tests::test\_pmem\_parsing ... ok

test config::tests::test\_parse\_fs ... ok

test config::tests::test\_vdpa\_parsing ... ok

test config::tests::test\_vsock\_parsing ... ok

test cpu::tests::test\_setup\_fpu ... ok

test cpu::tests::test\_setlint ... ok

test cpu::tests::test\_setup\_msrs ... ok

test device\_tree::tests::test\_device\_tree ... ok

test unit\_tests::test\_vmm\_vm\_cold\_add\_device ... ok

test unit\_tests::test\_vmm\_vm\_cold\_add\_disk ... ok

test cpu::tests::test\_setup\_regs ... ok

test unit\_tests::test\_vmm\_vm\_cold\_add\_fs ... ok

test unit\_tests::test\_vmm\_vm\_cold\_add\_pmem ... ok

test unit\_tests::test\_vmm\_vm\_cold\_add\_net ... ok

test unit\_tests::test\_vmm\_vm\_cold\_add\_user\_device ... ok

test unit\_tests::test\_vmm\_vm\_cold\_add\_vdpa ... ok

test unit\_tests::test\_vmm\_vm\_cold\_add\_vsock ... ok

test unit\_tests::test\_vmm\_vm\_create ... ok

test vm::tests::test\_hob\_memory\_resources ... ok

test vm::test\_vm ... ok

test vm::tests::test\_vm\_created\_transitions ... ok

test vm::tests::test\_vm\_paused\_transitions ... ok

test vm::tests::test\_vm\_running\_transitions ... ok

test vm::tests::test\_vm\_shutdown\_transitions ... ok

test result: ok. 33 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.05s

### **Unixbench**

UnixBench is an open source benchmark suite that provides a basic indicators of the performance of a Unix-like system. The purpose of UnixBench is to provide a basic indicator of the performance of a Unix-like system; hence, multiple tests are used to test various aspects of the system's performance.

In FTx we use Unixbench to stress system in Combo2 test with other test contents in parallel.

UnixBench is an open source benchmark suite that provides a basic indicators of the performance of a Unix-like system. The purpose of UnixBench is to provide a basic indicator of the performance of a Unix-like system; hence, multiple tests are used to test various aspects of the system's performance.

In FTx we use Unixbench to stress system in Combo1 and Combo 2 tests with other test contents in parallel.

**Prerequisites**

N/A

**System Requirement**

* Ensure that you have one of the following required operating systems:
  + Ubuntu\* 20.04 LTS Desktop 64bits

**Steps and Procedures**

**Create Dockerfile based on below steps:**

1. Install dependencies and tools:
   * 1. sudo apt update
     2. sudo apt-get install libx11-dev libgl1-mesa-dev libxext-dev perl perl-modules make git
2. Get source code:
   * 1. git clone <https://github.com/kdlucas/byte-unixbench.git>
3. cd byte-unixbench/UnixBench/

**Run script:**

Run below command in while loop:

./Run

**Run Time:** 86400 seconds

**Setup Time:** 1 hours

**Expected Results:** Complete test successfully.

### **Firecracker**

Firecracker is a virtual machine monitor, VMM, that uses the Linux Kernel-based Virtual Machine (KVM) to create and manage microVMs. Firecracker has a minimalist design. It excludes unnecessary devices and guest functionality to reduce the memory footprint and attack surface area of each microVM. This improves security, decreases the startup time, and increases hardware utilization.

In FTx we use Firecracker to execute the Combo2 test.

**Prerequisites**

* Firecracker uses KVM and needs read/write access that can be granted as shown below:

|  |
| --- |
| $ sudo setfacl -m u:${USER}:rw /dev/kvm |

* The generic requirements are explained below:
  + Ubuntu\* 20.04 LTS Desktop 64bits

docker ( or podman).

