Journal of Information Systems Engineering and Management
2025, 10(12s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Comparative Review of AWS and Azure Confidential Computing Systems

Nikhil Sagar Miriyala[1*], Kiran Babu Macha[2], Shubham Metha[3], Dyuti Dave[4]
[1]*Senior Software Engineer, Oracle America Inc., USA*
[2]*Sr. Manager – Software Engineering, Maximus Inc., USA*
[3]*Software Engineer II, Northwest Bank, USA*
[4]*Technology Analyst, Barclays, USA*
**Corresponding Author: nmiriya7@gmail.com*

ARTICLE INFO

Received: 18 Nov 2024

Revised: 30 Dec 2024

Accepted: 18 Jan 2025

ABSTRACT

Confidential computing is an emerging field that focuses on ensuring the privacy and security of data during computation, addressing the increasing concerns with respect to data breaches and unauthorized access to sensitive information in cloud environments. The paper provides a comparative review of two prominent cloud platforms offering confidential computing solutions, namely, AWS Nitro Enclaves and Azure Confidential VMs with Intel SGX Enclaves. We analyse both platforms across multiple dimensions, such as their core components, attestation processes, communication mechanisms, and performance limitations. Additionally, we will evaluate practical factors such as regional availability, types of instances, and resource allocations. By analysing the unique features, advantages, and trade-offs of each platform, the paper aims to provide a clear understanding of why confidential computing is important in the context of increasing data privacy requirements and regulatory frameworks, such as GDPR and how these technologies enable secure and isolated execution of sensitive workloads on the cloud.

Keywords: AWS, Microsoft Azure, Confidential Computing, Nitro Enclaves, Intel SGX, etc

INTRODUCTION

For many years, the primary focus in the data security industry has been on protecting data at rest and data in transit. These two areas were addressed using a variety of encryption and security technologies, but as data processing environments become more complex and distributed, a new challenge emerged: protecting data while it is actively being processed. Let's first look at how traditional methods addressed the security of data at rest and in transit.

Protecting Data at Rest

Data at rest pertains to information that is held on physical storage devices, such as hard drives, databases, or cloud storage solutions. It is essential to secure this information because an attacker who gains physical access to the storage medium should not be able to read or exploit the data. To safeguard data at rest, encryption has traditionally been the favoured approach. Encryption guarantees that even if an unauthorized individual accesses the storage, the information remains inaccessible without the matching decryption keys. Below are some of the prevalent technologies employed to protect data at rest [1]:

1) AES (Advanced Encryption Standard): This is a widely used encryption protocol designed to safeguard data within databases, file systems, and cloud platforms. AES guarantees data security by utilizing a robust cryptographic algorithm for encryption.
2) Key Management Systems (KMS): These systems are responsible for overseeing the management and access control of encryption keys. KMS ensures that the keys are securely stored, rotated frequently, and only available to authorized users or systems, thereby minimizing the risk of key theft or unauthorized data decryption.

Protecting Data at Transit

Data in transit refers to information that is being sent across a network, whether within a local network, through the internet, or between various cloud services. While data is being transmitted, it faces risks of interception and manipulation. Therefore, safeguarding data in transit is essential to prevent sensitive information from being disclosed to unauthorized individuals during transfer. To secure data in transit, encryption methodologies are utilized. These methods protect the data as it travels between systems, ensuring its confidentiality and integrity. Below are some of the common technologies utilized for safeguarding data during transit [2]:

1) TLS (Transport Layer Security): A protocol commonly used to encrypt data transferred between web browsers and servers. TLS ensures secure communication over the Internet, preventing data from being intercepted and read by malicious actors.
2) VPNs (Virtual Private Networks) and IPsec (Internet Protocol Security): These are used to secure private communications over public networks. They provide encryption and authentication, making sure that the data exchanged over the network remains confidential and unaltered.

The Need for Protecting Data in Use

As organizations continue to migrate their operations to the cloud, new security challenges are emerging, and **Figure 1** shows some of the high-level challenges in handling large amounts of data on the cloud. Most of these security challenges revolve around data-in-use, which refers to data that are actively processed by applications or services. With the increasing adoption of cloud computing, data is often handled on shared infrastructure that lies outside the organization's direct control. This increase in cloud traffic is thus increasing the concerns with respect to data leakage, unauthorized access, and the potential misuse of sensitive information during processing [3].

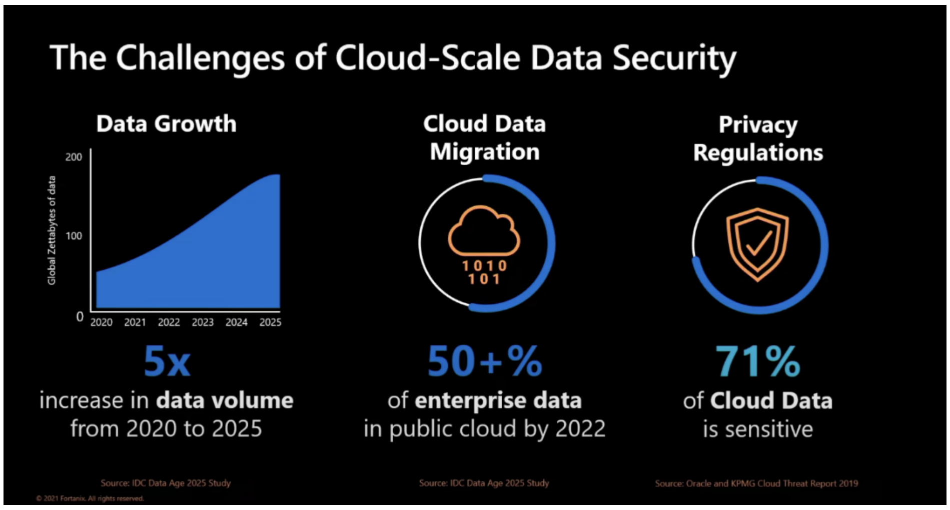


Figure 1. Challenges for Data Security on Cloud [4]

Data Protection Regulation (GDPR) [5] have further emphasized the importance of protecting personal data throughout its lifecycle. GDPR and similar regulations require businesses to safeguard not only the storage and transmission of sensitive data but also the processing of this data. For example, under GDPR, businesses must ensure that data is processed securely and that personal data is not exposed to unauthorized parties during computation. This has led to the growing demand for solutions that can protect data during its processing — a need that traditional security models do not fully address.

