An Analysis of AWS Nitro Enclaves for Database Workloads

Adrian Lutsch
Technical University of Darmstadt
Darmstadt, Germany

Christian Franck
Technical University of Darmstadt
Darmstadt, Germany

Muhammad El-Hindi
Technical University of Darmstadt
Darmstadt, Germany

Zsolt István
Technical University of Darmstadt
Darmstadt, Germany

Carsten Binnig
Technical University of Darmstadt
Darmstadt, Germany
DFKI
Darmstadt, Germany

Abstract

Cloud databases have become prevalent, as evidenced by the rapid growth of systems such as BigQuery, Snowflake, and Databricks. Concurrently, there has been a significant increase in the requirements for secure data processing when outsourcing databases to the cloud. For this, Trusted Execution Environments (TEEs) have emerged as a key technology in the cloud, which is witnessed by the fact that all cloud providers offer TEEs in their service portfolios. However, Amazon Web Services' (AWS) approach to TEEs based on Nitro Enclaves fundamentally differs from that of other cloud providers like Microsoft and Google or standard technologies such as Intel SGX. In this paper, we thus set out the goal to understand the implications of using AWS Nitro Enclaves for cloud databases. Although Nitro Enclaves initially appear to be a promising platform for pure TEE performance, they come with significant limitations regarding communication with the Nitro Enclave. Our benchmark results provide insight into the performance and practical challenges of deploying database workloads in AWS Nitro Enclaves, offering valuable guidance for practitioners and researchers.

CCS Concepts

• **Information systems** → **Database performance evaluation**; • **Security and privacy** → **Database and storage security**; *Information accountability and usage control.*

Keywords

Database performance, TEE, AWS Nitro Enclaves

ACM Reference Format:
Adrian Lutsch, Christian Franck, Muhammad El-Hindi, Zsolt István, and Carsten Binnig. 2025. An Analysis of AWS Nitro Enclaves for Database Workloads. In *21st International Workshop on Data Management on New Hardware (DaMoN '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3736227.3736234

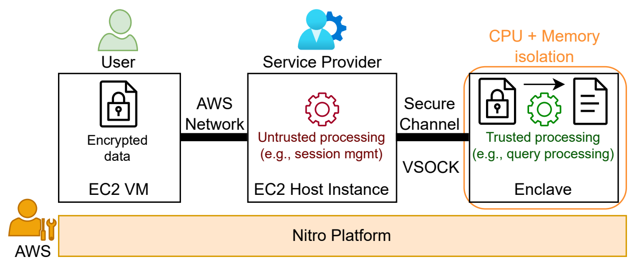

Figure 1: Using Nitro Enclaves for Cloud DBMSs: Nitro Enclaves provide separation from application service providers, but customers need to trust AWS. To enforce strong isolation, Nitro Enclaves communicate solely through a local VSOCK.

1 Introduction

Secure data processing in the cloud. Public clouds have become prevalent for database management systems (DBMSs), as evidenced by the rapid growth of systems such as BigQuery, Snowflake, and Databricks. While cloud DBMSs offer many benefits, including on-demand scaling and access to immense compute and storage capacity, the cloud model places a significant responsibility on the service provider regarding data security [26]. Customers must trust service providers to keep their data safe and protect against breaches or data corruption – a trust that is not always warranted, as recent attacks show [9, 30]. Such incidents are particularly concerning, given the increasing cloud DBMS trend.

Trusted Execution Environments. A key technology to enable secure and high-performance data processing in the cloud is Trusted Execution Environments (TEEs) [1, 12, 14, 16, 21, 22, 33]. TEEs create isolated environments, or enclaves, where sensitive data can be processed securely, protecting data and code from potentially privileged attackers. Various TEE technologies and implementations exist, including hardware-based solutions from Intel and AMD and cloud-based offerings from Amazon Web Services (AWS). These technologies provide different security guarantees, such as hardware-enforced encryption, integrity protection, and freshness verification [35], making it hard to navigate the TEE landscape.

Relevance of AWS Nitro Enclaves. While hardware-based TEEs such as Intel SGX [14] and AMD SEV-SNP [1] have been widely researched in academia and industry [13, 14, 16, 21, 22, 25, 27], AWS Nitro Enclaves, announced in 2020, remain heavily underexplored for cloud DBMSs. Unlike hardware-based TEEs, which require customers to trust only the hardware and offer hardware-level protections such as memory encryption, AWS Nitro Enclaves are built around the assumption of AWS as a trusted infrastructure provider, leading to different security guarantees. Moreover, their architecture and performance characteristics are still poorly understood. Hence, this paper presents the first systematic analysis of AWS Nitro Enclaves for cloud DBMS.

Lacking integrity & privacy guarantees. Nitro Enclaves are virtual machines (VMs) based on AWS' Nitro Platform [2]. AWS controls the underlying hardware and software, including the hypervisor, which impacts the security guarantees. In particular, AWS can (in theory) inject modified firmware or hypervisor updates, affecting system integrity. Unlike other TEEs, such as Intel SGX, Nitro Enclaves do not encrypt data in memory, which opens up privacy attacks. For example, a compromised Nitro system could allow AWS to extract a memory dump, compromising data confidentiality.

What do we get from Nitro Enclaves? Nitro Enclaves provide separation from application service providers. For example, when a cloud database service provider, such as Snowflake, runs its DBMS software in Nitro Enclaves, it cannot – unlike today – access customer data during processing. Additionally, like other major TEEs, Nitro Enclaves support attestation, allowing customers to verify the integrity of the software inside an enclave. In a data processing scenario, this ensures that a client application connects to a trusted DBMS instance before storing or processing data.

