Securing AI workloads on Dell PowerEdge with Intel® Xeon® processors using Intel® Trust Domain Extensions

May 2024

H20040

White Paper

Abstract

This technical white paper offers guidance on implementing strong security measures and achieving peak performance for AI workloads. It focuses on two features provided by the 5th generation Intel® Xeon® processor: implementing advanced security measures using Intel® TDX and leveraging an AI accelerator called Intel® AMX. By utilizing these features, it is possible to achieve Confidential AI while minimizing any negative impact to achieve optimal performance.

DELL Technologies

Copyright

Contents

Executive summary....4
Solution overview....6
Solution design....9
Results or findings....11
Conclusion....13
References....15
Appendix A - Enable Intel TDX on Dell server....16
Appendix B – Enable Intel TDX in BIOS....18
Appendix C – Benchmark setup....19
Appendix D – Definitions for Virtual Machines....21

Executive summary

Overview

Getting the most out of server performance is crucial in today's AI-driven landscape. As organizations increasingly embrace AI for innovation, decision-making, and operational efficiency, they encounter challenges related to compute resource consumption. AI workloads, particularly those involving complex deep learning models, demand substantial compute resources due to the computational intensity of AI algorithms.

These workloads often involve running business-critical AI models and processing sensitive data, making it essential to prioritize data protection and application security. To address security concerns, the concept of Confidential Compute has emerged, focusing on protecting data and applications during processing. By implementing Confidential Compute for AI workloads, organizations can ensure their sensitive data remains secure, and their critical AI models are protected from unauthorized access or tampering. This approach helps maintain the confidentiality and integrity of both the data and the applications, enabling organizations to leverage the power of AI while maintaining a strong security posture.

One common concern for implementing security is the potential negative impact on performance. IT administrators are often caught between the desire to extract maximum system performance and the need to safeguard AI workloads. Security measures such as encryption and access controls can introduce additional computational overhead, which may slow down system performance and affect the overall user experience. This is particularly relevant for AI workloads, which are known to consume significant resources. Another factor contributing to this reluctance is the perception that security measures can be complex and time-consuming to implement and manage. IT administrators may worry about the additional workload and potential disruptions that come with implementing and maintaining security measures. Additionally, there might be a fear of compatibility issues or conflicts with existing systems and applications. IT administrators may hesitate to implement security measures if they believe it could lead to compatibility issues with critical software or hardware components.

This white paper addresses these challenges and offers guidance on optimizing performance, securing your AI workloads and ensuring ease of implementation using Dell PowerEdge 16th generation with 5th Gen Intel® Xeon® processors. The solution outlined in this paper showcases the ease of implementation to achieve Confidential AI while minimizing any negative impact on performance by focusing on two features provided by the 5th generation Intel® Xeon® processor: leveraging an AI accelerator called Intel® AMX and implementing advanced security measures using Intel® TDX. The purpose of this document is to provide guidance on achieving peak performance for AI workloads and implementing strong security measures.

Audience

This document is intended for solution architects, data scientists, IT decision makers, and security specialists.

Revisions

Date	Part number/ revision	Description
May 2024	H20040	Initial release

We value your feedback

Dell Technologies and the authors of this document welcome your feedback on this document. Contact the Dell Technologies team by email.

Author: Dell Server TME team

Contributors: Lokendra Uppuluri (Intel), Deepak Rangaraj (Dell), Ted Streete (Dell, Shyam Iyer (Dell), Syama Kim Kinahan (Dell)

Note: For links to other documentation for this topic, see https://infohub.delltechnologies.com/en-us/t/poweredge-cyber-security/

Solution overview

Business challenges

Following are some specific business challenges that Dell PowerEdge 5th Generation of Intel® Xeon® Processors leveraging Intel® Trust Domain Extensions (TDX) and Intel® Advanced Matrix Extensions (AMX) can address.

As technology advances and more industries recognize the potential of artificial intelligence, there has been a significant rise in the adoption of AI across various sectors. AI workloads are being used in fields such as healthcare, finance, retail, manufacturing, and so on. This growth can be attributed to increasing availability of data, advancements in machine learning algorithms, and the need for automation and intelligent decision-making. As a result, there is a growing demand for AI solutions, leading to an upward trend in the number of AI workloads being deployed.

Deploying an AI workload can be a costly endeavor. By utilizing Intel® Xeon® processors with AMX acceleration, you can effectively run your AI workloads without the need for specialized hardware. Dell PowerEdge 5th Generation of Intel® Xeon® Processors offers enhanced versatility and simplicity, allowing organizations to run any workload—including AI—on Intel® Xeon® processors. By leveraging this optimization, organizations can maximize their AI deployment efficiency while minimizing expenses.

Because these workloads often involve running business-critical AI models and processing sensitive data, it is essential to prioritize data protection and application security. Code integrity and intellectual property (IP) protection are crucial aspects in ensuring the security and confidentiality of AI models and software. AI models, being valuable IP assets, must be safeguarded against unauthorized access and theft. This involves implementing measures to encrypt and protect the code, as well as controlling access to the models.

Equally important is safeguarding sensitive data, which includes implementing robust security measures to safeguard data while it is being stored, transmitted, and actively processed by the CPU. Additionally, compliance with data privacy regulations such as GDPR or HIPAA is essential to maintaining legal and ethical standards for data protection.

Another challenge is protecting data and workload integrity against difficult-to-defend and Zero Day threats, which is crucial to maintaining a secure computing environment. While IT departments have tools and procedures in place to restrict privileges and define access permissions, there are still potential vulnerabilities that can be exploited. For instance, a lower-privilege user or software may exploit a vulnerability to escalate privileges, a privileged application could be compromised, a privileged user might turn malicious, or a previously unknown vulnerability could be exploited.

In such scenarios, Dell PowerEdge 5th Generation Intel® Xeon® processors offer confidential computing with Intel® TDX. By leveraging Intel® TDX on PowerEdge, organizations can create secure enclaves where sensitive data and critical workloads are processed in a protected environment, shielding them from potential attacks and ensuring data and workload integrity in the cloud. One of the key benefits of Intel® TDX is that it can protect the system even if the host operating system or hypervisor is compromised because they are outside of the trust boundary. This is because the trust domains are created and managed by the CPU itself rather than relying on software-based security measures, which can also help protect against advanced threats such as rootkits,

bootkits, and kernel-level attacks. By embracing confidential computing with Intel® TDX on Dell PowerEdge, organizations can effectively protect their sensitive information and intellectual property, enabling them to leverage the power of AI without compromising security.