* Meets the requirements to run Firecracker, you can leverage the tool

|  |
| --- |
| $ git clone https://github.com/firecracker-microvm/firecracker.git |

|  |
| --- |
| $ cd firecracker-microvm  $ sudo /tools/devtool checkenv |

|  |
| --- |
| [Firecracker devtool] Checking prerequisites for running Firecracker.  [Firecracker devtool] Please check ../docs/getting-started.md#prerequisites in case of any error.  [Firecracker devtool] Checking Host Security Configuration.  [Firecracker devtool] Please check ../docs/prod-host-setup.md in case of any error.  [Firecracker devtool] WARNING: KSM ENABLED  [Firecracker devtool] WARNING: retpoline, IBPB, IBRS: DISABLED.  [Firecracker devtool] WARNING: Mitigation: PTE Inversion: DISABLED  [Firecracker devtool] WARNING: VMX: cache flushes: DISABLED  [Firecracker devtool] WARNING: Hyperthreading ENABLED. |

**System Requirement**

* Linux

Firecracker currently supports physical Linux x86\_64 and aarch64 hosts. The currently supported kernel versions can be found [here](https://github.com/firecracker-microvm/firecracker/blob/main/docs/kernel-policy.md).

* KVM, please make sure that:

you have KVM enabled in your Linux kernel, and you have read/write access to /dev/kvm. If you need help setting up access to /dev/kvm, you should check out [Appendix A](https://github.com/firecracker-microvm/firecracker/blob/main/docs/getting-started.md#appendix-a-setting-up-kvm-access)

**Steps and Procedures**

**Create Dockerfile based on below steps**

**Run scripts:**

* In your Dockerfile
* Execute “gramine-sgx sandstone -vv --on-crash backtrace+core -T 12h --strict-runtime --output-log=/tmp/sandstone.log”

|  |
| --- |
| FIRECRACKER\_VER="v0.24.6" # must use particular version  git clone <https://github.com/firecracker-microvm/firecracker.git>  pushd firecracker  git checkout ${ FIRECRACKER\_VER} && git switch – release\_${ FIRECRACKER\_VER}  # If you only need to check functional cases, you can try the  # ./tools/devtool test -- --ignore integration\_tests/functional  ./tools/devtool test -- --ignore integration\_tests/performance |

* If you encounter KeyError: ‘ht\_enabled’, you probably need to modify the source code.

Errors would like this

|  |
| --- |
| integration\_tests/functional/test\_snapshot\_basic.py:82: in \_test\_seq\_snapshots  enable\_diff\_snapshots=enable\_diff\_snapshots)  framework/builder.py:120: in build  ht\_enabled=bool(microvm\_config['ht\_enabled']),  E KeyError: 'ht\_enabled' |

modify the source code as follows:

|  |
| --- |
| # /firecracker/tests/framework/builder.py, L120  ht\_enabled=bool(microvm\_config['ht\_enabled']) if ‘ht\_enabled’ in microvm\_config else True , |

Also, you can directly put the line in the Dockerfile.

|  |
| --- |
| # directly compose it in Dockerfile  sed -i.bak \  "s/ht\_enabled=bool(microvm\_config\['ht\_enabled']),/ht\_enabled=bool(microvm\_config\['ht\_enabled']) if 'ht\_enabled' in microvm\_config else True,/g" builder.py |

**Run Time** 24 hours

**Setup Time** 1 hours

**Expected Results**

The result may fail on certain circumstances due to performance of the peripherical devices such as input/output operations per second, IOPS, for example:

|  |
| --- |
| =================================FAILURES=================================  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_test\_patch\_drive\_limiter[ubuntu\_with\_ssh]\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  integration\_tests/functional/test\_drives.py:407: in test\_patch\_drive\_limiter  check\_iops\_limit(ssh\_connection, 4096, 1000, 0.85, 1.25)  integration\_tests/functional/test\_drives.py:363: in check\_iops\_limit  assert float(tokens[7]) < max\_time  E AssertionError: assert 1.34598 < 1.25  E + where 1.34598 = float('1.34598') |