Confidential Computing: A Solution to Protect Data in Use

In response to this need, confidential computing [6] has emerged as a critical technology. Confidential computing protects data during its processing by leveraging specialized hardware and trusted execution environments (TEEs) to create isolated environments for executing sensitive computations. Below are some of the major underlying principles that define how confidential computing can protect data in use [7].

1) Isolation: Using TEEs, confidential computing ensures that data is isolated from all other processes, including the host OS, hypervisor, and the cloud provider. The only code that can access the data within the enclave is trusted and verified code running inside the enclave.
2) End-to-End Encryption: Data remains encrypted throughout its lifecycle - while being stored, transmitted, and processed. The encryption keys are securely managed within the enclave, and data cannot be accessed or modified during computation.
3) Attestation: Attestation mechanisms ensure that only verified and trusted code is running within the enclave. This adds a layer of assurance that the enclave has not been tampered with and is operating securely.

Amazon Web Services (AWS) and Microsoft Azure, two of the major public cloud providers in the industry, now have support for confidential virtual machines as part of their infrastructure suite. While AWS has designed and developed their own mechanism through which they provide confidential VMs and enclaves support [8], Microsoft Azure confidential VMs are built on top of AMD SEV-SNP [9] and Intel SGX [10] technologies. In this paper, we will review key differences between AWS Nitro Enclaves and Intel SGX-based Azure Confidential VMs with respect to their architecture, attestation processes, and other key areas.

CORE COMPONENTS

AWS Nitro Enclaves

AWS Nitro Enclaves are built on top of the AWS Nitro System, a hardware-accelerated virtualization platform. The Nitro Enclaves architecture is specifically designed to provide secure, isolated environments within EC2 instances. Below are the key components of AWS Nitro Enclaves.

1) AWS Nitro Hypervisor:
   a) The Nitro Hypervisor is a lightweight, high-performance hypervisor that manages EC2 instances and the allocation of resources between instances. It is responsible for managing memory, CPU, and I/O resources.
   b) It also enables resource isolation between the parent EC2 instance and the Nitro Enclave, ensuring that the enclave's resources are allocated securely and isolated from the instance.
2) Nitro Security Chip:
   a) The Nitro Security Chip is a dedicated hardware module embedded in the physical server. It is a core part of the AWS Nitro System, responsible for providing secure boot processes and enforcing security policies.
   b) It ensures the root of trust by cryptographically verifying that the EC2 instance and any associated Nitro Enclaves are running the expected, untampered software.
   c) The Nitro Security Chip also handles attestation, ensuring that the Nitro Enclave is running trusted software and generating cryptographic proofs to external services.
3) Enclave Controller:
   a) The Enclave Controller is a software component that runs inside the EC2 instance and manages the lifecycle of the enclave. It coordinates the creation, termination, and secure communication of enclaves within the parent EC2 instance.
   b) The Enclave Controller also helps with resource allocation and management, ensuring that the enclave runs with the appropriate amount of CPU and memory resources.
4) Nitro Enclave:
   a) A Nitro Enclave is an isolated, secure environment that is created from a portion of the EC2 instance's CPU and memory resources. It is designed to run sensitive workloads without exposing them to the parent instance, the host, or AWS administrators.
   b) The enclave has no network access, no persistent storage, and no root access, ensuring that data inside the enclave is secure from external threats and unauthorized access.
   c) The enclave communicates with the parent EC2 instance through a secure, encrypted channel for data exchange. This communication is strictly controlled and isolated to maintain confidentiality.

Azure Intel-SGX Enclaves

Azure Confidential Computing uses Intel SGX (Software Guard Extensions) to provide secure enclaves, allowing sensitive workloads to be processed securely without exposing the data to unauthorized access, even by Azure administrators. The architecture of Azure Confidential Computing VMs integrates SGX technology to create these secure enclaves. Below are the key components of Intel SGX Enclaves:

1) Intel SGX (Software Guard Extensions):
    a) SGX is a set of hardware-based security features integrated into Intel processors. It allows the creation of secure enclaves in memory that are isolated from the rest of the system, including the OS, hypervisor, and other applications.
    b) SGX enclaves provide memory encryption, ensuring that the data inside the enclave is protected while it is being processed.
    c) SGX also offers remote attestation, allowing the enclave to prove its identity and integrity to external parties before they exchange sensitive data.
2) Azure Confidential VM:
    a) An Azure Confidential VM is a virtual machine running on Intel SGX-enabled hardware. The VM is configured to support the execution of SGX enclaves and provides a secure environment for workloads that require data protection during computation.
    b) The Azure hypervisor provides isolation between different VMs running on the same physical host, and the SGX hardware ensures that data within enclaves is protected.
    c) The Azure Confidential VM also runs SGX-enabled software that allows users to create and manage SGX enclaves within the VM for sensitive computations.
3) Attestation Service:
    a) Azure uses remote attestation to verify the integrity of the SGX enclave. The enclave can attest to a trusted attestation service, such as Intel’s Attestation Service, to prove that the software running inside the enclave is legitimate and hasn’t been tampered with.
    b) This process involves generating an attestation report that includes the hash of the enclave image, the public key of the enclave, and other relevant data to validate the enclave's authenticity.
4) SGX Enclave Runtime:
    a) The SGX Enclave Runtime is the software environment that manages the lifecycle of SGX enclaves. It provides essential functionality such as enclave creation, memory allocation, and managing communication between the enclave and the host.
    b) The runtime ensures that the enclave operates securely within its allocated memory and provides interfaces for developers to interact with the enclave for secure computations.
5) SGX Enclave:
    a) An SGX Enclave is a secure region of memory where sensitive data is processed in isolation from the host OS and other applications. The enclave can execute trusted code in a protected environment without exposing its contents.
    b) The enclave’s memory is encrypted, preventing access by the host OS or other malicious processes.
    c) SGX provides fine-grained isolation, allowing sensitive operations to be run within the enclave while maintaining confidentiality and integrity.