Lastly, Intel® TDX requires no code changes to the application. This approach streamlines the implementation of security measures and reduces the time and effort required to ensure application protection. By improving efficiency and enabling new business models, organizations can drive innovation and enhance productivity.

What is Intel® TDX

Intel® Trust Domain Extensions (TDX) is the newest confidential computing technology from Intel®. 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.

A CPU-measured Intel® TDX module enables Intel® TDX. This software module runs in a new CPU Secure Arbitration Mode (SEAM) as a peer virtual machine manager (VMM) and supports TD entry and exit using the existing virtualization infrastructure. The module is hosted in a reserved memory space identified by the SEAM Range Register (SEAMRR).

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 uses architectural elements such as SEAM, a shared bit in Guest Physical Address (GPA), the secure Extended Page Table (EPT), the physical-address-metadata table, Intel® Total Memory Encryption – Multi-Key (Intel® TME-MK), and remote attestation.

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.

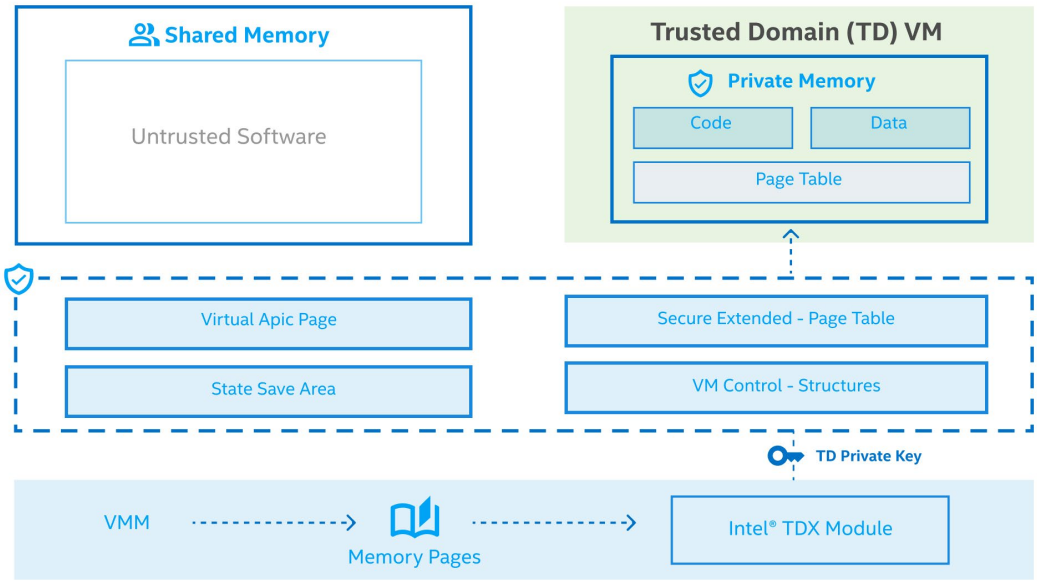

Figure 1. Architecture of Intel® TDX

What is Intel® AMX

Dell PowerEdge 16th generation with 5th Gen Intel® Xeon® processors support AMX and can handle the computational demands of Generative AI workloads more efficiently, allowing users to run these workloads without the need for additional specialized hardware and making it more accessible and cost-effective to utilize Generative AI capabilities.

To help increase the efficiency and cost-effectiveness of your deep learning workloads and make them easier to train and deploy, Intel® AMX on Intel® Xeon® Scalable processors delivers acceleration for inferencing and training while minimizing the need for specialized hardware. Intel® AMX is a dedicated hardware block found on the Intel® Xeon® Scalable processor core that helps optimize and accelerate deep learning training and inferencing workloads that rely on matrix math.

Intel® AMX can help with running Generative AI workloads without the need for specialized hardware by providing accelerated matrix operations. Generative AI involves complex computations, particularly in tasks like image generation, natural language processing, and deep learning. These workloads heavily rely on matrix operations, which can be computationally intensive. Intel® AMX introduces a new set of x86 instructions specifically designed to accelerate matrix operations. By leveraging Intel® AMX, software developers can optimize their Generative AI algorithms to take advantage of these instructions, resulting in improved performance and faster execution of matrix operations without necessitating specialized hardware.

Intel® AMX enables AI workloads to run on the CPU instead of offloading them to a discrete accelerator, providing a significant performance boost.[2] Its architecture supports BF16 (training/inference) and int8 (inference) data types and includes two main components:

- **Tiles**: These consist of eight two-dimensional registers, each 1 kilobyte in size, that store large chunks of data.
- **Tile Matrix Multiplication (TMUL)**: TMUL is an accelerator engine attached to the tiles that performs matrix-multiply computations for AI.

Together, these components enable Intel® AMX to store more data in each core and compute larger matrices in a single operation. Additionally, Intel® AMX is architected to be fully extensible and scalable.

Dell PowerEdge 16th generation with 5th Gen Intel® Xeon® processors and Intel® AMX acceleration offers a cost-effective and efficient solution for AI deployment optimization. Organizations can reduce costs to effectively run AI workloads without the need for specialized hardware while also gaining enhanced versatility and simplicity, allowing organizations to run any workload, including AI. With this optimization, organizations can maximize their AI deployment efficiency while minimizing expenses.

Solution approach

The solution outlined in this paper leverages Intel® TDX technology to protect generative AI inference workloads using a large language model to showcase that it is quick, easy to use, and performant. You can easily secure your current workloads running on bare metal by transferring them to Secured Virtual Machines that are using Intel® TDX to encrypt their memory. Intel® TDX on Dell PowerEdge offers easy deployment, low touch enablement, compatibility with existing software, and minimal performance overhead.

