# Table of Contents

What is Nitro Enclaves? .................................................................................................................. 1
Learn more ........................................................................................................................................ 1
Requirements ..................................................................................................................................... 2
Considerations ..................................................................................................................................... 2
Pricing ................................................................................................................................................ 2
Related services ................................................................................................................................. 2
Nitro Enclaves concepts .................................................................................................................... 4
Enclave ............................................................................................................................................. 4
Enclave ID .......................................................................................................................................... 4
Parent instance ................................................................................................................................. 4
Enclave image file ............................................................................................................................. 4
AWS Nitro Enclaves CLI .................................................................................................................. 5
AWS Nitro Enclaves SDK .................................................................................................................. 5
Cryptographic attestation ................................................................................................................. 5
Attestation document ....................................................................................................................... 5
Platform configuration registers ...................................................................................................... 5
KMS proxy ........................................................................................................................................ 5
Vsock socket ...................................................................................................................................... 6
Getting started: Hello enclave ........................................................................................................ 7
Step 1: Prepare the enclave-enabled parent instance ....................................................................... 7
Step 2: Build the enclave image file ................................................................................................ 8
Step 3: Run the enclave .................................................................................................................... 9
Step 4: Validate the enclave ............................................................................................................ 9
Step 5: Terminate the enclave ......................................................................................................... 10
Using enclaves ................................................................................................................................ 11
Enclaves workflow ........................................................................................................................... 11
   Involved parties ............................................................................................................................. 11
   Data and environment preparation ............................................................................................... 11
   Attestation and data decryption ................................................................................................. 12
Building an enclave image file ......................................................................................................... 12
Creating an enclave .......................................................................................................................... 14
   Launch the parent instance ......................................................................................................... 14
   Create the enclave ...................................................................................................................... 15
Working with multiple enclaves ...................................................................................................... 15
Nitro Enclaves and Amazon EKS .................................................................................................... 17
   Prerequisites .................................................................................................................................. 18
   Step 1: Create a launch template ............................................................................................... 19
   Step 2: Create Kubernetes cluster and node ............................................................................. 20
   Step 3: Install the Nitro Enclaves Kubernetes device plugin .................................................... 21
   Step 4: Prepare the image .......................................................................................................... 21
   Step 5: Deploy the application to the cluster ............................................................................. 22
Cryptographic attestation ................................................................................................................ 26
   Integration with AWS KMS ......................................................................................................... 26
   Where to get an enclave’s measurements ................................................................ ............... 26
     PCR0, PCR1, and PCR2 .............................................................................................................. 27
     PCR3 .......................................................................................................................................... 28
     PCR4 .......................................................................................................................................... 28
     PCR8 .......................................................................................................................................... 28
   How to get an enclave’s attestation document .......................................................................... 30
   Using cryptographic attestation with AWS KMS .................................................................... 30
     Secret data preparation ............................................................................................................. 11
     KMS key preparation ............................................................................................................... 30
   Getting started with cryptographic attestation: KMS Tool sample application .................... 32
Nitro Enclaves application development ....................................................................................... 33
What is AWS Nitro Enclaves?

AWS Nitro Enclaves is an Amazon EC2 feature that allows you to create isolated execution environments, called enclaves, from Amazon EC2 instances. Enclaves are separate, hardened, and highly-constrained virtual machines. They provide only secure local socket connectivity with their parent instance. They have no persistent storage, interactive access, or external networking. Users cannot SSH into an enclave, and the data and applications inside the enclave cannot be accessed by the processes, applications, or users (root or admin) of the parent instance. Using Nitro Enclaves, you can secure your most sensitive data, such as personally identifiable information (PII), and your data processing applications.

Note
Nitro Enclaves is processor agnostic and it is supported on most Intel, AMD, and AWS Graviton-based Amazon EC2 instance types built on the AWS Nitro System.

Nitro Enclaves also supports an attestation feature, which allows you to verify an enclave’s identity and ensure that only authorized code is running inside it. Nitro Enclaves is integrated with the AWS Key Management Service, which provides built-in support for attestation and enables you to prepare and protect your sensitive data for processing inside enclaves. Nitro Enclaves can also be used with other key management services.

Nitro Enclaves use the same Nitro Hypervisor technology that provides CPU and memory isolation for Amazon EC2 instances in order to isolate the vCPUs and memory for an enclave from a parent instance. The Nitro Hypervisor ensures that the parent instance has no access to the isolated vCPUs and memory of the enclave.