ATTESTATION PROCESS

As mentioned in earlier sections, attestation is a key principle in confidential computing that guarantees the enclave operating within a trusted execution environment (TEE) is genuine and has not been altered. Moreover, the attestation document that is created and presented to key management systems (like AWS KMS and Azure Key Vault) serves as evidence that both encryption and decryption tasks are carried out solely by applications functioning within the enclave. If any application outside the TEE tries to decrypt sensitive information, the operation will not succeed, as the KMS checks the attestation document to verify the enclave's legitimacy before permitting access to the decryption keys. Now, let’s examine how the attestation process operates in AWS Nitro Enclaves and Azure VMs utilizing SGX enclaves.

AWS Nitro Enclaves

Below are the steps involved in the AWS Nitro Enclave Attestation procedure [11]:

1) Attestation Request: Once a new enclave is created, it triggers an attestation document generation request, which invokes the AWS Nitro Hypervisor.

2) Generation of Attestation Document: Upon receiving the request, the nitro hypervisor generates a new document that contains all the information about the enclave, which includes the signing key, hash of the image, Platform configuration registers (PCRs), and the IAM role information attached to the EC2 instance.
3) Attestation Document Validation: Further, the attestation document generated is signed using the COSE_Sign1 structure, and it's verified using the AWS Nitro Attestation Public Key Infrastructure (PKI).
4) Use of Nonce: To ensure that the attestation document is not reused, AWS Nitro Hypervisor utilizes an optional nonce while generating the document and prevents the system from replay attacks.
5) Service Interaction: Any third-party service requesting the document establishes a TLS connection with the enclave, whose authenticity is verified by the enclave first using the public certificate of the service, and the enclave then sends the attestation document to the service.
6) Attestation Validation by Service: The party service decodes and verifies the attestation document and the associated signature upon receival. The certificate chain is further validated to ensure that the document was correctly signed by the Nitro PKI.
7) Final Verification: The service ensures that the Attestation Document has the correct structure and that required fields (such as module_id, digest, timestamp, and PCRs) are present and valid.

Azure Intel-SGX Enclaves

Below are the steps involved in an Azure VM with SGX Enclave Attestation procedure [12]:

1) Attestation Request: Once an SGX enclave has been created, the guest VM generates an attestation quote which contains all the information about the state of an enclave. This quote is requested from the SGX hardware.
2) Attestation Evidence Submission: Once the attestation quote is generated, it is then submitted to the Microsoft or Intel Attestation service for validation. The quote generated includes enclave's signing certificate along with few SGX-specific values such as MRSIGNER, MRENCLAVE, SVN, etc.
3) Quote Validation: The Attestation service verifies the SGX quote against the default Trusted Computing Base (TCB) or a custom TCB baseline if configured. This step ensures that the enclave's software has not been tampered with.
4) Generation of Attestation Token: Once the validation is successful, the Attestation service issues an attestation token, which acts as a certificate that confirms that the enclave is running on trusted hardware and has not been compromised. The attestation token is then signed and returned to the third-party service party that requested it.
5) Secure Communication: The attestation token can be used by the third-party service to create a safe connection for further communication, such as retrieving encryption keys or accessing sensitive information.
6) Policy Evaluation: Custom policies can be set up for additional control of access to sensitive information and the attestation token will be evaluated against these by the attestation service.
7) Service-Side Validation (Optional): If required, the third-party service can also validate the attestation token (similar to mTLS) to ensure that the token has been generated by a valid attestation service, which is done using the signature exposed via an OpenID metadata endpoint.
8) Final Verification: Once all the attestation and policy validation steps are complete, the third-party service can then finally trust the enclave and allow it to handle sensitive data or perform secure computations.

COMMUNICATION WITH THE ENCLAVE

Since applications running inside enclaves are isolated from network calls, and regular TCP and UDP communications are restricted to and from the enclave, it's essential to ensure that the application can still listen for incoming communication and perform downstream operations, as it would in a typical scenario. Achieving this requires some code modifications within the application, along with additional configuration. In this section, we will review how to accomplish this in both the AWS and Azure world.

AWS Nitro Enclaves

Unlike Azure Confidential VM's using Intel SGX Enclaves, where a piece of the host memory is isolated and encrypted, AWS Nitro Enclaves offers both CPU and memory isolation features and these enclaves are treated as a separate entity within the EC2 instance. Thus, not only can the host access enclave memory, but even the enclaves cannot access anything outside its space directly and all the communication is driven through secure local channels.