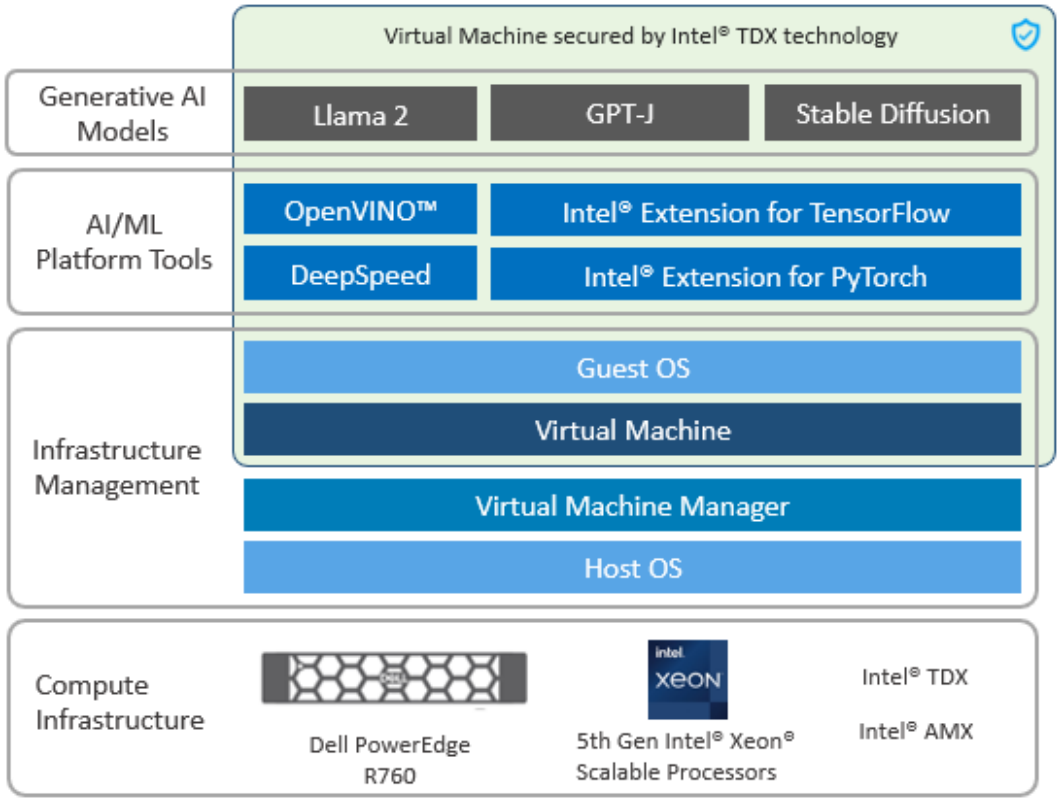

Figure 2. High-level perspective for securing different workloads with Intel® TDX

This solution includes the following hardware and software components:

- **Compute Infrastructure**: Dell PowerEdge R760 Platform with 5th Gen Intel® Xeon® Scalable Processors
- **Infrastructure Management**: Intel® TDX enlightened Host OS, Virtual Machine Manager, and Guest VM using OS that is enlightened for TDX
- **AI/ML Platform Tools & Models**: Deep Learning Frameworks including but not limited to Intel® optimized frameworks like Intel® Extensions for Pytorch, Intel® Extensions for Tensorflow, and OpenVINO; Model Distribution and scaling frameworks like Deep Speed; and AI models like LLaMA2, GPT-J, and Stable Diffusion.

Solution design

This section covers the whole solution that uses Intel® TDX technology to secure the AI workload. We used the Dell PowerEdge R760 with 5th generation Intel® Xeon® processors. On the Dell R760, we installed the latest Ubuntu 23.10 with a new version of Linux kernel. We then enabled Intel® TDX technology, so that we could secure Virtual Machines. Enabling Intel® TDX is covered in Appendix A of this document.

As an example of a workload, we used the LLM inference workload that is running on an optimized version of PyTorch called Intel® Extension for PyTorch with the Llama 2 7B model, leveraging the efficiency of DeepSpeed technology.

The benchmark setup is described in Appendix C.

We compared the LLM results from the normal VM against the secured VM with Intel® TDX.

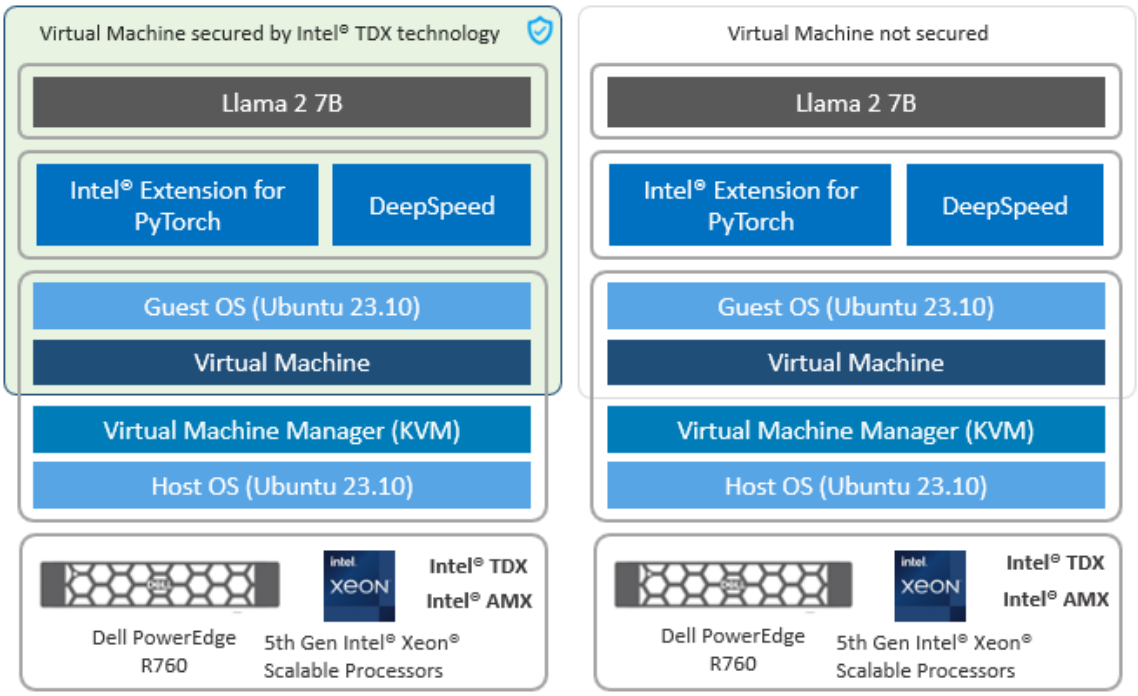

Figure 3. Environment used for this study using Intel® TDX to secure the LLM benchmark