Using Nitro enclaves in cloud DBMSs. Figure 1 illustrates how Nitro Enclaves could be used in a cloud DBMS. Notably, Nitro Enclaves do not support IP networking and persistent storage inside the enclave [11]. Instead, they communicate solely through a local socket (VSOCK in Figure 1) with the so-called *EC2 host instance.* To communicate with external components, e.g., database clients or I/O devices, developers must split their application into a trusted component (running inside the enclave) and an untrusted component (running on the host) handling all the external I/O traffic and forwarding it to the enclave. The restriction to a local socket connection has architectural implications and performance overheads, which have not been studied thoroughly.

Contributions. This paper presents the first database-centric analysis of AWS Nitro Enclaves, systematically deriving implications for database system design and implementation. To that end, we provide the following contributions: First, since AWS Nitro Enclaves are not well understood in the database community, we discuss their position in the TEE landscape and high-level architecture in Sections 2 and 3. Second, we conduct database-centric benchmarks (macro- and micro-level) to reveal the performance overhead of Nitro Enclaves and, particularly, the VSOCK approach in different settings (Section 4). Third, we report on practical insights for DBMS design beyond pure performance results (Section 5). All code and data used in this paper are publicly available at https://github.com/DataManagementLab/nitro-enclaves-benchmarks.

2 Trusted Execution Environments Landscape

Trusted Execution Environments (TEEs) provide isolated environments for securely processing data. However, TEEs differ in many dimensions, including the size and scope of the Trusted Computing Base (TCB), trust assumptions, threat models, memory encryption, and the root of trust. These differences affect system design, particularly in the context of DBMSs. In the following, we discuss these differences and compare AWS Nitro Enclaves with process-based TEEs and confidential VM technologies.

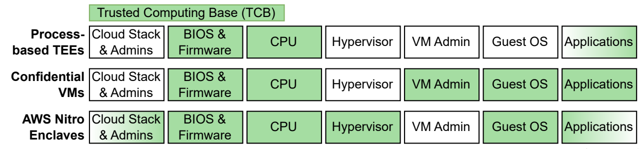

Figure 2: Comparison of Trusted Computing Base (TCB) of different TEE approaches. Cloud hardware and software stack from left to right. TCB is marked green for the different TEE technologies. Nitro Enclaves have the largest TCB.

Process-based TEEs like Intel's Software Guard Extensions (SGX) (Figure 2 top row) have the smallest TCB, limited to the enclave's code, the CPU, and its firmware (marked green in Figure 2). A smaller TCB reduces the risk of vulnerabilities and improves security assurance. However, because of the very small TCB that does not include the operating system, system calls are not supported inside SGX enclaves, making it difficult to integrate with existing DBMSs without re-architecting them into a trusted and untrusted component. To achieve their security guarantees, Intel SGX enclaves are isolated at the CPU level, with the CPU as the root of trust. The CPU ensures that only enclave code can access plaintext data. Hardware-level memory encryption and access control protect the confidentiality of enclave data even from privileged components such as administrators, operating systems, or hypervisors.

Confidential Virtual Machines (VMs) (e.g., Intel TDX [12], AMD SEV [1]) extend TEE protections to entire VMs. Unlike process-based TEEs, confidential VMs (Figure 2 middle) support off-the-shelf guest operating systems, facilitating DBMS deployment without a re-design. These TEEs encrypt VM memory and isolate it from the hypervisor and other VMs, preventing a compromised hypervisor from accessing the VM's memory. However, as shown in Figure 2 in green, their TCB includes the entire OS and software stack, increasing the attack surface. As a result, confidential VMs cannot protect against rouge VM administrators or backdoors in the OS.

AWS Nitro Enclaves combine aspects of process-based TEEs and confidential VMs (Figure 2, bottom row). Like confidential VMs, they run a full kernel and OS, which eases DBMS integration. However, instead of isolating the VMs from the hypervisor via special CPU mechanisms and memory encryption, Nitro enclaves rely on a trusted hypervisor to enforce isolation between enclaves and the host. Like process-based TEEs, AWS Nitro Enclaves are further isolated by design constraints such as the absence of persistent storage and network access (see Figure 1). In contrast to existing process-based TEEs and confidential VMs, Nitro Enclaves do not inherently include memory encryption or hardware-based protection against physical attacks. Their security depends on the Nitro infrastructure, including the hypervisor and management firmware. If the underlying hardware supports memory encryption, such as Intel TME-MK [10], Nitro Enclaves can inherit this functionality.

Overall, and as illustrated in Figure 2, Nitro Enclaves' trust model occupies a unique place in the TEE landscape. Unlike confidential virtual machines, VM Admins (i.e., database service providers) do not need to be trusted. In exchange, they require trust in the hypervisor and the cloud operator. This is a valid trust model for customers trusting AWS and looking for, e.g., database services built on top of AWS's infrastructure, such as Snowflake or Firebolt.

3 How do AWS Nitro Enclaves Work?

To clarify how Nitro Enclaves can be used in DBMSs and provide the required background for our benchmarks, we discuss the inner workings of AWS Nitro Enclaves in the following.

3.1 The Nitro System

As a basis for their TEE technology, AWS uses the Nitro System that offloads key functions for managing the cloud infrastructure to dedicated hardware. The Nitro System consists of three key components that enforce the security guarantees of AWS EC2 instances: The Nitro Cards, the Nitro Security Chip, and the Nitro Hypervisor. The Nitro Cards and the Nitro Security Chip are dedicated and CPU vendor-independent hardware that enables AWS to create and manage VMs, update firmware, and keep control of their hardware even if they give customers bare-metal access without a hypervisor. For example, the nitro security chip prevents customers from changing CPU firmware [2]. The Nitro Hypervisor is an AWS-specific hypervisor that manages the hardware resources between the EC2 instances as instructed by the Nitro Card. Overall, these three components assure AWS' control over their infrastructure and isolate clients using their hardware [2].