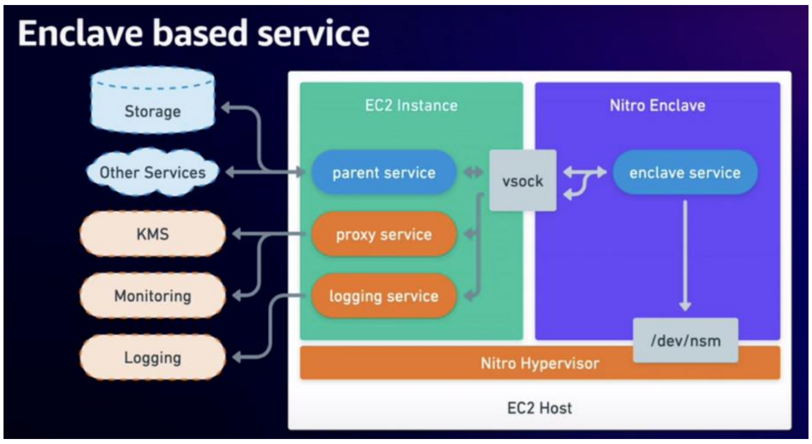
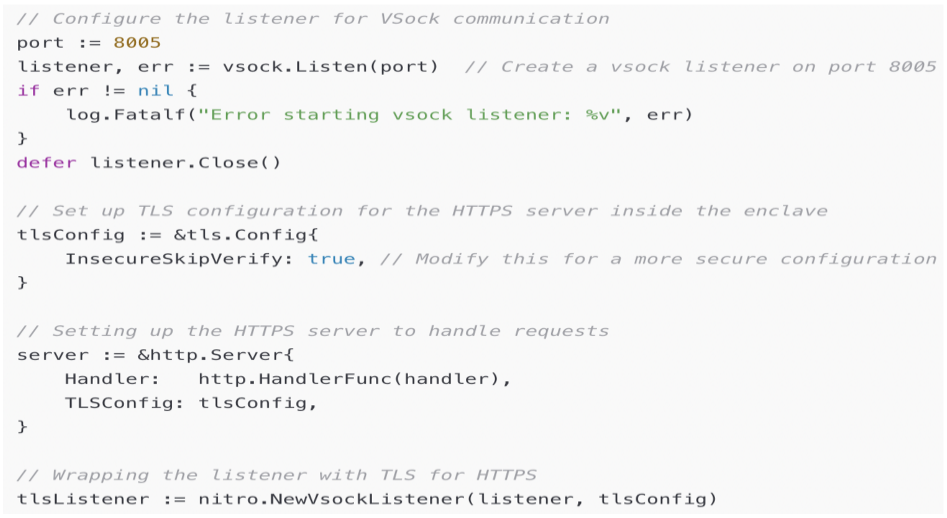


Figure 2. AWS Nitro Enclaves – Communication Model [14]

As seen in **Figure 2**, the core communication model of Nitro Enclaves is using secure local channels, called are Virtual Sockets (Vsocks) [13]. According to AWS [14], "Vsock is a type of socket interface that is defined by a context ID (CID) and port number. The context ID is parallel to an IP address in a TCP/IP connection. Vsock utilizes standard, well-defined POSIX Sockets APIs (e.g. connect, listen, accept) to communicate with an enclave. Applications can use these APIs to communicate natively over vsock, or they can send HTTP requests over vsock through a proxy."



```
// Configure the listener for VSock communication
port := 8005
listener, err := vsock.Listen(port)  // Create a vsock listener on port 8005
if err != nil {
    log.Fatalf("Error starting vsock listener: %v", err)
}
defer listener.Close()

// Set up TLS configuration for the HTTPS server inside the enclave
tlsConfig := &tls.Config{
    InsecureSkipVerify: true, // Modify this for a more secure configuration
}

// Setting up the HTTPS server to handle requests
server := &http.Server{
    Handler:    http.HandlerFunc(handler),
    TLSConfig: tlsConfig,
}

// Wrapping the listener with TLS for HTTPS
tlsListener := nitro.NewVsockListener(listener, tlsConfig)
```

Figure 3. AWS Nitro Enclaves – Code Changes in the Application

To explain it further, let us take an example of a sample HTTPS server-based application running inside enclaves. To ensure that the application can listen for Vsock based requests, instead of using a typical https server and listener function, we need to replace the https listener with a Vsock listener. **Figure 3** showcases how the sample code changes would look like. Once we have the application ready with all the necessary code changes, we need to build and dockerize the application and then nitro-cli tools to create an encrypted image file. **Figure 4** shows the entire build pipeline. Once we have made the necessary code changes, dockerized the application, we create an encrypted image file (EIF), that can be assigned with certain amount of vCPUs and memory and we can start the enclave application.

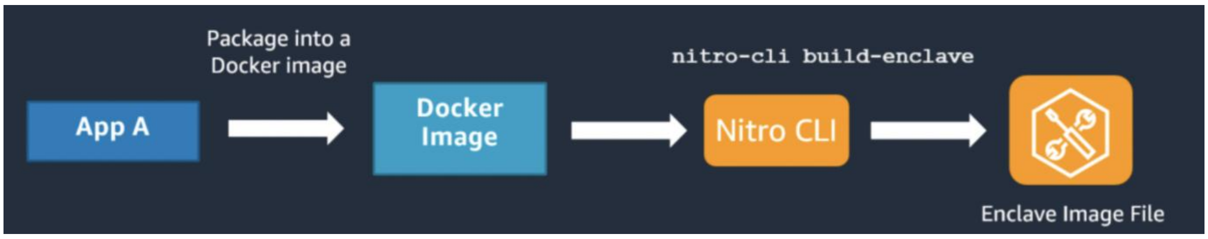
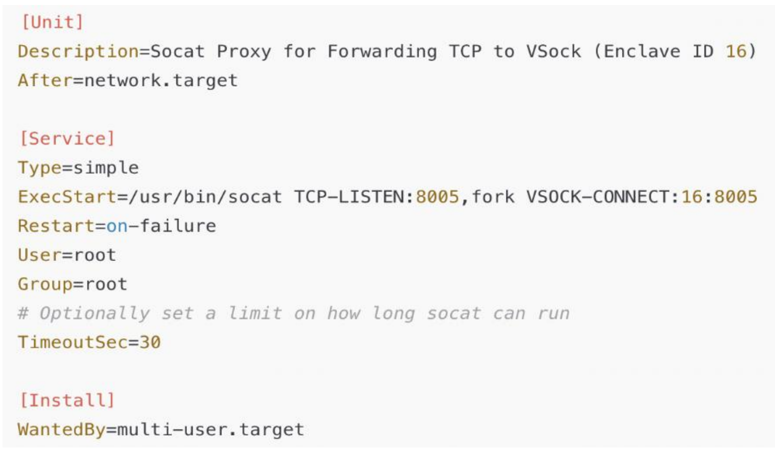