The expected results would be as follows:

|  |
| --- |
| $ ./tools/devtool test -- --ignore integration\_tests/performance  [Firecracker devtool] 2022-04-25 07:00:26 UTC  [Firecracker devtool] Starting test run ...  ======================================================================================== test session starts ==============================================================================  platform linux -- Python 3.6.9, pytest-6.1.2, py-1.9.0, pluggy-0.13.1  cachedir: ../build/pytest\_cache  rootdir: /firecracker/tests, configfile: pytest.ini  plugins: timeout-1.4.2  timeout: 300.0s  timeout method: signal  timeout func\_only: False  collected 125 items  [ unsafe | 15 tests | 1 worker(s) ]  PASSED integration\_tests/build/test\_coverage.py::test\_coverage  PASSED integration\_tests/functional/test\_initrd.py::test\_microvm\_initrd\_with\_serial[minimal\_with\_initrd]  PASSED integration\_tests/functional/test\_max\_vcpus.py::test\_max\_vcpus[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_rate\_limiter.py::test\_tx\_rate\_limiting[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_rate\_limiter.py::test\_rx\_rate\_limiting[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_rate\_limiter.py::test\_rx\_rate\_limiting\_cpu\_load[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_signals.py::test\_generic\_signal\_handler[ubuntu-Signals.SIGBUS]  PASSED integration\_tests/functional/test\_signals.py::test\_generic\_signal\_handler[ubuntu-Signals.SIGSEGV]  PASSED integration\_tests/functional/test\_signals.py::test\_generic\_signal\_handler[ubuntu-Signals.SIGXFSZ]  PASSED integration\_tests/functional/test\_signals.py::test\_generic\_signal\_handler[ubuntu-Signals.SIGXCPU]  PASSED integration\_tests/functional/test\_signals.py::test\_generic\_signal\_handler[ubuntu-Signals.SIGPIPE]  PASSED integration\_tests/functional/test\_signals.py::test\_generic\_signal\_handler[ubuntu-Signals.SIGHUP]  PASSED integration\_tests/functional/test\_signals.py::test\_generic\_signal\_handler[ubuntu-Signals.SIGILL]  PASSED integration\_tests/functional/test\_signals.py::test\_sigxfsz\_handler[ubuntu]  PASSED integration\_tests/functional/test\_signals.py::test\_handled\_signals[ubuntu\_with\_ssh]  [ safe | 110 tests | 1 worker(s) ]  PASSED integration\_tests/build/test\_binary\_size.py::test\_binary\_sizes  PASSED integration\_tests/build/test\_clippy.py::test\_rust\_clippy[x86\_64-unknown-linux-gnu]  PASSED integration\_tests/build/test\_clippy.py::test\_rust\_clippy[x86\_64-unknown-linux-musl]  PASSED integration\_tests/build/test\_unittests.py::test\_unittests[x86\_64-unknown-linux-gnu]  SKIPPED integration\_tests/build/test\_unittests.py::test\_unittests[x86\_64-unknown-linux-musl]  PASSED integration\_tests/functional/test\_api.py::test\_api\_happy\_start[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_api\_put\_update\_pre\_boot[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_net\_api\_put\_update\_pre\_boot[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_api\_put\_machine\_config[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_api\_put\_update\_post\_boot[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_rate\_limiters\_api\_config[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_api\_patch\_pre\_boot[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_api\_patch\_post\_boot[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_drive\_patch[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_send\_ctrl\_alt\_del[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_api\_vsock[ubuntu]  PASSED integration\_tests/functional/test\_api.py::test\_api\_balloon[debian\_with\_ssh\_and\_balloon]  PASSED integration\_tests/functional/test\_balloon.py::test\_rss\_memory\_lower[debian\_with\_ssh\_and\_balloon]  PASSED integration\_tests/functional/test\_balloon.py::test\_inflate\_reduces\_free[debian\_with\_ssh\_and\_balloon]  PASSED integration\_tests/functional/test\_balloon.py::test\_deflate\_on\_oom\_true[debian\_with\_ssh\_and\_balloon]  PASSED integration\_tests/functional/test\_balloon.py::test\_deflate\_on\_oom\_false[debian\_with\_ssh\_and\_balloon]  PASSED integration\_tests/functional/test\_balloon.py::test\_reinflate\_balloon[debian\_with\_ssh\_and\_balloon]  