3.2 AWS Nitro Enclaves

Next, we detail how AWS Nitro Enclaves are created and used.

Enclave creation & isolation. AWS Nitro Enclaves operate as virtual machines with their own Linux kernel, independent of the host VM. When the Nitro System launches a Nitro Enclave, it takes configurable number of CPU cores and memory pages from the host VM and assigns them to the enclave. This isolates and protects the enclave from the host VM, ensuring that sensitive data processing remains secure. As such, Nitro Enclaves inherit the isolation and security properties of the Nitro system.

Programming model. Nitro Enclaves are created from enclave images. Customers can create enclave images from Docker images containing the enclave application. AWS provides a command-line tool, nitro-cli [4], to compile Docker images into enclave image files by adding a kernel. As mentioned before, Nitro Enclaves do not have a normal networking interface or storage access. They can only communicate with their EC2 host instance via a VSOCK interface [5]. VSOCK was originally designed for communication between a hypervisor and its virtual machines. AWS repurposed it for communication between an enclave and its host EC2 instance. VSOCK uses 32-bit context IDs and ports as identifiers and enables streaming communication between two sockets, similar to TCP. Because of the restriction to VSOCK communication, enclave DBMS require a host application as a proxy for networking and storage access. This is similar to Intel SGX, which requires the application to split into the secure enclave and the insecure host process communicating with the OS. If a DBMS does not support VSOCK, it also requires a proxy inside the enclave that translates from VSOCK to TCP/UDP.

To summarize, the following steps are required to run a DBMS inside a Nitro Enclave: (1) Create a Docker image that contains the DBMS and, if required, a VSOCK proxy for communication. (2) Convert the Docker image to an enclave image using nitro-cli. (3) Run the enclave and a network and storage access proxy on the host instance.

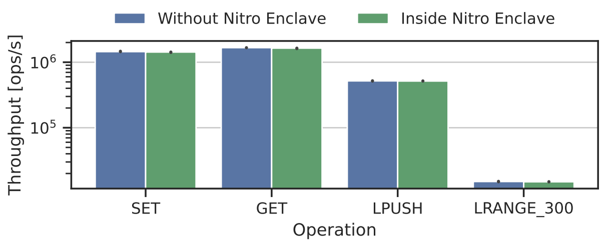

Figure 3: Performance comparison of running the Redis benchmark via local loopback without an enclave vs. inside an enclave. Performance is the same in both cases.

4 Performance Evaluation of Nitro Enclaves

In the following, we present the results of our performance analysis of database workloads in Nitro Enclaves. To the best of our knowledge, we are the first to investigate the performance implications of AWS Nitro Enclaves in detail. We start by benchmarking a full DBMS inside a Nitro Enclave in Section 4.1 and then investigate the underlying causes of performance regressions with micro-benchmarks in Section 4.2. We used various standard AWS EC2 instances for all our experiments, specifying the instance types in the descriptions. Each experiment was executed at least 9 times, and we report the arithmetic mean unless otherwise stated.

4.1 Running a DBMS in Nitro Enclaves

First, we provide a high-level picture of Nitro Enclave performance for DBMS workloads. Since AWS Nitro Enclaves lack direct storage access, we decided to use Redis [28]. Redis is an in-memory key-value store with optional disk storage and has a standardized benchmark suite – Redis benchmark – that is relatively network-heavy [28]. First, we measure performance with the server and clients inside the enclave, isolating any effects of the VSOCK interface. We then move the clients outside the enclave and keep Redis inside to account for VSOCK-based communication. For all experiments in this section, we used c6in.16xlarge instances with 32 cores, 64 threads, and 100 Gbit/s network interface.

Client and DBMS inside Enclave. We compare the Redis performance inside a Nitro Enclave to its performance on the host instance (outside the enclave) using the same hardware. Figure 3 shows a representative subset of results. It compares the throughput of four database operations between both settings. Because Nitro Enclaves do not impose additional overhead for in-memory processing, the throughput inside and outside the enclave is the same (as expected).

Client outside Enclave. Next, we investigate the performance effects of running the Redis server inside a Nitro Enclave while keeping the clients outside – a setup resembling database-as-a-service. Because Redis does not support VSOCK-based communication, we use proxies, as suggested by AWS [4], to translate between VSOCK and Redis' TCP connections. Specifically, a proxy inside the enclave connects VSOCK to Redis' TCP interface, and another proxy on the host forwards incoming TCP traffic to the enclave. We use the general-purpose tool socat [15] as the proxy for our experiments.

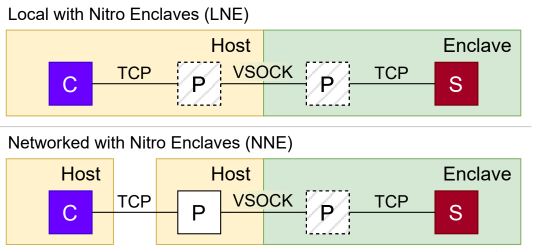
Local with Nitro Enclaves (LNE)
Host
Enclave
C
TCP
P
VSOCK
P
TCP
S
Networked with Nitro Enclaves (NNE)
Host
Host
Enclave
C
TCP
P
VSOCK
P
TCP
S

Figure 4: Experiment settings used in this paper. (C)lients and (S)ervers are located either on a normal EC2 instance (Host) or inside an enclave. The hatched (P)roxies are not required in VSOCK-enabled applications.

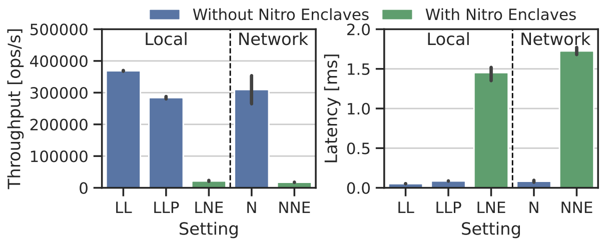

Figure 5: Local Redis performance (without networking) vs. with networking. Running the Redis server in a Nitro Enclave (green bars) reduces performance by more than 10x compared to running it without a Nitro Enclave (blue bars).