Hardware design

Server

- Dell PowerEdge R760
- 2 x 5th Gen Intel® Xeon® Scalable Processors Platinum 8562Y+
- Memory 512GB (16x32GB DDR5 5600 MT/s)

Storage

- 3x 3.8T Dell NVMe PM1743
- 1x BOSS-N1 with 2x M.2 480GB

Networking

- 2x Intel® E810-XXV Dual Port 10/25GbE SFP28, OCP NIC 3.0
- 4x Intel E810-C Dual Port 100Gb/s

Software design

AI software

The solution design leverages the Intel® Extension for PyTorch, a powerful tool that enhances PyTorch with the latest features and optimizations for superior performance on Intel® hardware. This extension is a critical component of our solution software, providing a significant performance boost. Optimizations take advantage of Intel® Advanced Vector Extensions 512 (Intel® AVX-512), Vector Neural Network Instructions (VNNI), and Intel® Advanced Matrix Extensions (Intel® AMX) on Intel® CPUs, as well as Intel® Xe Matrix Extensions (XMX) AI engines on Intel® discrete GPU.

The solution uses DeepSpeed, an open-source library for deep learning optimization, distributing large models, and scaling. DeepSpeed includes innovations in parallelism technology, custom inference kernels, communication optimizations, and heterogeneous memory technologies.

Large language models

The large language model (LLM) LLama 2 7B was used in the solution as a representative Generative AI model that is popular and widely used in various use cases.

Llama 2 is a family of open-source, pre-trained, and fine-tuned LLMs released by Meta AI in 2023. It is designed for tasks like:

- **Generating text**, including different creative text formats like poems, code, scripts, musical pieces, email, letters, and so on
- **Answering your questions in an informative way**, even if they are open ended, challenging, or strange
- **Engaging in conversations that are informative and comprehensive**

Platform software

The platform software used in our solution includes:

- Host Operating System (OS): Ubuntu 23.10
- Kernel-based Virtual Machine (KVM)
- Guest OS: Ubuntu 23.10

The KVM is configured with 256 GB RAM and utilizes all available cores in a 2-socket platform, amounting to 64 cores. This ensures full resource allocation and performance when running the Llama 2 model.

The use of the KVM is justified by the integration of Intel® TDX technology into our solution. Intel® TDX provides hardware-based isolation on the Virtual Machine level to protect sensitive data and code from external threats. By securing the KVM, Intel® TDX effectively encrypts the memory utilized by the VM, enhancing the overall security of our solution.

Results or findings

This section highlights Gen AI inference workload performance with and without being secured on Intel® 5th Gen Xeon® processors. The objective of benchmarks is twofold:

1. To characterize the performance overhead of securing the AI workload with Intel® TDX
2. To characterize the performance improvement of AI workloads secured with Intel® TDX on Intel® 5th Gen Xeon® processors with built-in AI accelerator Intel® AMX

Test methodology

For a good user experience in interactive applications with an LLM, the next token/2nd token latency is considered a KPI. We measured the next token latency with various input token sizes ranging from small to large represented by 32 and 2k tokens. We also conducted a few experiments in varying the output token length, however that did not significantly impact the KPI of next token latency. As such, we left the output token length

to 32 tokens. We took an average of many runs with each run being 100 individual requests (batch size 1). We measured the 2nd token latency for the Llama 2 7B inference with and without being secured by TDX across fp32, bf16, and int8 precisions in the VM. FP32 uses AVX512 instructions while bf16 and int8 are boosted in performance by Intel® AMX on Xeon® processors.

Test results

The outcome of the tests is tabulated in the following table:

Table 1. ADD TABLE TITLE

	128 Input Tokens/32 Output Tokens			2048 Input Tokens/32 Output Tokens		
Precision	FP32	BF16	INT8	FP32	BF16	INT8
TDX	61.2	35.9	25.63	67.03	39.93	29.33
NO TDX	59.93	34.97	24.93	65.2	38.8	28.83
TDX Overhead	2.12%	2.66%	2.81%	2.81%	2.91%	1.73%

Table 1 presents 5th Gen Intel® Xeon® Platinum 8562Y+ for LLama2-7B performance with TDX vs. without TDX and shows an overhead of less than 3% across fp32, bf16, and int8. For most use cases, such overhead is minimal and acceptable, reinforcing the fact that the benefits of security need not be traded off for performance.

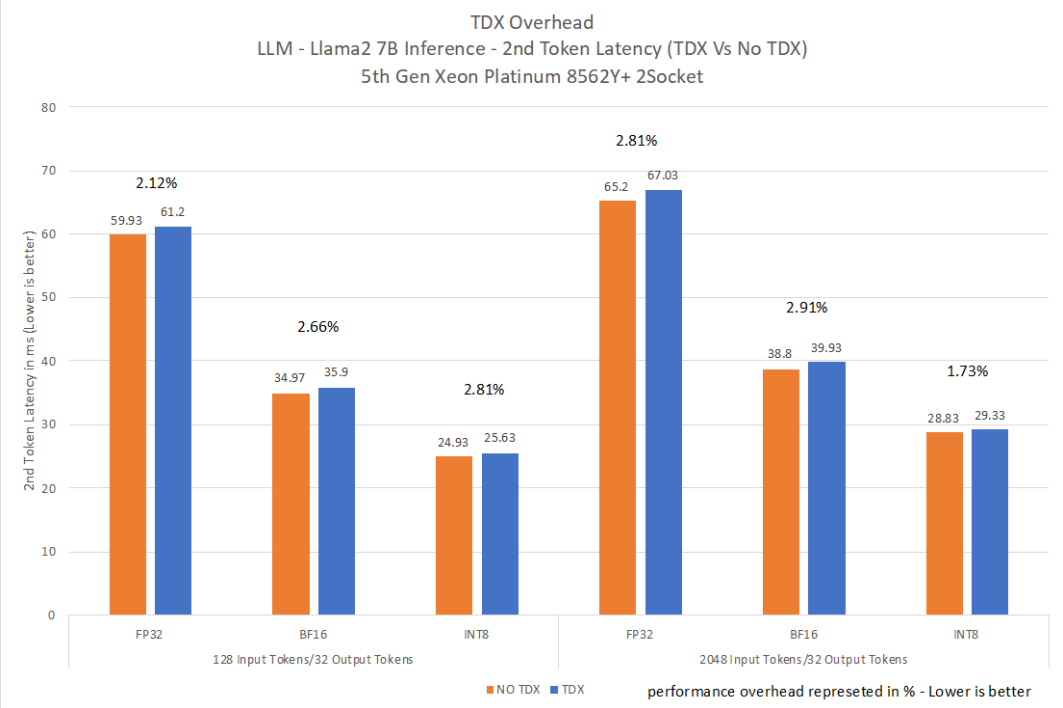

Figure 4. ADD FIGURE TITLE

Figure 4 presents 5th Gen Intel® Xeon® Platinum 8462Y+ for LLama2-7B performance secured with TDX and shows a performance improvement of 1.7x for bf16 and 2.3x for int8 using Intel® AMX over fp32 using Intel® AVX512. The performance acceleration benefits of Intel® AMX for AI workloads continue to be seen even when the security is enabled with TDX.