PASSED integration\_tests/functional/test\_balloon.py::test\_size\_reduction[debian\_with\_ssh\_and\_balloon]  PASSED integration\_tests/functional/test\_balloon.py::test\_stats[debian\_with\_ssh\_and\_balloon]  PASSED integration\_tests/functional/test\_balloon.py::test\_balloon\_snapshot  PASSED integration\_tests/functional/test\_balloon.py::test\_snapshot\_compatibility  PASSED integration\_tests/functional/test\_balloon.py::test\_memory\_scrub  PASSED integration\_tests/functional/test\_cmd\_line\_start.py::test\_config\_start\_with\_api[ubuntu\_with\_ssh-framework/vm\_config.json]  PASSED integration\_tests/functional/test\_cmd\_line\_start.py::test\_config\_start\_no\_api[ubuntu\_with\_ssh-framework/vm\_config.json]  PASSED integration\_tests/functional/test\_concurrency.py::test\_run\_concurrency[ubuntu\_with\_ssh, 20 instance(s)]  PASSED integration\_tests/functional/test\_cpu\_features.py::test\_cpuid[ubuntu\_with\_ssh-True-1]  PASSED integration\_tests/functional/test\_cpu\_features.py::test\_cpuid[ubuntu\_with\_ssh-True-2]  PASSED integration\_tests/functional/test\_cpu\_features.py::test\_cpuid[ubuntu\_with\_ssh-True-16]  PASSED integration\_tests/functional/test\_cpu\_features.py::test\_cpuid[ubuntu\_with\_ssh-False-1]  PASSED integration\_tests/functional/test\_cpu\_features.py::test\_cpuid[ubuntu\_with\_ssh-False-2]  PASSED integration\_tests/functional/test\_cpu\_features.py::test\_cpuid[ubuntu\_with\_ssh-False-16]  SKIPPED integration\_tests/functional/test\_cpu\_features.py::test\_cpu\_features[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_cpu\_features.py::test\_brand\_string[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_cpu\_features.py::test\_cpu\_template[ubuntu\_with\_ssh-T2]  PASSED integration\_tests/functional/test\_cpu\_features.py::test\_cpu\_template[ubuntu\_with\_ssh-C3]  PASSED integration\_tests/functional/test\_drives.py::test\_rescan\_file[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_drives.py::test\_device\_ordering[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_drives.py::test\_rescan\_dev[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_drives.py::test\_non\_partuuid\_boot[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_drives.py::test\_partuuid\_boot[ubuntu\_with\_partuuid]  PASSED integration\_tests/functional/test\_drives.py::test\_partuuid\_update[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_drives.py::test\_patch\_drive[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_drives.py::test\_patch\_drive\_limiter[ubuntu\_with\_ssh]  SKIPPED integration\_tests/functional/test\_error\_code.py::test\_enosys\_error\_code[minimal\_with\_initrd]  PASSED integration\_tests/functional/test\_logging.py::test\_no\_origin\_logs[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_logging.py::test\_no\_level\_logs[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_logging.py::test\_no\_nada\_logs[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_logging.py::test\_info\_logs[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_logging.py::test\_warn\_logs[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_logging.py::test\_error\_logs[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_logging.py::test\_log\_config\_failure[ubuntu]  PASSED integration\_tests/functional/test\_logging.py::test\_api\_requests\_logs[ubuntu]  PASSED integration\_tests/functional/test\_max\_devices.py::test\_attach\_maximum\_devices[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_max\_devices.py::test\_attach\_too\_many\_devices[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_metrics.py::test\_flush\_metrics[ubuntu]  PASSED integration\_tests/functional/test\_mmds.py::test\_custom\_ipv4[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_mmds.py::test\_json\_response[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_mmds.py::test\_imds\_response[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_mmds.py::test\_larger\_than\_mss\_payloads[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_mmds.py::test\_mmds\_dummy[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_mmds.py::test\_guest\_mmds\_hang[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_net.py::test\_high\_ingress\_traffic[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_net\_config\_space.py::test\_net\_change\_mac\_address[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_pause\_resume.py::test\_pause\_resume  