The nitro-cli repository also contains a VSOCK to TCP proxy [3], but it only allows outgoing connections and is thus unsuitable for DBMS servers. To isolate the performance effects of the network, the VSOCK connection, and the proxies, we vary the placement of both client and server, as detailed below. The main settings for this experiment are depicted in Figure 4:

- *Local with Nitro Enclaves (LNE):* The clients run on the host instance, while the server runs inside the enclave and communicates with the clients via proxies. This setup does not involve external network traffic.
- *Networked with Nitro Enclaves (NNE):* The server again runs in an enclave, but the clients run on a different EC2 instance in the same availability zone (AZ). This setup introduces external network traffic.

To compare these settings against non-enclave scenarios, we use the following baselines:

- *Local without Nitro Enclaves (LL&LLP):* Both client and server run on the same instance and communicate only via local loopback, either without proxies (LL) or with proxies (LLP).
- *Networked without Nitro Enclaves (N):* The Clients and the server run on different instances in the same AZ and connect via TCP.

We focus on Redis' Set operation in this experiment because its short latency reveals network overhead and it shows stable performance across repeated runs. We configure the benchmark to use 10 clients with a pipeline depth of 3.

The benchmark results are depicted in Figure 5. Comparing the settings with Nitro Enclaves (green) to those without enclaves (blue) reveals throughput (left) and latency (right) degradations of more than an order of magnitude when enclaves are used. To analyze the overhead source further, we examine the LLP setting that includes a TCP to TCP socat proxy. It shows that the proxy only introduces a minor reduction in throughput and a minor increase in latency. Accounting for a network overhead with the N setting shows only a slight performance degradation, too. Overall, it is clear that the transition to the Nitro Enclave in the Redis benchmark introduces a far larger performance penalty than the proxy or network alone.
Performance insight. Running the Redis server inside an enclave reduces the benchmark throughput by up to 92.5 % and increases latency by up to 20x. Data processing inside the enclave is not slower. Thus, the slowdown must be caused by translating between TCP and VSOCK and by the VSOCK transfer between the host instance and enclave. The following section investigates the throughput and latency of the VSOCK interface in more detail with micro-benchmarks.
Practical insight. A notable side effect of the VSOCK proxy setup in this experiment is its reduced stability: The higher the number of clients, the more benchmark runs fail when proxied. We attribute these failures to the implementation of socat. Consequently, production-grade DBMSs running in Nitro Enclaves require a more robust proxy implementation.

4.2 Micro-Benchmarks for VSOCK

As shown in the previous section, the VSOCK interface between Nitro Enclaves and their host instance significantly decreases performance for a full DBMS inside the enclave. Motivated by this result, we quantify the performance characteristics of the VSOCK interface in isolation. We investigate the interface's throughput using a VSOCK-enabled fork of the well-known network performance benchmark tool iperf3 [29]. In addition, we implement a VSOCK client-server application to measure the round-trip latency for varying message sizes.
Influence of HW, connection count and direction. We begin with analyzing the throughput depending on multiple potential influence factors: (1) We vary the EC2 instance size influencing the number of CPU cores and the instance network interface speed to understand the effect of different hardware characteristics. (2) Since multiple connections are often required to saturate client-server interfaces, we also vary the number of VSOCK connections (potentially multiplexing connections on a single CPU core for small instance sizes). (3) We vary the communication direction to analyze its impact on throughput. We always create one enclave per host instance with half of the host instance's CPU cores. Thus, enclaves grow with instance size.

From the experiment results in Figure 6, we can draw four conclusions: (1) Similar to communication over regular sockets, multiple threads and connections are required to achieve maximal throughput. In our experiments, the maximum throughput is generally achieved with 4 to 8 threads, independent of the number of CPU cores the host instance and the enclave have. (2) VSOCK throughput does not increase with instance size. We measured the fastest VSOCK throughput for the relatively small 2xlarge instances and the lowest for the relatively large 8xlarge instances. Thus, the enclave performance of instances with high CPU core count will likely be limited by VSOCK throughput. (3) When considering the communication direction, we can observe that throughput into the

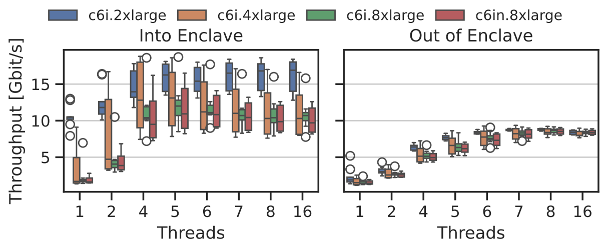

Figure 6: Comparison of host-enclave VSOCK throughput between different instance types. Throughput does not increase with instance size and network capability. Throughput increases with number of threads until 4-8 threads. Throughput is asymmetric: ingress is faster than egress.

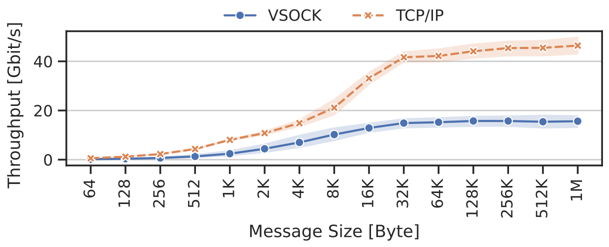

Figure 7: VSOCK and TCP/IP throughput depending on message size. Optimal throughput is achieved for message sizes larger than 32 kB.