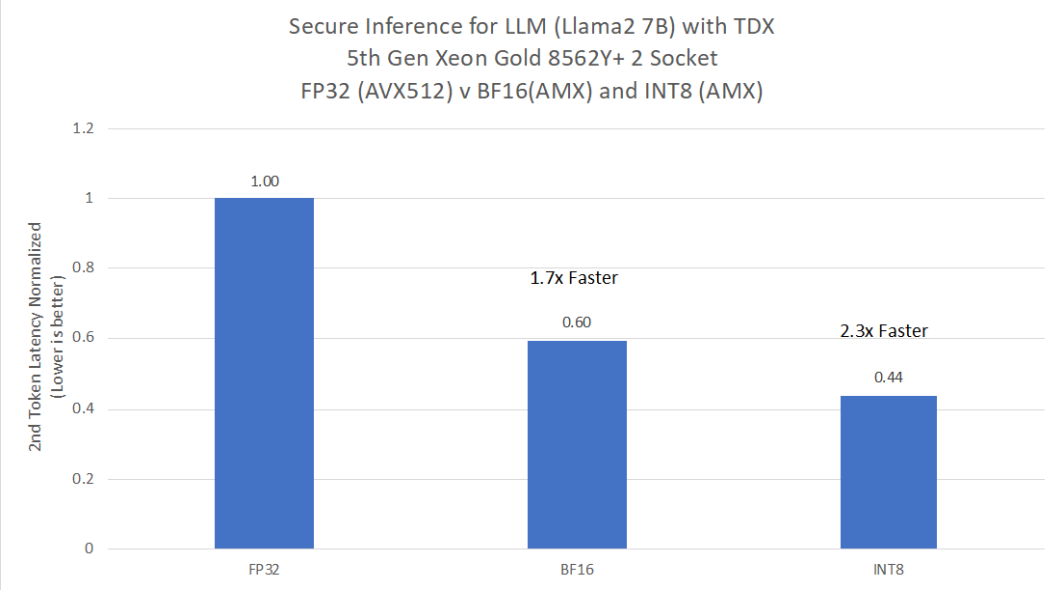

Figure 5. ADD FIGURE TITLE

Benchmark results are highly dependent upon the workload, the specific application requirements, and the system design and implementation. Relative system performance will vary due to these and other factors. Therefore, this workload should not be used as a substitute for a specific customer application benchmark when critical capacity planning and/or product evaluation decisions are contemplated.

All performance data contained in this report was obtained in a rigorously controlled environment. Results obtained in other operating environments may vary significantly. Dell Technologies does not guarantee or represent that a user can or will achieve similar performance results.

Conclusion

This paper highlights the importance of security in AI workloads and explores how Dell PowerEdge 16th generation with 5th Gen Intel® Xeon® platforms can accelerate and secure these workloads for customers embracing Confidential AI.

The key takeaways from this paper are as follows:

- Security should be prioritized and need not be traded off for performance.
- Intel® Xeon® Scalable processors are a great choice for many AI workloads, including Generative AI Inference without the need for specialized hardware.
- Intel® AMX provides significant acceleration for Deep Learning Inference.

- By prioritizing data confidentiality, organizations can mitigate risks, maintain trust, and safeguard sensitive information.
- The overhead of security is minimal with Intel® TDX.

The Dell PowerEdge 16th generation with 5th Gen Intel® Xeon® processors alongside Intel® AMX acceleration offer a cost-effective and efficient solution for AI deployment optimization. This solution enables organizations to effectively run AI workloads without the need for specialized hardware, significantly reducing costs. Additionally, it provides enhanced versatility and simplicity, allowing organizations to run any workload, including AI. To ensure the security of sensitive data and valuable intellectual property without compromising AI performance or creating data silos, organizations can utilize confidential computing with Intel® TDX. Intel® TDX provides a secure environment for data processing while maintaining optimal performance. By embracing confidential computing with Intel® TDX, organizations can effectively protect their sensitive information and intellectual property, enabling them to leverage the power of AI without compromising security.

References

Dell Technologies documentation

The following Dell Technologies documentation provides additional and relevant information related to this solution.

- Enabling Intel TDX on Dell PowerEdge | Technical paper
- TDX on Dell PowerEdge | Infographic
- TDX on Dell PowerEdge: Fraud Detection
- TDX on Dell PowerEdge: RAG for GenAI

Intel® documentation

The following Intel documentation provides greater context.

- *https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/trust-domain-extensions.html*

Appendix A - Enable Intel® TDX on a Dell server

This section describes how to enable Intel® TDX technology on a Dell server.