SKIPPED integration\_tests/functional/test\_rtc.py::test\_rtc[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_serial\_io.py::test\_serial\_console\_login[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_serial\_io.py::test\_serial\_dos[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_shut\_down.py::test\_reboot[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_snapshot\_advanced.py::test\_restore\_old\_snapshot\_all\_devices  PASSED integration\_tests/functional/test\_snapshot\_advanced.py::test\_restore\_old\_version\_all\_devices  PASSED integration\_tests/functional/test\_snapshot\_basic.py::test\_patch\_drive\_snapshot  PASSED integration\_tests/functional/test\_snapshot\_basic.py::test\_5\_full\_snapshots  PASSED integration\_tests/functional/test\_snapshot\_basic.py::test\_5\_inc\_snapshots  PASSED integration\_tests/functional/test\_snapshot\_basic.py::test\_load\_snapshot\_failure\_handling[ubuntu]  PASSED integration\_tests/functional/test\_snapshot\_basic.py::test\_cmp\_full\_and\_first\_diff\_mem  PASSED integration\_tests/functional/test\_snapshot\_version.py::test\_create\_with\_past\_version[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_snapshot\_version.py::test\_create\_with\_too\_many\_devices[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_topology.py::test\_cpu\_topology[ubuntu\_with\_ssh-True-1]  PASSED integration\_tests/functional/test\_topology.py::test\_cpu\_topology[ubuntu\_with\_ssh-True-2]  PASSED integration\_tests/functional/test\_topology.py::test\_cpu\_topology[ubuntu\_with\_ssh-True-16]  PASSED integration\_tests/functional/test\_topology.py::test\_cpu\_topology[ubuntu\_with\_ssh-False-1]  PASSED integration\_tests/functional/test\_topology.py::test\_cpu\_topology[ubuntu\_with\_ssh-False-2]  PASSED integration\_tests/functional/test\_topology.py::test\_cpu\_topology[ubuntu\_with\_ssh-False-16]  PASSED integration\_tests/functional/test\_topology.py::test\_cache\_topology[ubuntu\_with\_ssh-True-1]  PASSED integration\_tests/functional/test\_topology.py::test\_cache\_topology[ubuntu\_with\_ssh-True-2]  PASSED integration\_tests/functional/test\_topology.py::test\_cache\_topology[ubuntu\_with\_ssh-True-16]  PASSED integration\_tests/functional/test\_topology.py::test\_cache\_topology[ubuntu\_with\_ssh-False-1]  PASSED integration\_tests/functional/test\_topology.py::test\_cache\_topology[ubuntu\_with\_ssh-False-2]  PASSED integration\_tests/functional/test\_topology.py::test\_cache\_topology[ubuntu\_with\_ssh-False-16]  PASSED integration\_tests/functional/test\_vsock.py::test\_vsock[ubuntu\_with\_ssh]  PASSED integration\_tests/functional/test\_vsock.py::test\_vsock\_epipe[ubuntu\_with\_ssh]  PASSED integration\_tests/security/test\_jail.py::test\_default\_chroot[ubuntu\_with\_ssh]  PASSED integration\_tests/security/test\_jail.py::test\_empty\_jailer\_id[ubuntu\_with\_ssh]  PASSED integration\_tests/security/test\_jail.py::test\_default\_chroot\_hierarchy[minimal\_with\_initrd]  PASSED integration\_tests/security/test\_jail.py::test\_arbitrary\_usocket\_location[minimal\_with\_initrd]  PASSED integration\_tests/security/test\_jail.py::test\_cgroups[minimal\_with\_initrd]  PASSED integration\_tests/security/test\_jail.py::test\_node\_cgroups[minimal\_with\_initrd]  PASSED integration\_tests/security/test\_jail.py::test\_args\_cgroups[minimal\_with\_initrd]  PASSED integration\_tests/security/test\_sec\_audit.py::test\_cargo\_audit  PASSED integration\_tests/security/test\_seccomp.py::test\_seccomp\_ls  PASSED integration\_tests/security/test\_seccomp.py::test\_advanced\_seccomp\_harmless  PASSED integration\_tests/security/test\_seccomp.py::test\_advanced\_seccomp\_malicious  PASSED integration\_tests/security/test\_seccomp.py::test\_advanced\_seccomp\_panic  PASSED integration\_tests/security/test\_seccomp.py::test\_seccomp\_applies\_to\_all\_threads[ubuntu]  PASSED integration\_tests/security/test\_ssbd\_mitigation.py::test\_ssbd\_mitigation[minimal\_with\_initrd]  ======================121 passed, 4 skipped in 2181.57s (0:36:21) ======================== |