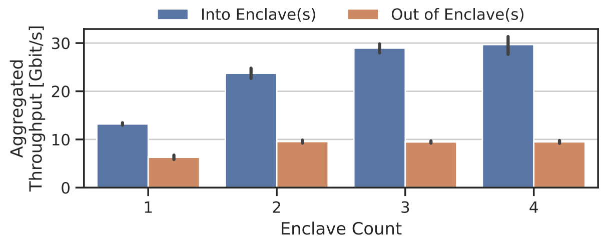

Figure 8: Cumulative VSOCK throughput when increasing the number of enclaves per host instance. Combined ingress increases until three concurrent enclaves. Combined egress increases for two enclaves.

enclave is up to 2 times higher than throughput for communication from the enclave to the host instance. (4) Because VSOCK's internal implementation is undocumented (i.e., through shared memory abstractions or network interfaces), we tested the effects of NIC speeds on VSOCK.As we see in our experimental results in Figure 6, switching from the c6i.8xlarge instance with a 12.5 Gbit/s network interface to the c6in.8xlarge instance with 50 Gbit/s slightly decreased the average throughput instead of increasing it. This suggests that VSOCK is not implemented using network interfaces. Consequently, a DBMS running in a Nitro Enclave will likely not benefit from network-optimized instances if VSOCK remains the bottleneck.

Performance insight. Overall, the achievable VSOCK throughput of the tested instances lies between 10 and 15 Gbit/s on average. This is slightly faster than the minimum network connection throughput in EC2 of 10 Gbit/s, yet an order of magnitude below modern network hardware capable of 200 Gbits/s. As a result, when a DBMS runs inside a Nitro Enclave, the VSOCK interface can easily become the limiting factor: large or network-optimized instances can deliver more data to the host than can be transferred into the enclave, wasting available CPU and memory.

Varying message sizes. In addition to the effect of hardware configurations on the VSOCK throughput, we investigate the impact of message size. Message size is a relevant factor for DBMS design: OLTP DBMS generally communicate in small messages but could try to batch messages for higher throughput at the cost of latency. OLAP DBMS can increase their data access granularity to optimize throughput. Message size can be varied in iperf3. For this experiment, we use four communication threads and channels, vary the message size from 64 B to 1 MB, and split a c6in.8xlarge instance into a host instance and an enclave. We run the experiment between two EC2 instances via the local network as a baseline. The results are depicted in Figure 7. The VSOCK throughput increases with message size. It reaches an optimum at approximately 32 kB message size. The observed pattern is similar to the TCP/IP baseline, although the absolute throughput is higher via the network. Thus, to achieve high throughput, OLAP DBMS will need to adjust message sizes accordingly.

Varying number of enclaves. Since larger enclaves and instance types with more CPU cores do not increase the VSOCK throughput, we evaluate if the combined VSOCK throughput of multiple smaller Nitro Enclaves on a host instance is higher than the throughput of a single large enclave. The AWS Nitro System allows one host instance to create up to four enclaves [6]. Thus, we scale the number of enclaves created concurrently and connect to each enclave with one client. We use four communication threads per enclave and execute the experiment on a c6i.4xlarge instance split into a host instance with 8 cores and four enclaves with two CPU cores each.

Figure 8 shows the aggregated throughput of all enclaves split by communication direction and number of enclaves. As we can see, throughput into the enclaves increases until three concurrently running enclaves. The throughput out of the enclaves increases from one to two enclaves. Thus, if the throughput available via one VSOCK interface limited a DBMS' performance, one solution could be to scale the DBMS to multiple Nitro Enclaves connected to the same host instance on the same hardware. However, this increases the asymmetry between ingress and egress that already exists for a single enclave even further, which might be problematic for communication between the enclaves and OLAP queries with large result sizes. We also tried to run multiple iperf servers in one enclave, but this did not increase throughput.

Latency of VSOCK. Next, we look at the latency of VSOCK communication, which is especially relevant for OLTP databases like Redis. To measure the latency, we wrote a VSOCK client-server application in C++ that allows us to measure round-trip latency. Since the server supports VSOCK, a proxy inside the enclave is unnecessary. All proxies that are not required when compared to the Redis setting are marked with hatches and dashed outlines in Figure 4. We varied the message sizes from 64 B to 1 MB and compare the two settings depicted in Figure 4 with the following three baselines: (1) TCP/IP local loopback round trip (LL) (2) TCP/IP local loopback including a proxy (LLP), and (3) TCP/IP round trip for communication in the same availability zone (N).

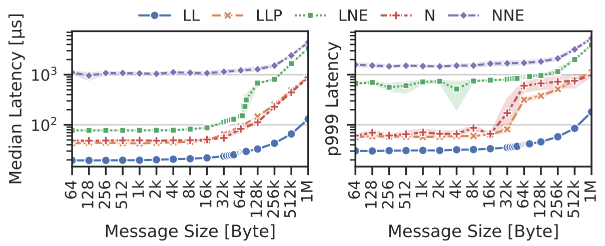

Figure 9: Median and p999 round trip latency of the settings introduced in Figure 4. VSOCK is slower than the intra-AZ network. The proxy introduces an order of magnitude higher latency for small messages. Tail latencies (p999) for VSOCK are significantly higher than for the other settings.

The results are depicted in Figure 9. They show that VSOCK round trip latency for small messages is approximately 80 µs. This is 3 times higher than local loopback latency (LL) and approximately 50 % higher than network latency within an AWS availability zone (N). The results also show that the proxy has a very high influence on latency. The latency in the network setting including a proxy (NNE) is more than 10 times higher than both the network baseline's latency without proxy and Nitro Enclave (N) and the non-network baseline proxying only TCP traffic (LLP). This indicates that additional latency is introduced by translating between TCP and VSOCK. Comparing the median latencies on the left with the 99.9th percentile on the right shows that the tail latencies of the VSOCK connection are significantly higher than for other means of communication.