Instructions

Follow these instructions to enable Intel® TDX technology:

1. Install Ubuntu 23.10, and run script according to documentation from: https://github.com/canonical/tdx?tab=readme-ov-file#setup-tdx-host
   a. Install Ubuntu 23.10
   b. Clone repository:

```
git clone https://github.com/canonical/tdx.git
```

   c. Run script:

```
cd tdx
sudo ./setup-tdx-host.sh
```

2. Enable TDX in BIOS
   a. Follow the instructions in Appendix B
3. Verify if TDX is enabled according to documentation from: https://github.com/canonical/tdx?tab=readme-ov-file#verify-tdx-is-enabled-on-host
   a. Run command:

```
sudo dmesg | grep -i tdx
```

   b. Similar output is expected:

```
[    0.000000] Command line: BOOT_IMAGE=/vmlinuz-6.5.0-1003-intel-opt root=/dev/mapper/ubuntu--vg-ubuntu--lv ro kvm_intel.tdx=1 nomodeset
[    0.855789] Kernel command line: BOOT_IMAGE=/vmlinuz-6.5.0-1003-intel-opt root=/dev/mapper/ubuntu--vg-ubuntu--lv ro kvm_intel.tdx=1 nomodeset
[    1.732532] tdx: BIOS enabled: private KeyID range [32, 64)
[   11.382550] tdx: TDX module: attributes 0x0, vendor_id 0x8086, major_version 1, minor_version 5, build_date 20231008, build_num 595
[   11.382553] tdx: CMR: [0x100000, 0x77800000)
[   11.382553] tdx: CMR: [0x100000000, 0x3ffe000000)
[   11.382554] tdx: CMR: [0x4080000000, 0x8000000000)
[   12.454195] tdx: 2084844 KBs allocated for PAMT.
[   12.454198] tdx: module initialized.
```

4. Create a TDX VM image
   a. Follow the documentation on: https://github.com/canonical/tdx?tab=readme-ov-file#create-a-new-td-guest-image
5. Boot the TDX VM
   a. Follow the documentation on: https://github.com/canonical/tdx?tab=readme-ov-file#boot-td-guest-with-virsh-libvirt
6. Verify the TDX VM

a. SSH to the VM

b. Run the following command to verify TDX:

```
sudo dmesg | grep -i tdx
```

c. Example output:

```
[    0.000000] tdx: Guest detected
[    0.000000] DMI: QEMU Standard PC (Q35 + ICH9, 2009), BIOS 2023.05-2+tdx1.0 11/05/2023
[   14.925052] process: using TDX aware idle routine
[   14.925052] Memory Encryption Features active: Intel TDX
```

Appendix B – Enable Intel® TDX in BIOS

This section describes how to setup BIOS to use Intel® TDX.

Instructions

Follow the instructions below to enable Intel® TDX in the BIOS:

1. Enable Intel® TDX in BIOS
    a. Go to:
    System BIOS -> System Security
    b. Set below values:
        i Intel(R) TXT -> On
        ii Memory Encryption -> Multiple Keys
        iii Global Memory Integrity -> Disabled
        iv TME Encryption Bypass -> Enabled
        v Intel Trust Domain Extension (TDX) -> Enabled
        vi TME-MT/TDX Key Split to non-zero value -> 1 (set nonzero value)
        vii Disable excluding Mem below 1MB in CMR -> Disabled
        viii TDX Secure Arbitration Mode Loader (SEAM) -> Enabled
        ix Intel(R) SGX -> On
        x PRMRR Size -> 2G

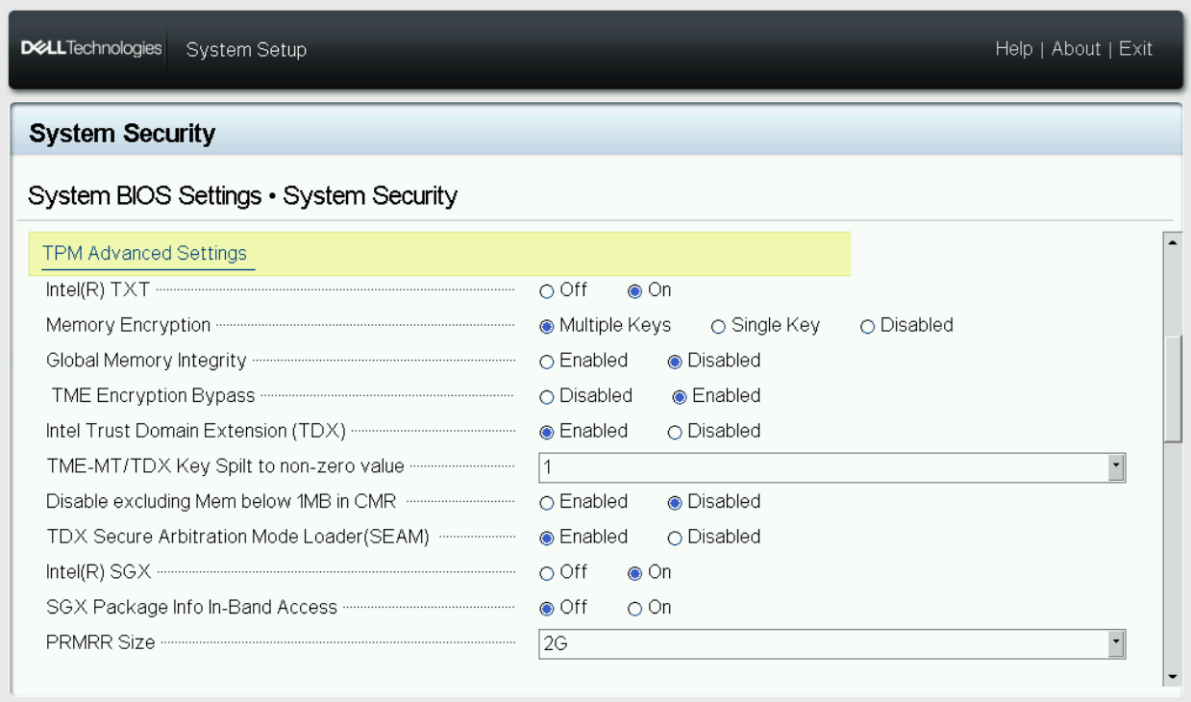

Figure 6. ADD FIGURE TITLE

Appendix C – Benchmark setup

This section describes how to prepare a benchmark setup.