The testing includes various architectures. The skipped tests are for *aarch64*. In x86, we will omit them.

### **Povray**

Povray (Persistence of View ray tracer) is an open-source ray tracing program that generates 3D images based on a text-based scene description language. It is used in the FTx program as a workload that has heavy parallel cacheline data sharing characteristics.

**Prerequisites**

* NA

**System Requirement**

* Ubuntu\* 20.04 LTS Desktop 64bits recommended

**Steps and Procedures**

**Create Dockerfile based on below steps**

* Povray can be installed directly through default Ubuntu repository
  + apt install povray
* (It is also possible to compile the tool through the source code located at <https://github.com/POV-Ray/povray/tree/3.7-stable>)

**Run scripts:**

Create script with while loop of following command

povray --benchmark <<< EOF

**Run Time:** 6hrs individually or match up to test time of Combo tests

**Setup Time:** 30 min

**Expected Results:** Complete test successfully.

### VSS

VSS.base

|  |
| --- |
| env.JENKINS\_TEST\_TAG="VSS-3.6.30c"  env.JENKINS\_TEST\_LABEL="vss-unified"  env.JENKINS\_NODE\_LABEL="unifiednode"  env.JENKINS\_TEST\_ARGS="S145"  env.JOB\_RUNTIME=190000  env.ITERATIONS=1 |

Dockerfile

|  |
| --- |
| FROM ubuntu:latest  COPY ilvss-3.6.\*.bin /root  RUN apt update  RUN apt -y upgrade  WORKDIR /root  RUN VSS=I ./ilvss-3.6.\*.bin  COPY 2024\_0801\_ilVSS\_Quanta.key /opt/ilvss  RUN mkdir -p /tests/scripts  COPY run-specific.sh /tests/scripts  COPY dbg.sh /tests/scripts  RUN chmod +x /tests/scripts/\*.sh  ENTRYPOINT ["/tests/scripts/run-specific.sh"] |

Run-specific.sh

|  |
| --- |
| #!/bin/sh  pid=0  # Shutdown child process and exit  remove\_child\_process() {  echo "cleaning up child process"  kill -9 "$pid" > /dev/null 2>&1  exit  }  # Print expected text markers and attempt to stop child process  terminate\_run() {  if [ $pid -eq 0 ]; then  exit  fi  date  echo "TEST ABORTED"  echo "test-end-indicator"  remove\_child\_process  exit  }  # Gracefully handle the TERM signal sent when a POD or container  # deleted/shutdown/stopped.  trap 'terminate\_run' TERM  # Gracefully handle SIGINT/CTRL-C, SIGQUIT/CTRL-D,  # kill -p PID signals  trap 'remove\_child\_process' INT QUIT KILL  # do the work  # This can hang, crash or return with error code  # Hangs are detected via timeout of the readiness probe  # Crashes and errors via exit code and telemetry  date  echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  echo " VSS - 3.6.30c"  echo "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"  echo "test-start-indicator"  #$SANDSTONE\_BIN $SANDSTONE\_ARGS &  echo "Pause for 30 seconds before starting the tests"  sleep 30  case "${SANDSTONE\_ARGS}" in  "S145")  echo "Running 48 hour VSS memory and i/o stress test"  FLOW="S145"  PKX="stress\_egs.pkx"  PC="EGS"  ;;  "S2")  echo "Running VSS storage stress test"  FLOW="S2"  PKX="stress\_egs.pkx"  PC="EGS"  ;;  "S3")  echo "Running VSS i/o test"  FLOW="S3"  PKX="stress\_egs.pkx"  PC="EGS"  ;;  "mem")  echo "Running VSS memory stress test"  FLOW="mem"  PKX="stress\_egs.pkx"  PC="EGS"  ;;  esac  sleep 3  dmesg -C  export PATH=$PATH:.:/opt/ilvss  # set ulimit  ulimit -n 1024  ulimit -H -n 262144  cd /opt/ilvss; /opt/ilvss/t /pkg /opt/ilvss/packages/${PKX} /pc ${PC} /flow ${FLOW} /reconfig /re err.log /run  /tests/scripts/dbg.sh  sleep 3  pid="$!"  wait ${pid}  if [ $? -eq 0 ]; then  # Test did not detect an error  date  echo "TEST PASSED"  else  # Something went wrong  date  echo "TEST FAILED"  fi  echo "test-end-indicator"  pid=0  touch /tmp/testdone  while true; do sleep 10; done |