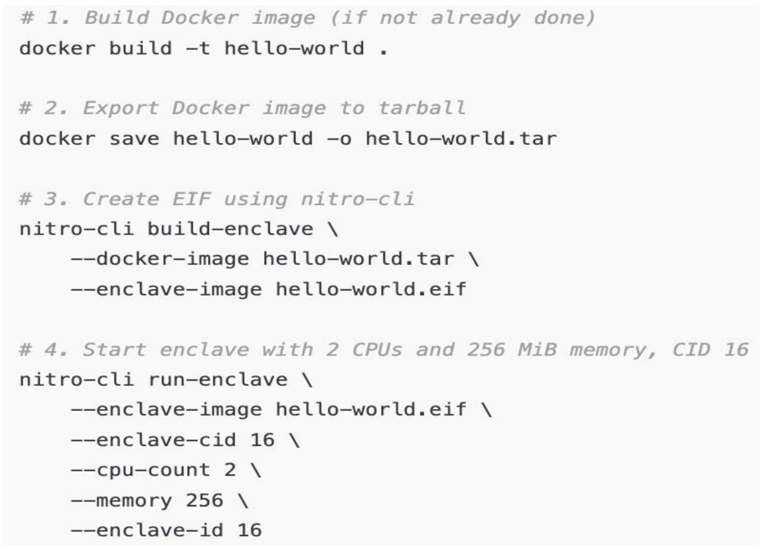
Figure 4. AWS Nitro Enclaves – Build Pipeline [14]



```
[Unit]
Description=Socat Proxy for Forwarding TCP to VSock (Enclave ID 16)
After=network.target

[Service]
Type=simple
ExecStart=/usr/bin/socat TCP-LISTEN:8005,fork VSOCK-CONNECT:16:8005
Restart=on-failure
User=root
Group=root
# Optionally set a limit on how long socat can run
TimeoutSec=30

[Install]
WantedBy=multi-user.target
```

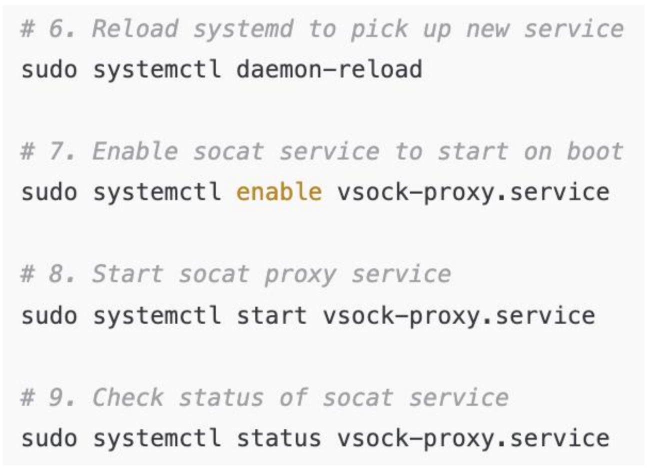
Figure 5. AWS Nitro Enclaves – Socat Proxy Configuration

The application within the enclaves can now listen and serve the incoming vsock traffic, however, clients using this API will continue to make HTTPS based API calls and TCP traffic needs to be converted to vSock traffic first. To achieve this, a socat proxy service can be used, and **Figure 5** shows how a sample configuration file would look like. Once we have the EIF file to run the application within the enclaves and the socat proxy configuration, we can use the list of steps specified in **Figure 6** and **Figure 7**, to start the proxy service on the host and the application inside an enclave. As seen in the image, we will allocate a certain amount of vCPUs and memory for each enclave spun up and assign an enclave id to it, which will be referred to, by the proxy services on the host, so that it knows which enclave to communicate with, since each EC2 instance can run multiple enclaves at once.



```
# 1. Build Docker image (if not already done)
docker build -t hello-world .

# 2. Export Docker image to tarball
docker save hello-world -o hello-world.tar

# 3. Create EIF using nitro-cli
nitro-cli build-enclave \
    --docker-image hello-world.tar \
    --enclave-image hello-world.eif

# 4. Start enclave with 2 CPUs and 256 MiB memory, CID 16
nitro-cli run-enclave \
    --enclave-image hello-world.eif \
    --enclave-cid 16 \
    --cpu-count 2 \
    --memory 256 \
    --enclave-id 16
```

Figure 6. AWS Nitro Enclaves – List of Commands to Start the Enclave



```
# 6. Reload systemd to pick up new service
sudo systemctl daemon-reload

# 7. Enable socat service to start on boot
sudo systemctl enable vsock-proxy.service

# 8. Start socat proxy service
sudo systemctl start vsock-proxy.service

# 9. Check status of socat service
sudo systemctl status vsock-proxy.service
```

Figure 7. AWS Nitro Enclaves – List of Commands to Start the Vsock Proxy

Azure Intel-SGX Enclaves

Microsoft Azure documentation provides multiple open-source libraries that can be used for developing applications utilizing enclave technologies, with one of them being, Open Enclave SDK, which is actively maintained by Microsoft Developer Community [15].