Instructions

Follow the instructions below to setup benchmark environment in VMs:

1. Based on Appendix A, use the same VM images for benchmarking.
2. Define VMs with the following commands (XML definitions are in Appendix D):
    a. `virsh define tdx.xml`
    b. `virsh define no-tdx.xml`
3. Start VM with `virsh start` command and login to it over SSH.
4. Prepare benchmark environment using point 1, 2b, and 3 from instructions: https://github.com/intel/intel-extension-for-pytorch/tree/v2.1.100%2Bcpu/examples/cpu/inference/python/llm#environment-setup
5. Request access to Llama 2 model from: https://huggingface.co/meta-llama/Llama-2-7b-hf
6. Generate Hugging Face token from: https://huggingface.co/settings/tokens
7. Log into Hugging Face with token in VM:

```
huggingface-cli login --token $HUGGINGFACE_TOKEN
```

8. Run benchmark with commands and adjust parameters to your needs:
    a. For int8 precision:

```
deepspeed --bind_cores_to_rank run.py \
  --benchmark --ipex --deployment-mode \
  --ipex-weight-only-quantization \
  --token-latency --greedy --autotp \
  --shard-model --output-dir "saved_results" \
  --int8 \
  --model-name-or-path meta-llama/Llama-2-7b-hf \
```

    b. For bf16 and fp32 precision:

```
deepspeed --bind_cores_to_rank run.py \
  --benchmark --ipex --deployment-mode \
  --token-latency --greedy --autotp \
  --shard-model --output-dir "saved_results" \
  --dtype ${PRECISION} \
  --model-name-or-path meta-llama/Llama-2-7b-hf \
  --num-warmup ${NUM_WARMUP} --num-iter ${NUM_ITER} \
```

Appendix D – Definitions for Virtual Machines

This section describes Virtual Machines definition for KVM.

Instructions

Following are the definitions of Virtual Machine for KVM:

- tdx.xml:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
 <name>tdx</name>
 <memory unit='GiB'>256</memory>
 <memoryBacking>
  <source type='anonymous'/>
  <access mode='private'/>
 </memoryBacking>
 <vcpu placement='static' cpuset='0-63'>64</vcpu>
 <cputune>
  <vcpupin vcpu='0' cpuset='0'/>
  <vcpupin vcpu='1' cpuset='2'/>
  <vcpupin vcpu='2' cpuset='4'/>
  <vcpupin vcpu='3' cpuset='6'/>
  <vcpupin vcpu='4' cpuset='8'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='12'/>
  <vcpupin vcpu='7' cpuset='14'/>
  <vcpupin vcpu='8' cpuset='16'/>
  <vcpupin vcpu='9' cpuset='18'/>
  <vcpupin vcpu='10' cpuset='20'/>
  <vcpupin vcpu='11' cpuset='22'/>
  <vcpupin vcpu='12' cpuset='24'/>
  <vcpupin vcpu='13' cpuset='26'/>
  <vcpupin vcpu='14' cpuset='28'/>
  <vcpupin vcpu='15' cpuset='30'/>
  <vcpupin vcpu='16' cpuset='32'/>
  <vcpupin vcpu='17' cpuset='34'/>
  <vcpupin vcpu='18' cpuset='36'/>
  <vcpupin vcpu='19' cpuset='38'/>
  <vcpupin vcpu='20' cpuset='40'/>
  <vcpupin vcpu='21' cpuset='42'/>
  <vcpupin vcpu='22' cpuset='44'/>
  <vcpupin vcpu='23' cpuset='46'/>
  <vcpupin vcpu='24' cpuset='48'/>
  <vcpupin vcpu='25' cpuset='50'/>
  <vcpupin vcpu='26' cpuset='52'/>
  <vcpupin vcpu='27' cpuset='54'/>
  <vcpupin vcpu='28' cpuset='56'/>
  <vcpupin vcpu='29' cpuset='58'/>
```

```
  <vcpupin vcpu='30' cpuset='60'/>
  <vcpupin vcpu='31' cpuset='62'/>
  <vcpupin vcpu='32' cpuset='1'/>
  <vcpupin vcpu='33' cpuset='3'/>
  <vcpupin vcpu='34' cpuset='5'/>
  <vcpupin vcpu='35' cpuset='7'/>
  <vcpupin vcpu='36' cpuset='9'/>
  <vcpupin vcpu='37' cpuset='11'/>
  <vcpupin vcpu='38' cpuset='13'/>
  <vcpupin vcpu='39' cpuset='15'/>
  <vcpupin vcpu='40' cpuset='17'/>
  <vcpupin vcpu='41' cpuset='19'/>
  <vcpupin vcpu='42' cpuset='21'/>
  <vcpupin vcpu='43' cpuset='23'/>
  <vcpupin vcpu='44' cpuset='25'/>
  <vcpupin vcpu='45' cpuset='27'/>
  <vcpupin vcpu='46' cpuset='29'/>
  <vcpupin vcpu='47' cpuset='31'/>
  <vcpupin vcpu='48' cpuset='33'/>
  <vcpupin vcpu='49' cpuset='35'/>
  <vcpupin vcpu='50' cpuset='37'/>
  <vcpupin vcpu='51' cpuset='39'/>
  <vcpupin vcpu='52' cpuset='41'/>
  <vcpupin vcpu='53' cpuset='43'/>
  <vcpupin vcpu='54' cpuset='45'/>
  <vcpupin vcpu='55' cpuset='47'/>
  <vcpupin vcpu='56' cpuset='49'/>
  <vcpupin vcpu='57' cpuset='51'/>
  <vcpupin vcpu='58' cpuset='53'/>
  <vcpupin vcpu='59' cpuset='55'/>
  <vcpupin vcpu='60' cpuset='57'/>
  <vcpupin vcpu='61' cpuset='59'/>
  <vcpupin vcpu='62' cpuset='61'/>
  <vcpupin vcpu='63' cpuset='63'/>
 </cputune>
 <cpu mode='host-passthrough'>
  <cache mode='passthrough'/>
  <numa>
   <cell id='0' cpus='0-31' memory='128' unit='GiB'/>
   <cell id='1' cpus='32-63' memory='128' unit='GiB'/>
  </numa>
 </cpu>
 <numatune>
  <memory mode='strict' nodeset='0-1'/>
  <memnode cellid='0' mode='strict' nodeset='0'/>
  <memnode cellid='1' mode='strict' nodeset='1'/>
```

```
</numatune>
<os>
  <type arch='x86_64' machine='q35'>hvm</type>
  <loader>/usr/share/qemu/OVMF.fd</loader>
  <boot dev='hd'/>
</os>
<features>
  <acpi/>
  <apic/>
  <ioapic driver='qemu'/>
</features>
<clock offset='utc'>
  <timer name='hpet' present='no'/>
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<pm>
  <suspend-to-mem enable='no'/>
  <suspend-to-disk enable='no'/>
</pm>
<devices>
  <emulator>/usr/bin/qemu-system-x86_64</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2'/>
    <source file='/root/tdx-guest-ubuntu-23.10.qcow2'/>
    <target dev='vda' bus='virtio'/>
  </disk>
  <interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
  </interface>
  <console type='pty'>
    <target type='virtio' port='1'/>
  </console>
  <channel type='unix'>
    <source mode='bind'/>
    <target type='virtio' name='org.qemu.guest_agent.0'/>
  </channel>
</devices>
<allowReboot value='no'/>
<launchSecurity type='tdx'>
  <policy>0x10000001</policy>
</launchSecurity>
<qemu:commandline>
  <qemu:arg value='-cpu'/>
```