Performance insight. From this experiment, we conclude that the proxy latency is crucial for latency-critical applications like OLTP DBMSs. A more detailed investigation on optimally implementing TCP to VSOCK proxies is thus an important area of future work.

Different CPU architectures. Finally, we investigate the influence of CPU generations and architectures on the throughput achievable via the VSOCK interface. We compare instances with the same number of CPU cores from Intel, AMD, and AWS (Graviton), as well as three generations of Intel-based EC2 instances. Each experiment uses 2xlarge instances with eight hardware threads, split evenly between the host and enclave. iperf3 is configured to use four threads. The results are depicted in Figure 10.

We draw the following three conclusions: (1) VSOCK throughput varies with CPU type (Figure 10, left). We measured the highest VSOCK throughput on AMD CPUs, followed by Intel CPUs. Throughput on Graviton-based instances is nearly an order of magnitude lower than on the x86_64 instances. Hence, AMD CPUs appear most suitable for throughput-sensitive DBMSs in Nitro Enclaves. (2) When comparing three generations of Intel-based AWS instances regarding their VSOCK throughput (Figure 10, right), we can observe that throughput decreases with newer-generation CPUs. We speculate that this is caused by CPU frequencies decreasing from 3 to 2 GHz over these generations. (3) Even for the faster AMD CPUs, VSOCK throughput to a single Nitro Enclave is limited to approx. 25 Gbit/s, eight times less than the fastest network connections currently available in AWS.

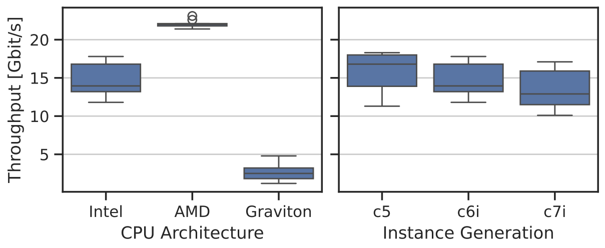

Figure 10: Comparison of host to enclave VSOCK throughput between CPU architectures and instance generations. The AMD-based instance has the fastest VSOCK, followed by the Intel and the Graviton-based instances. VSOCK throughput decreases with newer-generation processors.

4.3 Summary

The Redis benchmark shows that the VSOCK interface, combined with the required proxies, significantly degrades performance. Our micro-benchmarks indicate that much of this overhead stems from the enclave interface. VSOCK throughput is limited and ranges from 3 GB/s to 25 GB/s, depending on number of connections, CPU architecture, and communication direction. The round-trip latency into and out of the enclave via VSOCK is approximately 80 µs, which is similar to the latency between two VMs in an AWS availability zone. Latency is increased 10x by using socat to connect TCP and VSOCK streams.

5 Practical Lessons & Discussion

This section summarizes our insights into non-performance-related issues of working with AWS Nitro Enclaves and discusses the overall viability of AWS Nitro Enclaves for DBMS.

5.1 Usability & Limitations

Beyond benchmark results and performance numbers, we gained several qualitative insights from working with AWS Nitro Enclaves that have practical implications for database engineers.

For example, we found the debugging mode of the AWS Nitro CLI, which enables reading the standard output stream of the Nitro Enclave and the applications running inside it, useful for tracking down issues and bugs. However, the AWS Nitro Enclaves tooling comes with several restrictions that lead to compatibility issues. In particular, the current tooling only supports outdated kernel and software versions. At the time of writing, the official Nitro CLI still builds all enclave images with Linux kernel version 4.14. This outdated kernel misses several new features like io_uring for asynchronous IO, as well as performance optimizations, such as improvements to the scheduler and VSOCK. Even worse, because of the old kernel version, we experienced incompatibilities with current libc versions, which necessitated a switch to older libraries and compilers. These compatibility issues can quickly lead to higher overhead than expected for shifting applications into Nitro Enclaves. We think that improvements and updates to the tooling are overdue if AWS wants to position AWS Nitro Enclaves as a practical and usable TEE technology with low development overhead.

A major restriction we noticed with respect to the VSOCK interface is that VSOCK in Nitro Enclaves currently only supports streaming sockets similar to TCP. Although the specification supports datagram sockets – which could potentially improve performance [18] – they are not yet implemented in Nitro Enclaves. This omission is particularly unfortunate for DBMS workloads that, e.g., use UDP-based protocols to optimize data shuffling.

5.2 Architectural Implications of VSOCK

Although the standard libraries of important programming languages, such as C, Python, and Go, contain support for VSOCK, many applications and DBMSs in particular are not built with support for this type of communication. Thus, most DBMSs require a proxy that forwards TCP/UDP from the host instance to the enclave via the VSOCK interface. Alternatively, they require a rewrite to split them into a trusted processing component inside the enclave and an untrusted network/storage access component outside the enclave that communicate via VSOCK. Some general-purpose network proxy implementations already exist, such as socat, which can proxy TCP, and Nitriding [31], which can proxy HTTPS requests. However, as our benchmarks above and the Nitriding paper show, both tools have subpar performance for data-heavy systems. Thus, DBMS developers who want to use the Nitro Enclave features while maintaining their DBMS performance must build better proxies or change the DBMS architecture. This will cause significant engineering costs. If the VSOCK throughput is a bottleneck for a VSOCK-adapted DBMS, distributing the DBMS over multiple enclaves could increase performance (see Figure 8), but requires further architectural changes. Finally, the area of storage proxies is currently not explored in the scientific literature and might come with additional challenges.

5.3 Is Nitro interesting for DBMS?

Our experimental results and practical lessons raise the question whether AWS Nitro Enclaves are an interesting and useful technology for further research and development of secure DBMS when compared to hardware-based TEEs. We discuss three important aspects to this question: the trust model, the performance, and the future prospects and developments.

