Confidential Computing on AWS: Protecting Data in Use with Nitro Enclaves

8 April 2026 - 9 min. read

AWS Nitro Enclaves *Confidential Computing* *Data Protection*

Cloud security has reached a significant level of maturity, ensuring high reliability and making it possible to meet most security compliance checklists with relatively little effort. Over the years, powerful tools and best practices have emerged; we have learned how to secure virtual network perimeters, manage identities granularly, and make widespread use of strong encryption for data in transit and at rest.

Yet, for a long time, a critical blind spot remained within this digital fortress—an inevitable moment of vulnerability where data, in order to be actually processed by CPUs, must sit unencrypted in memory.

In this article, we will explore the concept of **Confidential Computing**, the technological paradigm that has permanently closed this security gap, redefining protection standards in the Cloud. We will examine the complex technological challenge behind protecting data during processing, explain why traditional containers - upon which almost all modern architectures rely - fall short for highly sensitive workloads, and show how **AWS Nitro Enclaves** provides an elegant, bulletproof solution for building true cryptographic "vaults" for critical applications.

Beyond storage and networking: what is Confidential Computing?

To fully grasp the importance and impact of Confidential Computing, we need to look at the entire data lifecycle through the lens of the so-called **Data Protection Trilemma**. Until recently, available tools could successfully address only two of these three fronts:

- **Data at Rest**: When data is stored in a relational database, on a block volume (Amazon EBS), or in object storage (like Amazon S3), it is protected by robust symmetric encryption algorithms (e.g., AES-256) integrated with Key Management services like AWS KMS. If someone were to physically steal a hard drive from the data center, or if a bucket were misconfigured, the attacker would only face a meaningless, unusable jumble of bits.
- **Data in Transit**: When data travels across the network - from a client device to the back-end, or laterally between microservices within a VPC - industry-standard cryptographic protocols like TLS/SSL (often in *mutual TLS* mode) are utilized. This encrypted tunnel prevents data from being intercepted, snooped on, or tampered with during transit, neutralizing threats like *man-in-the-middle* or *sniffing* attacks.
- **Data in Use**: Here lies the real challenge, the weak link in the chain. For applications to process data - whether that means running a database query and formatting the results for the frontend, calculating the total of a financial transaction, or running AI inference on medical records - **the data must reside unencrypted in RAM** and processor registers. In that precise moment, the data is vulnerable.

Confidential Computing encompasses the hardware and software technologies specifically designed to protect this *Data in Use*. By leveraging deep isolation based on specialized hardware components, a **TEE (Trusted Execution Environment)** is established. A TEE is a secure, processor-level isolated execution environment where plaintext data and the code processing it cannot be viewed, exfiltrated, or altered from the outside. Not even by those who hold the highest administrative privileges on the underlying infrastructure. Today, in an era dominated by the training of proprietary LLM models and *Multi-Party Computation* (where different companies collaborate by merging sensitive datasets to gain insights without exposing raw data to partners), Confidential Computing is no longer a niche technology restricted to government or banking sectors.

Container isolation DOES NOT protect data in use

Taking a step back to accurately analyze the dynamics of the problem Confidential Computing aims to solve, in a Cloud architecture based on virtual machines (EC2 instances) running fleets of containers orchestrated by platforms like Docker or Kubernetes, we must ask an uncomfortable question: who else besides me can access the container?

The answer is the system administrator of the host instance, or better yet, the **Root** user.

It is crucial to internalize that the isolation offered by containers is purely logical (based on *namespaces* and *cgroups*), not physical. Multiple containers running on the same EC2 instance share the same Kernel and, ultimately, the same RAM (albeit logically partitioned). This means that if a malicious actor—or even just an *insider threat* like a rogue employee, or malware that has achieved *privilege escalation*—manages to gain Root privileges on the instance hosting the containers, the security game is over.

However, containers are not the villain of this story. In fact, avoiding them and falling back on processes executed directly on a traditional EC2 instance does not solve the root of the problem; rather, it exposes you to even more direct attack vectors. In this scenario, the attacker doesn't even need to gain Root privileges at the OS level. Compromising the single system user running the application is enough: by exploiting a security flaw in the code (various types of attacks are possible), the attacker can gain legitimate access to the portion of RAM allocated to that specific process.

In both cases, once memory access is gained, the outcome is identical: a **complete memory dump** of the target process can be executed. In a fraction of a second, a file containing exactly everything an application was processing in plaintext at that moment can be saved to disk. By analyzing the dump, the attacker can extract various critical and sensitive pieces of information: active JWT session tokens, user passwords, cryptographic *seeds* for generating TOTP codes, private SSL keys, and Personally Identifiable Information (PII).The key concept here is the **TCB** (*Trusted Computing Base*). The TCB represents the sum of all hardware, software, and human entities that must be blindly "trusted" to guarantee system security. In a traditional container-based scenario, the TCB includes the Cloud Provider's infrastructure, the Hypervisor layer, the entire Host operating system, orchestration daemons (kubelet, dockerd), and all system administrators. For truly mission-critical and sensitive workloads, this trusted circle might be far too wide.