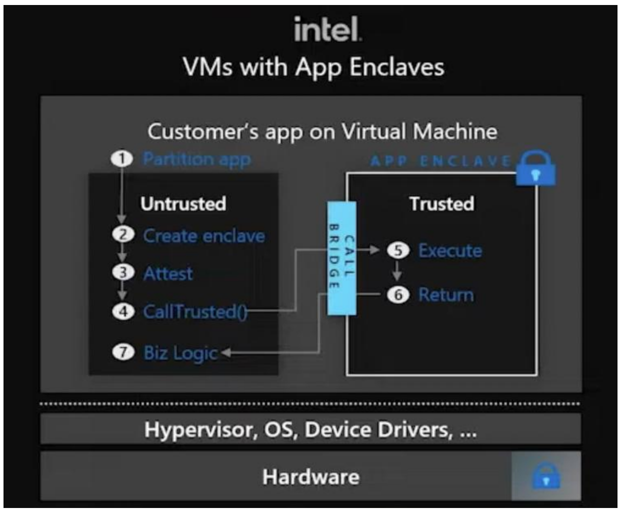
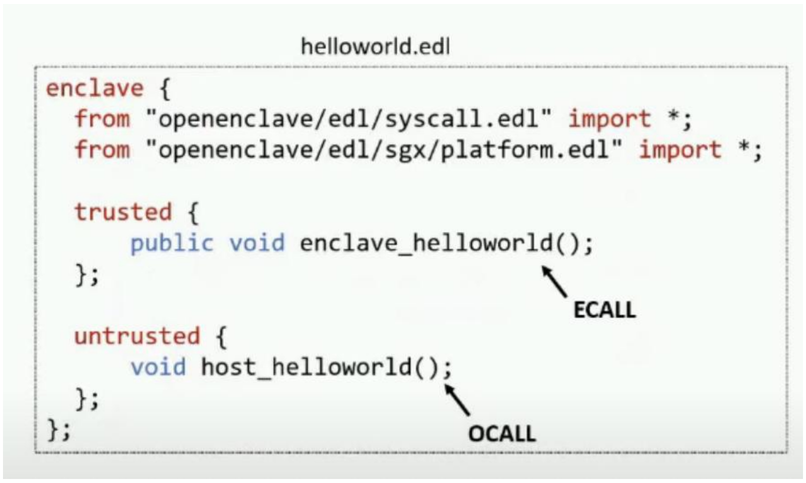


Figure 8. Azure Intel-SGX Enclaves – Communication Model [15]

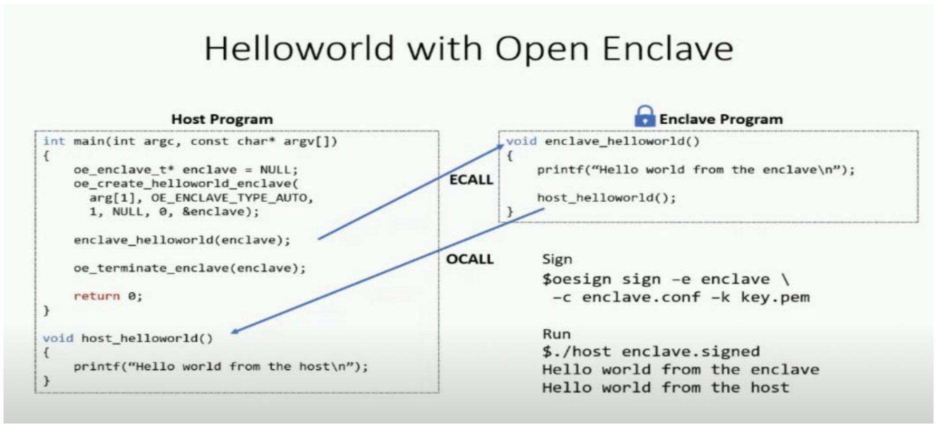


helloworld.edl

```
enclave {
  from "openenclave/edl/syscall.edl" import *;
  from "openenclave/edl/sgx/platform.edl" import *;

  trusted {
      public void enclave_helloworld();
  };

  untrusted {
      void host_helloworld();
  };
};
```

Figure 9. Azure Intel-SGX Enclaves – EDL File for a Hello World Function [18]

Intel SGX enclave functionalities can be invoked using additional OS level functions, namely ECALL and OCALL, which are used for communications to the enclave and from the enclave respectively [16], and **Figure 8** refers to the high-level communication flow when using these functions. Since the EPC (Enclave Page Cache) memory used by the enclave is encrypted and isolated, the host cannot access the EPC memory, but the enclave application can still access the host memory, and this is basically how data sharing and enclave invocations are done. Below are the high-level steps involved in communication between host and enclave using Open Enclave SDK, with a simple hello world example.


Helloworld with Open Enclave

Host Program

```
int main(int argc, const char* argv[])
{
    oe_enclave_t* enclave = NULL;
    oe_create_helloworld_enclave(
      arg[1], OE_ENCLAVE_TYPE_AUTO,
      1, NULL, 0, &enclave);

    enclave_helloworld(enclave);

    oe_terminate_enclave(enclave);

    return 0;
}
void host_helloworld()
{
    printf("Hello world from the host\n");
}
```

Enclave Program

```
void enclave_helloworld()
{
    printf("Hello world from the enclave\n");

    host_helloworld();
}
```

Sign

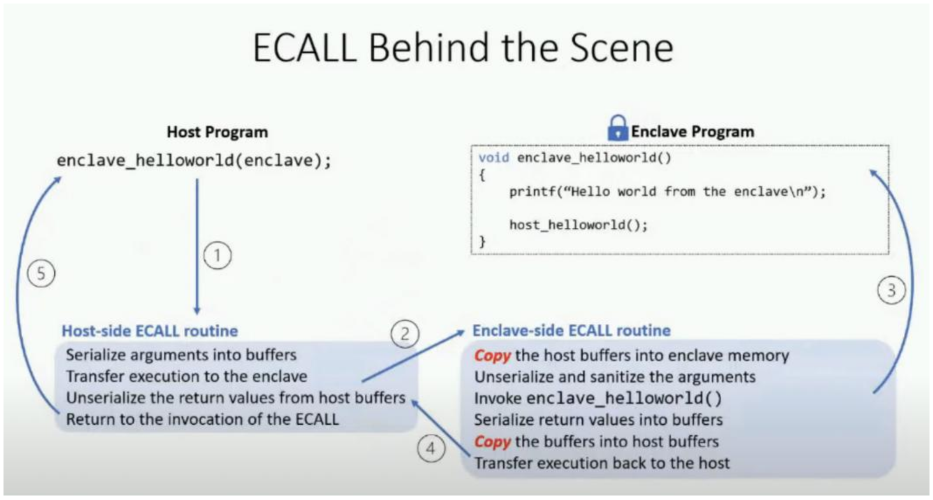
```
$oesign sign -e enclave \
 -c enclave.conf -k key.pem
```