Trust model. As introduced in Section 2, the trust model and security assumptions of AWS Nitro Enclaves are different from those of SGX, SEV, and similar TEEs. In contrast to hardware-based TEEs that promise isolation from the cloud provider, AWS Nitro Enclaves use the cloud provider as their trust anchor. Thus, AWS Nitro Enclaves can be a fitting security technology if AWS is a common trusted party for users and service providers, and proof that the software is running in an AWS data center is required. Such a model might be beneficial for Database-as-a-Service providers.

Performance. As our experiments have shown, the processing performance inside AWS Nitro Enclaves is equal to normal VMs on the same hardware. However, the restriction to communication via VSOCK and reliance on proxies to enable communication with legacy DBMSs introduce severe performance reductions. As database systems heavily rely on I/O (either for communication with clients or for persistence), AWS Nitro Enclaves will currently not be a good choice for performance-critical database use cases.

Future developments. Reducing these performance bottlenecks is an interesting research and engineering challenge. However, we think that hardware-based TEEs currently have three important upsides compared to AWS Nitro Enclaves: (1) their security model is more general, (2) their design prevents cloud vendor lock-in, and (3) their design allows for more flexibility and performance optimizations in the systems built on top. For example, with the developments in multi-level confidential VMs and trusted I/O for confidential VMs [1, 19, 20], there are clear next steps to further minimize the TCB size and improve the performance of hardware-based TEEs.

6 Related Work

To the best of our knowledge, we are the first to investigate the performance implications of AWS Nitro Enclaves in detail, particularly for database workloads. Multiple research prototypes and commercial products for AWS Nitro Enclaves runtimes exist [7, 17, 31, 32]. These runtimes are meant to make executing arbitrary applications inside Nitro Enclaves easy by automating the setup and configuration of network and storage access proxies. Some of these runtimes come with their own proxies [7, 17, 31, 32]. In terms of pure proxy implementations, there is socat, which we used for our experiments, the AWS VSOCK proxy [4], that only enables applications inside enclaves to connect to pre-determined network addresses, and parts of commercial products [24]. To the best of our knowledge, no work currently compares the performance of different proxies.

In contrast to AWS Nitro Enclaves, there is a lot of research investigating the performance characteristics of other TEEs and their usability for database workloads, such as Intel SGX [8, 16, 21–23], Intel TDX [13, 33], and AMD SEV [27, 34].

7 Conclusions & Future Directions

This paper analyzes the viability of AWS Nitro Enclaves for secure DBMS in the AWS cloud. Although Nitro Enclaves are VMs that allow in-memory data processing without overheads, changes to DBMS architecture are required to deal with the limiting enclave interface. The VSOCK interface necessitates a proxy on the host instance that translates normal network traffic and storage accesses. Additionally, the VSOCK interface limits the achievable network and storage throughput and increases latency. Future work should investigate performance optimizations through newer kernel and hypervisor versions, the development of high-performance network and storage proxies, and the integration of VSOCK support into the DBMS architecture.

Acknowledgments

This work was supported by SAP SE, the grant "High-Performance Analytical & Secure Query Processing for Cloud Databases (HiPeAC)" by the National Research Center for Applied Cybersecurity ATHENE, and the LOEWE Spitzenprofessur of the state of Hesse. We also thank hessian.AI and DFKI for their support.

References

[1] Advanced Micro Devices, Inc. 2020. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/solution-briefs/amd-secure-encrypted-virtualization-solution-brief.pdf

[2] Amazon Web Services. 2024. The Security Design of the AWS Nitro System. https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.html

[3] Amazon Web Services. 2025. Aws-Nitro-Enclaves-Cli/Vsock_proxy/README.Md. https://github.com/aws/aws-nitro-enclaves-cli/blob/main/vsock_proxy/README.md

[4] Amazon Web Services. 2025. Aws/Aws-Nitro-Enclaves-Cli. Amazon Web Services. https://github.com/aws/aws-nitro-enclaves-cli

[5] Amazon Web Services. 2025. What Is Nitro Enclaves? - AWS. https://docs.aws.amazon.com/enclaves/latest/user/nitro-enclave.html

[6] Amazon Web Services. 2025. Working with Multiple Enclaves - AWS. https://docs.aws.amazon.com/enclaves/latest/user/multiple-enclaves.html

[7] Anjuna Security, Inc. 2025. Anjuna Seaglass | Universal Confidential Computing. https://www.anjuna.io/product/seaglass

[8] Ilaria Battiston, Lotte Felius, Sam Ansmink, Laurens Kuiper, and Peter Boncz. 2024. DuckDB-SGX2: The Good, The Bad and The Ugly within Confidential Analytical Query Processing. In *Proceedings of the 20th International Workshop on Data Management on New Hardware (DaMoN '24)*. Association for Computing Machinery, New York, NY, USA, 1–5. doi:10.1145/3662010.3663447

[9] Marion Bonnet. 2023. Cloud Assets the Biggest Targets for Cyberattacks, as Data Breaches Increase | Thales Group. https://www.thalesgroup.com/en/worldwide/security/press_release/cloud-assets-biggest-targets-cyberattacks-data-breaches-increase

[10] David Bronleewe, Hormuzd Khosravi, Shanmathi Rajasekar, Shiny Sebastian, and Raghuram Yeluri. 2025. Runtime Encryption of Memory with Intel® Total Memory Encryption–Multi-Key (Intel® TME-MK). https://www.intel.com/content/www/us/en/developer/articles/news/runtime-encryption-of-memory-with-intel-tme-mk.html

[11] David Brown. 2021. Confidential Computing: An AWS Perspective | AWS Security Blog. https://aws.amazon.com/blogs/security/confidential-computing-an-aws-perspective/