The solution: AWS Nitro Enclaves

To drastically reduce this TCB, **AWS Nitro Enclaves** comes into play. This service is built on the foundations of AWS Nitro System technology, the custom hardware architecture from AWS that revolutionized cloud computing by physically decoupling

networking, storage, and management functions from the instance's main CPU. Nitro Enclaves leverages this architecture to allow you to "carve out" and create fully isolated CPU and Memory partitions directly from an existing "Parent" EC2 instance.

An Enclave should not be confused with a "more secure" container or a stripped-down VM; it is a completely different architectural and conceptual paradigm:

- **Silicon-level isolation**: CPU and RAM allocated to the Enclave at startup are literally and physically removed from the availability and mapping of the Parent instance. The host operating system (and consequently the almighty Root user) simply "no longer sees" that portion of memory, as if it had never been installed on the motherboard. Dumping non-existent memory from the host is technically impossible.
- **No interactive access**: Inside the Enclave, no SSH server or remote access daemon is running. There is no shell to connect to, no emergency console. Nobody—not even the lead developer who wrote the application, the AWS account administrator, or AWS technical support themselves—has the power to "log in" and see what's happening. This is the true embodiment of the *Zero Access* principle.
- **No external networking**: By design, the Enclave does not possess a TCP/IP network stack, an IP address, or routing toward the VPC or the Internet. The only way the confidential code can communicate with the outside world (i.e., the Parent instance) is via a bidirectional hypervisor-level communication channel called **Vsock**. The Parent instance must therefore act as a strictly controlled proxy to forward input and output, preventing any direct network attacks on the Enclave.
- **Cryptographic Attestation**: Before processing begins, the Nitro Hypervisor deterministically calculates a series of hashes (PCRs) that mathematically "prove" the identity and integrity of the running code and the Enclave's kernel. The Enclave can use this "cryptographic ID card" to obtain decryption keys or other secrets from AWS KMS. PCRs can be included in authorization policies, ensuring that no one has injected malicious code or altered the original image.

Even if the Parent EC2 instance were completely compromised by a hacker, the attacker would at most find the proxy software bridging to the Enclave. The "engine" grinding through sensitive data, along with plaintext keys and unencrypted data, would remain locked in an impenetrable black box.

In practice: using Nitro Enclaves

Despite the immense cryptographic complexity and underlying hardware isolation, AWS has worked to make the Developer Experience surprisingly simple and accessible through the **nitro-cli** command-line tool. The major advantage for development teams is that there is no need to learn new languages or rewrite applications from scratch: the starting point is always a standard Docker image.
Here are the three essential steps to package and launch a confidential process.

1. Image preparation

Assuming you have a containerized application named **my-secure-app** containing the logic to decrypt and process sensitive data, the first step is to convert the standard Docker image into a special, verifiable format called EIF (Enclave Image Format). Directly on the EC2 instance (previously configured via Launch Template to support Enclaves), run the build command:

```
nitro-cli build-enclave \
    --docker-uri my-secure-app:latest \
    --output-file my-secure-app.eif
```

This command packages the application alongside a minimal Linux kernel, generates the resulting **.eif** file, and—crucially for security—outputs the PCR measurements to the screen. These cryptographic hashes uniquely identify the newly created environment. These specific hashes are exactly what you will meticulously configure in the Condition Keys of your AWS KMS policies to unlock Attestation.

2. Running the Enclave

Once you have the EIF image, you can start the Enclave. This command will physically subtract CPU cores and megabytes of RAM from the Parent instance, assigning them exclusively to the new shielded environment:

```
nitro-cli run-enclave \
    --eif-path my-secure-app.eif \
    --cpu-count 2 \
    --memory 1024 \
    --enclave-cid 16
```

The critical parameter here is **--enclave-cid**, which assigns a numerical identifier (Context ID) to the Enclave. You will use this specific CID within the Parent instance to

establish communication with the Enclave through the Vsock.

3. Checking the status

To verify that the Enclave is running properly (remembering that the environment is isolated and you cannot use classic orchestration commands like **docker ps** or traditional monitoring), you can query the Nitro allocator by running:

```
nitro-cli describe-enclaves
```

This command will return the Enclave's status, uptime, and allocated resources. If, at this point, you tried to simulate an attacker's behavior by snooping through the active processes of the Parent instance's operating system using **ps aux**, or tried forcing a RAM dump, you would find absolutely no trace of the running binary, let alone the plaintext data managed inside **my-secure-app.eif**.

Conclusion

The shift to Confidential Computing represents a real and profound strengthening of the "Trust" paradigm among cloud providers, software development companies, and the end-users who entrust them with their most intimate information.

Permanently removing the system administrator and the underlying infrastructure from the trust equatio - successfully locking down data even and especially while it is being processed - is the only key to unlocking revolutionary new use cases. Consider the secure sharing of intellectual property for collaborative Artificial Intelligence model training, or the cross-analysis of shared banking data between institutions for fraud prevention.

With AWS Nitro Enclaves, building these impregnable digital "vaults" is no longer a laboratory experiment; it has become a seamless and integral part of the standard software development lifecycle.

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-

seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Alessio Gandini

Cloud-native Development Line Manager @ beSharp, DevOps Engineer and AWS expert.Since I was still in the Alfa version, I'm a computer geek, a computer science-addicted, and passionate about electronics all-around.At this moment, I'm hanging out in the IoT world, also exploring the voice user experience, trying to do amazing (Io)Things.Passionate about cinema and great TV series consumer, Sunday videogamer

---