Run

```
$./host enclave.signed
Hello world from the enclave
Hello world from the host
```

Figure 10. Azure Intel-SGX Enclaves – Application Code Changes [18]

When using the Open Enclave SDK, the interactions between host and enclave code are done using Enclave Definition Language (EDL) files. These EDL files define the enclave functions and allows the developer to specify the trusted and untrusted sections of the code. After defining the EDL file, let's say for a Hello World function as shown in **Figure 9**, the developer can use the tools provided by Open Enclave to generate the boundary C code as shown in **Figure 10**. This code abstracts low-level interactions, making the developer's experience easier and more secure [17].


ECALL Behind the Scene

Host Program

```
enclave_helloworld(enclave);
```

Enclave Program

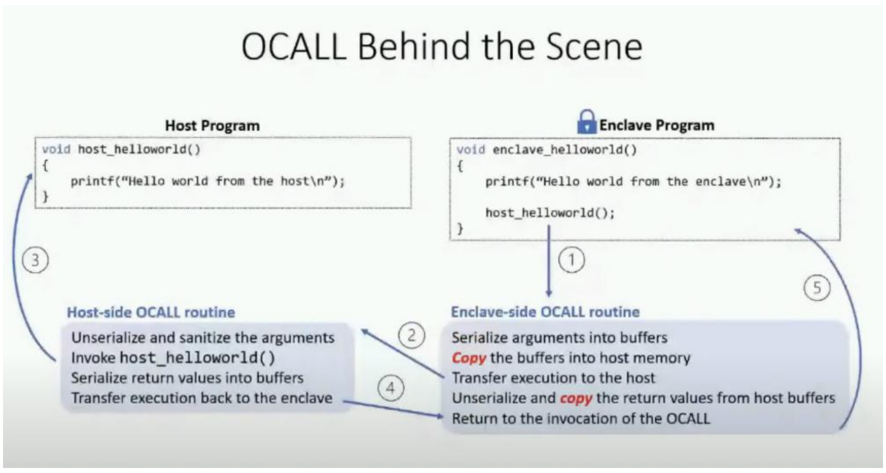
```
void enclave_helloworld()
{
    printf("Hello world from the enclave\n");

    host_helloworld();
}
```

Host-side ECALL routine
- Serialize arguments into buffers
- Transfer execution to the enclave
- Unserialize the return values from host buffers
- Return to the invocation of the ECALL

Enclave-side ECALL routine
- Copy the host buffers into enclave memory
- Unserialize and sanitize the arguments
- Invoke enclave_helloworld()
- Serialize return values into buffers
- Copy the buffers into host buffers
- Transfer execution back to the host

Figure 11. Azure Intel-SGX Enclaves – ECALL Behind the Scene [18]

Once the generated code is ready, developers can implement the defined enclave functions in both the host and enclave sides. Writing a program on top of OE is very similar to writing a normal C program. The main function in the host program takes an argument that specifies the path to the signed enclave binary. It then creates the enclave, invokes the enclave function (like hello_world), and handles the execution flow. Behind the scenes, the SDK handles low-level details such as transferring data between the host and the enclave securely. **Figure 11** and **Figure 12** briefly explain how the actual communication works between the host and the enclave [18].



```
void host_helloworld()
{
    printf("Hello world from the host\n");
}
```

```
void enclave_helloworld()
{
    printf("Hello world from the enclave\n");

    host_helloworld();
}
```

Unserialize and sanitize the arguments
Invoke `host_helloworld()`
Serialize return values into buffers
Transfer execution back to the enclave

Serialize arguments into buffers
Copy the buffers into host memory
Transfer execution to the host
Unserialize and *copy* the return values from host buffers
Return to the invocation of the OCALL

Figure 12. Azure Intel-SGX Enclaves – OCALL Behind the Scene [18]

LIMITATIONS & OTHER FACTORS

Below are some additional comparisons between both technologies:

1) **Performance Limitations:** Both systems introduce additional overhead due to the attestation process, which is used to maintain the integrity of the enclaves. In AWS Nitro Enclaves, the use of vsock-based secure channels adds complexity and latency [19], whereas in SGX Enclaves, the management of the encrypted page cache (EPC) results in increased latency [20].
2) **Memory Limitations:** AWS Nitro Enclaves support dynamic memory allocation [21]. The total memory for enclaves is carved out from the host EC2 instance's memory using an allocator service. In contrast, Azure SGX Enclaves have static EPC memory allocation. The DCsv2 series supports up to 168 MiB of EPC memory, while the DCsv3 series supports up to 256 GiB of EPC memory [22].
3) **Regional Availability:** As of March 2024, AWS Nitro Enclave EC2 instances are available in 25 out of 36 global regions [23], whereas Azure SGX Enclave VMs are available in 6 regions out of 60 regions [24].
4) **Instance Types:** All AWS EC2 instances developed since 2018 are equipped with the AWS Nitro System and support Nitro Enclaves. However, bare metal instances, the T3 family, and any instance with only 1 CPU core are not compatible with Nitro Enclaves. For Microsoft Azure, the DCsv2 and DCsv3 series are the two VM families that currently support Intel SGX Enclaves [21] [24].

CONCLUSION

In this article, we have examined the growing need for confidential computing in today's public cloud landscape, driven by the increasing importance of data privacy and security. Further, we conducted a comparative review of AWS Nitro Enclaves and Azure VMs with Intel SGX Enclaves across several critical aspects, including their architecture, attestation processes, communication mechanisms, performance and memory limitations, and additional factors such as regional availability and supported instance types.