[12] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel TDX Demystified: A Top-Down Approach. *ACM Comput. Surv.* 56, 9 (April 2024), 238:1–238:33. doi:10.1145/3652597

[13] Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, and Luigi Romano. 2025. An Experimental Evaluation of TEE Technology: Benchmarking Transparent Approaches Based on SGX, SEV, and TDX. *Computers & Security* 154 (July 2025), 104457. doi:10.1016/j.cose.2025.104457

[14] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. https://eprint.iacr.org/2016/086.pdf

[15] dest-unreach.org. 2025. Socat. http://www.dest-unreach.org/socat/

[16] Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig. 2022. Benchmarking the Second Generation of Intel SGX Hardware. In *Data Management on New Hardware (DaMoN'22)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/3533737.3535098

[17] Evervault Inc. 2025. Evervault Enclaves. https://evervault.com/products/enclaves

[18] Amery Hung and Bobby Eshleman. 2023. VSOCK: From Convenience to Performant VirtIO Communication. https://lpc.events/event/17/contributions/1626/attachments/1334/2674/VSOCK_%20From%20Convenience%20to%20Performant%20VirtIO%20Communication.pdf

[19] Intel Corporation. 2023. Intel® TDX Connect Architecture Specification. https://cdrdv2.intel.com/v1/dl/getContent/773614

[20] Intel Corporation. 2024. Intel® Trust Domain Extensions (Intel® TDX) Module TD Partitioning Architecture Specification. https://cdrdv2.intel.com/v1/dl/getContent/773039

[21] Adrian Lutsch, Muhammad El-Hindi, Matthias Heinrich, Daniel Ritter, Zsolt István, and Carsten Binnig. 2025. Benchmarking Analytical Query Processing in Intel SGXv2. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, Konstanz, Germany, 516–528. doi:10.48786/EDBT.2025.41

[22] Adrian Lutsch, Muhammad El-Hindi, Zsolt István, and Carsten Binnig. 2025. Towards High-performance and Trusted Cloud DBMSs. *Datenbank-Spektrum* 25, 1 (March 2025), 39–50. doi:10.1007/s13222-025-00495-8

[23] Kajetan Maliszewski, Jorge-Arnulfo Quiané-Ruiz, Jonas Traub, and Volker Markl. 2021. What Is the Price for Joining Securely? Benchmarking Equi-Joins in Trusted Execution Environments. *Proceedings of the VLDB Endowment* 15, 3 (Nov. 2021), 659–672. doi:10.14778/3494124.3494146

[24] Marlin Foundation. 2025. Oyster-Monorepo/Networking. https://github.com/marlinprotocol/oyster-monorepo/tree/master/networking

[25] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. 2018. A Comparison Study of Intel SGX and AMD Memory Encryption Technology. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP '18)*. Association for Computing Machinery, New York, NY, USA, 1–8. doi:10.1145/3214292.3214301

[26] Siani Pearson and Azzedine Benameur. 2010. Privacy, Security and Trust Issues Arising from Cloud Computing. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. 693–702. doi:10.1109/CloudCom.2010.66

[27] Lina Qiu, Rebecca Taft, Alexander Shraer, and George Kollios. 2024. The Price of Privacy: A Performance Study of Confidential Virtual Machines for Database Systems. In *Proceedings of the 20th International Workshop on Data Management on New Hardware*. ACM, Santiago AA Chile, 1–8. doi:10.1145/3662010.3663440

[28] Redis Ltd. 2025. Redis Benchmark. https://redis.io/docs/latest/operate/oss_and_stack/management/optimization/benchmarks/

[29] Various Authors. 2025. iPerf - The TCP, UDP and SCTP Network Bandwidth Measurement Tool. https://iperf.fr/

[30] Zack Whittaker. 2023. Danish Cloud Host Says Customers 'lost All Data' after Ransomware Attack. https://techcrunch.com/2023/08/23/cloudnordic-azero-cloud-host-ransomware/

[31] Philipp Winter, Ralph Giles, Moritz Schafhuber, and Hamed Haddadi. 2023. Nitriding: A Tool Kit for Building Scalable, Networked, Secure Enclaves. arXiv:2206.04123 [cs] http://arxiv.org/abs/2206.04123

[32] Eugene Yakubovich. 2022. Introducing Enclaver: An Open-Source Tool for Building, Testing and Running Code within Secure Enclaves. https://edgebit.io/blog/enclaver/

[33] Xinying Yang, Cong Yue, Wenhui Zhang, Yang Liu, B. Ooi, and Jianjun Chen. 2024. SecuDB: An In-enclave Privacy-preserving and Tamper-resistant Relational Database. https://www.semanticscholar.org/paper/SecuDB%3A-An-In-enclave-Privacy-preserving-and-Yang-Yue/bf95cb32d222a794a0b2beb29402550e3cb42640?utm_source=alert_email&utm_content=LibraryFolder&utm_campaign=AlertEmails_WEEKLY&utm_term=LibraryFolder&email_index=9-0-15&utm_medium=39105601

[34] Junseung You, Kyeongryong Lee, Hyungon Moon, Yeongpil Cho, and Yunheung Paek. 2023. KVSEV: A Secure In-Memory Key-Value Store with Secure Encrypted Virtualization. In *Proceedings of the 2023 ACM Symposium on Cloud Computing (SoCC '23)*. Association for Computing Machinery, New York, NY, USA, 233–248. doi:10.1145/3620678.3624658

[35] Xuyang Zhao, Mingyu Li, Erhu Feng, and Yubin Xia. 2022. Towards A Secure Joint Cloud With Confidential Computing. In *2022 IEEE International Conference on Joint Cloud Computing (JCC)*. IEEE, Fremont, CA, USA, 79–88. doi:10.1109/JCC56315.2022.00019