Dbg.sh

|  |
| --- |
| #!/usr/bin/bash  BIOS=`dmidecode -t BIOS | grep Version:`  NUMA\_EN=`ls /sys/devices/system/node/ | grep node1`  OS=`cat /etc/os-release | grep -i PRETTY\_NAME | cut -d"=" -f 2`  HNAME=`cat /etc/hostname`  NUM\_NODES=`ls /sys/devices/system/node/ | grep node | wc -l`  echo " LINUX OS: $OS"  echo " HOSTNAME: $HNAME"  echo " BIOS VERSION: $BIOS"  if [ $NUMA\_EN = "node1" ]  then  echo "NUMA Enabled! Found $NUM\_NODES nodes."  cat /sys/devices/system/node/node\*/meminfo | grep MemTotal:  else  echo "NUMA NOT Enabled!"  cat /proc/meminfo | grep MemTotal:  fi  echo  dmidecode -t PROCESSOR | grep -A 5 "Processor Information"  echo  echo "DIMM Configuration:"  NODIMM=`dmidecode -t memory | grep Size: | grep "No Module" | wc -l`  DIMMSLOT=`dmidecode -t memory | awk '$1 == "Size:" {print $1}' | wc -l`  echo " DIMM Slots Total = $DIMMSLOT"  echo " DIMM Slots Empty = $NODIMM"  POP=$(($DIMMSLOT-$NODIMM))  echo " DIMM Slots Populated = $POP"  echo -n "Searching for errors in system log - "  LXMEM=`dmesg -T | grep lxmem`  OOM=`dmesg -T | grep oom-kil`  CTC=`dmesg -T | grep ctc`  SEGFLT=`dmesg -T | grep segfault`  MCE=`dmesg -T | grep -i mce`  CNT=0  if [ ${#CTC} -gt 0 ] ; then  echo;echo "CTC ERROR:"  echo "${CTC}"  CNT=$((CNT+=1))  fi  if [ ${#LXMEM} -gt 0 ] ; then  echo;echo "LXMEM (memory stress test) ERROR:"  echo "${LXMEM}"  CNT=$((CNT+=1))  fi  if [ ${#OOM} -gt 0 ] ; then  echo;echo "oom-killer envoked (out of memory) ERROR:"  echo "${OOM}"  CNT=$((CNT+=1))  fi  if [ ${#SEGFLT} -gt 0 ] ; then  echo;echo "Segmentation Fault ERROR:"  echo "${SEGFLT}"  CNT=$((CNT+=1))  fi  if [ ${#MCE} -gt 0 ] ; then  echo;echo "MCE ERROR:"  echo "${SEGFLT}"  CNT=$((CNT+=1))  fi  if [ $CNT -eq 0 ] ; then  sleep 3  echo "No ERRORS found in system log!"  fi  echo;echo "===============VSS error log BEGIN==============="  if [ -f /opt/ilvss/err.log ] ; then  cat /opt/ilvss/err.log  else  echo "NO err.log found!"  fi  echo "===============VSS error log END==============="  echo;echo "===============VSS Summary of test results on ${HNAME}==============="  if [ -f /opt/ilvss/SUMMARY\_T.TXT ]; then  cat /opt/ilvss/SUMMARY\_T.TXT  else  echo "No SUMMARY\_T.TXT file!"  fi  echo;echo "===============VSS DEBUG INFO END===============" |

Container Build Process

1. Put the following files in a single folder.  
   2024\_0801\_ilVSS\_Quanta.key, Dockerfile, dbg.sh, ilvss-3.6.30c.bin, run-specific.sh
2. docker build . -t myimage:VSS.V3.6.30c
3. rename and upload the container to specific container repository