In conclusion, while both of these technologies offer secure, isolated environments for confidential computing, the underlying mechanisms are vastly different with respect to how they are enabled and how they operate. The choice between the two will largely depend on factors such as specific workload requirements, geographical constraints, and the need for remote attestation. Azure's integration with OpenEnclave SDK and other open-source frameworks offers an advantage with respect to external support required from application code changes, while AWS's dynamic memory and vCPU allocation provide flexibility for varying workloads. Both these cloud providers are continuing to evolve in the confidential computing space, addressing the increasing demand for secure and private data processing in the cloud.

ACKNOWLEDGEMENT

The authors would like to thank Amazon Web Services (AWS), Microsoft Azure, and the Confidential Computing Consortium for their contributions to the development of confidential computing technologies.

REFRENCES

[1] A. Solterbeck, “Protecting data at rest and in motion,” Network Security, vol. 2006, pp. 14–17, Sep. 2006. DOI: 10.1016/S1353-4858(06)70424-X.

[2] J. S. Hui, “Systems, methods and protocols for securing data in transit over networks,” 2002.

[3] S. De Capitani di Vimercati, S. Foresti, and P. Samarati, “Data security issues in cloud scenarios,” in 2015. DOI: 10.1007/978-3-319-26961-0_1.

[4] Microsoft Developer. “How to use Azure Confidential Computing using Intel SGX to protect apps and solutions | BRK412” Accessed: 2025-01-13. (2021), [Online]. Available: https://www.youtube.com/watch?v=Q6-nD1PvWco

[5] S. Bhaimia, “The general data protection regulation: The next generation of eu data protection,” Legal Information Management, 2018. DOI: 10.1017/S1472669618000051.

[6] Confidential Computing Consortium, “Confidential Computing: The next frontier in cloud security,” Accessed: 2025-01-13. (2022), [Online]. Available: https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC_outreach_whitepaper_updated_November_2022.pdf

[7] Confidential Computing Consortium, “A technical analysis of confidential computing,” Accessed: 2025-01-13. (2023), [Online]. Available: https://confidentialcomputing.io/wp-content/uploads/sites/10/2023/03/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.3_unlocked.pdf

[8] Amazon Web Services, “Security design of the AWS Nitro System, ” Accessed: 2025-01-13. (2023), [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.html

[9] AMD, “SEV-SNP: Strengthening VM Isolation with Integrity Protection and More,” Accessed: 2025-01-13. (2023), [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[10] Intel, “Overview of Signing and Whitelisting Intel SGX Enclaves,” Accessed: 2025-01-13. (2023), [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/overview-signing-whitelisting-intel-sgx-enclaves.pdf

[11] Amazon Web Services, “AWS Nitro Enclaves Attestation Process,” Accessed: 2025-01-13. (2023), [Online]. Available: https://github.com/aws/aws-nitro-enclaves-nsm-api/blob/main/docs/attestation_process.md

[12] Microsoft, “Azure Attestation Overview,” Accessed: 2025-01-13. (2024), [Online]. Available: https://learn.microsoft.com/en-us/azure/attestation/overview

[13] Amazon Web Services, “AWS Nitro Enclaves FAQ,” Accessed: 2025-01-13. (2023), [Online]. Available: https://aws.amazon.com/ec2/nitro/nitro-enclaves/faqs/

[14] Amazon Web Services, “AWS Nitro Enclaves: Isolated EC2 Environments to Process Confidential Data,” Accessed: 2025-01-13. (2020), [Online]. Available: https://aws.amazon.com/blogs/aws/aws-nitro-enclaves-isolated-ec2-environments-to-process-confidential-data/

[15] Microsoft, “Application Development with Azure Confidential Computing,” Accessed: 2025-01-13. (2024), [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/application-development

[16] Zheng, W., Wu, Y., Wu, X. *et al.* A survey of Intel SGX and its applications. *Front. Comput. Sci.* 15, 153808 (2021). https://doi.org/10.1007/s11704-019-9096-y.

[17] Open Enclave, “Getting Started with Open Enclave SDK,” Accessed: 2025-01-13. (2023), [Online]. Available: https://github.com/openenclave/openenclave/tree/master/docs/GettingStartedDocs

[18] M.W. Shih, “Open Enclave by Ming-Wei Shih (Microsoft) | OC3 2021,” Accessed: 2025-01-13. (2021) [Online]. Available: https://www.youtube.com/watch?v=4ni7xRb6YvM

[19] Anjuna Security, “Performance Guidelines for Nitro Enclaves,” Accessed: 2025-01-13. (2023), [Online]. Available: https://docs.anjuna.io/nitro/latest/getting_started/best_practices/performance_guidelines.html

[20] C. Zhao, D. Saifuding, H. Tian, Y. Zhang and C. Xing, "On the Performance of Intel SGX," 2016 13th Web Information Systems and Applications Conference (WISA), Wuhan, China, 2016, pp. 184-187, doi: 10.1109/WISA.2016.45.

[21] Amazon Web Services, “Amazon EC2 Nitro Enclaves User Guide,” Accessed: 2025-01-13. (2023), [Online]. Available: https://docs.aws.amazon.com/enclaves/latest/user/nitro-enclave.html

[22] Microsoft, “Azure Virtual Machine Solutions for Intel SGX,” Accessed: 2025-01-13. (2023), [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/virtual-machine-solutions-sgx

[23] Amazon Web Services, “AWS Nitro Enclaves Now Available in Asia Pacific (Hyderabad),” Accessed: 2025-01-13. (2024), [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2024/03/aws-nitro-enclaves-asia-pacific-hyderabad/

[24] Microsoft, “Quick Create an Azure Confidential Computing VM using the Azure Portal,” Accessed: 2025-01-13. (2023), [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/quick-create-portal