```
  <qemu:arg value='host'/>
 </qemu:commandline>
</domain>
```

- no-tdx.xml:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
 <name>no-tdx</name>
 <memory unit='GiB'>256</memory>
 <memoryBacking>
  <source type='anonymous'/>
  <access mode='private'/>
 </memoryBacking>
 <vcpu placement='static' cpuset='0-31,32-63'>64</vcpu>
 <cputune>
  <vcpupin vcpu='0' cpuset='0'/>
  <vcpupin vcpu='1' cpuset='2'/>
  <vcpupin vcpu='2' cpuset='4'/>
  <vcpupin vcpu='3' cpuset='6'/>
  <vcpupin vcpu='4' cpuset='8'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='12'/>
  <vcpupin vcpu='7' cpuset='14'/>
  <vcpupin vcpu='8' cpuset='16'/>
  <vcpupin vcpu='9' cpuset='18'/>
  <vcpupin vcpu='10' cpuset='20'/>
  <vcpupin vcpu='11' cpuset='22'/>
  <vcpupin vcpu='12' cpuset='24'/>
  <vcpupin vcpu='13' cpuset='26'/>
  <vcpupin vcpu='14' cpuset='28'/>
  <vcpupin vcpu='15' cpuset='30'/>
  <vcpupin vcpu='16' cpuset='32'/>
  <vcpupin vcpu='17' cpuset='34'/>
  <vcpupin vcpu='18' cpuset='36'/>
  <vcpupin vcpu='19' cpuset='38'/>
  <vcpupin vcpu='20' cpuset='40'/>
  <vcpupin vcpu='21' cpuset='42'/>
  <vcpupin vcpu='22' cpuset='44'/>
  <vcpupin vcpu='23' cpuset='46'/>
  <vcpupin vcpu='24' cpuset='48'/>
  <vcpupin vcpu='25' cpuset='50'/>
  <vcpupin vcpu='26' cpuset='52'/>
  <vcpupin vcpu='27' cpuset='54'/>
  <vcpupin vcpu='28' cpuset='56'/>
  <vcpupin vcpu='29' cpuset='58'/>
  <vcpupin vcpu='30' cpuset='60'/>
```

```
  <vcpupin vcpu='31' cpuset='62'/>
  <vcpupin vcpu='32' cpuset='1'/>
  <vcpupin vcpu='33' cpuset='3'/>
  <vcpupin vcpu='34' cpuset='5'/>
  <vcpupin vcpu='35' cpuset='7'/>
  <vcpupin vcpu='36' cpuset='9'/>
  <vcpupin vcpu='37' cpuset='11'/>
  <vcpupin vcpu='38' cpuset='13'/>
  <vcpupin vcpu='39' cpuset='15'/>
  <vcpupin vcpu='40' cpuset='17'/>
  <vcpupin vcpu='41' cpuset='19'/>
  <vcpupin vcpu='42' cpuset='21'/>
  <vcpupin vcpu='43' cpuset='23'/>
  <vcpupin vcpu='44' cpuset='25'/>
  <vcpupin vcpu='45' cpuset='27'/>
  <vcpupin vcpu='46' cpuset='29'/>
  <vcpupin vcpu='47' cpuset='31'/>
  <vcpupin vcpu='48' cpuset='33'/>
  <vcpupin vcpu='49' cpuset='35'/>
  <vcpupin vcpu='50' cpuset='37'/>
  <vcpupin vcpu='51' cpuset='39'/>
  <vcpupin vcpu='52' cpuset='41'/>
  <vcpupin vcpu='53' cpuset='43'/>
  <vcpupin vcpu='54' cpuset='45'/>
  <vcpupin vcpu='55' cpuset='47'/>
  <vcpupin vcpu='56' cpuset='49'/>
  <vcpupin vcpu='57' cpuset='51'/>
  <vcpupin vcpu='58' cpuset='53'/>
  <vcpupin vcpu='59' cpuset='55'/>
  <vcpupin vcpu='60' cpuset='57'/>
  <vcpupin vcpu='61' cpuset='59'/>
  <vcpupin vcpu='62' cpuset='61'/>
  <vcpupin vcpu='63' cpuset='63'/>
 </cputune>
 <cpu mode='host-passthrough'>
  <cache mode='passthrough'/>
  <numa>
    <cell id='0' cpus='0-31' memory='128' unit='GiB'/>
    <cell id='1' cpus='32-63' memory='128' unit='GiB'/>
  </numa>
 </cpu>
 <numatune>
  <memory mode='strict' nodeset='0-1'/>
  <memnode cellid='0' mode='strict' nodeset='0'/>
  <memnode cellid='1' mode='strict' nodeset='1'/>
 </numatune>
```

```
<os>
  <type arch='x86_64' machine='q35'>hvm</type>
  <loader>/usr/share/qemu/OVMF.fd</loader>
  <boot dev='hd'/>
</os>
<features>
  <acpi/>
  <apic/>
  <ioapic driver='qemu'/>
</features>
<clock offset='utc'>
  <timer name='hpet' present='no'/>
</clock>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>destroy</on_crash>
<pm>
  <suspend-to-mem enable='no'/>
  <suspend-to-disk enable='no'/>
</pm>
<devices>
  <emulator>/usr/bin/qemu-system-x86_64</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2'/>
    <source file='/root/no-tdx-guest-ubuntu-23.10.qcow2'/>
    <target dev='vda' bus='virtio'/>
  </disk>
  <interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
  </interface>
  <console type='pty'>
    <target type='virtio' port='1'/>
  </console>
  <channel type='unix'>
    <source mode='bind'/>
    <target type='virtio' name='org.qemu.guest_agent.0'/>
  </channel>
</devices>
<allowReboot value='no'/>
<qemu:commandline>
  <qemu:arg value='-cpu'/>
  <qemu:arg value='host'/>
</qemu:commandline>
</domain>
```