To learn more about creating your first enclave using a sample enclave application, see Getting started: Hello enclave (p. 7).

Topics
- Learn more (p. 1)
- Requirements (p. 2)
- Considerations (p. 2)
- Pricing (p. 2)
- Related services (p. 2)

Learn more

- To learn about the concepts used in Nitro Enclaves, see Nitro Enclaves concepts (p. 4).
- To get started with your first enclave using a sample enclave application, see Getting started: Hello enclave (p. 7).
• To learn about using the AWS Nitro Enclaves CLI to manage the lifecycle of enclaves, see Nitro Enclaves Command Line Interface (p. 65).
• To learn about developing custom enclave applications and the AWS Nitro Enclaves SDK, see Nitro Enclaves application development (p. 33).
• To learn about multiple enclaves, see Working with multiple enclaves (p. 15).

Requirements

Nitro Enclaves has the following requirements:

- Parent instance requirements:
  - Virtualized Nitro-based instance
    - Intel or AMD-based instances with at least 4 vCPUs, excluding M7i, M7i-flex, T3, T3a, and u-*
    - AWS Graviton-based instances with at least 2 vCPUs, excluding A1, T4g, G5g, Im4gn, and Is4gen
  - Linux or Windows (2012 R2 or later) operating system
- Enclave requirements:
  - Linux operating system only

Considerations

Keep the following in mind when using Nitro Enclaves:

- Nitro Enclaves is supported in the following Regions: US East (Ohio), US East (N. Virginia), US West (N. California), US West (Oregon), Africa (Cape Town), Asia Pacific (Hong Kong), Asia Pacific (Jakarta), Asia Pacific (Mumbai), Asia Pacific (Osaka), Asia Pacific (Seoul), Asia Pacific (Tokyo), Canada (Central), Europe (Frankfurt), Europe (Ireland), Europe (London), Europe (Milan), Europe (Paris), Europe (Stockholm), Middle East (Bahrain), South America (São Paulo), AWS GovCloud (US-East), and AWS GovCloud (US-West).
- You can create up to four individual enclaves per parent instance.
- Enclaves can communicate only with the parent instance. Enclaves running on the same or different parent instances cannot communicate with each other.
- Enclaves are active only while their parent instance is in the running state. If the parent instance is stopped or terminated, its enclaves are terminated.
- You cannot enable hibernation and enclaves on the same instance.
- Nitro Enclaves is not supported on Outposts.
- Nitro Enclaves is not supported in Local Zones or Wavelength Zones.

Pricing

There are no additional charges for using Nitro Enclaves. You are billed the standard charges for the Amazon EC2 instance and for the other AWS services that you use.

Related services

Nitro Enclaves is integrated with the following AWS services:
AWS Key Management Service

AWS Key Management Service (KMS) makes it easy for you to create and manage cryptographic keys and control their use across a wide range of AWS services and in your applications. Nitro Enclaves integrates with AWS KMS and it allows you to perform selected KMS operations from the enclave using the AWS Nitro Enclaves SDK. These operations can be tied to the cryptographic attestation process of Nitro Enclaves by setting a AWS KMS key policy to ensure that the operation works only when the measurements of the enclave match the KMS key policy. For more information, see AWS KMS condition keys for Nitro Enclaves in the AWS Key Management Service Developer Guide.

AWS Certificate Manager

AWS Certificate Manager (ACM) is a service that lets you easily provision, manage, and deploy public and private Secure Sockets Layer/Transport Layer Security (SSL/TLS) certificates for use with AWS services and your internal connected resources. SSL/TLS certificates are used to secure network communications and to establish the identity of websites over the internet, as well as resources on private networks. ACM removes the time-consuming manual process of purchasing, uploading, and renewing SSL/TLS certificates. For more information, see Nitro Enclaves application: AWS Certificate Manager for Nitro Enclaves (p. 48).
Nitro Enclaves concepts

The following concepts are important to your understanding and use of AWS Nitro Enclaves.

Concepts
- Enclave (p. 4)
- Enclave ID (p. 4)
- Parent instance (p. 4)
- Enclave image file (p. 4)
- AWS Nitro Enclaves CLI (p. 5)
- AWS Nitro Enclaves SDK (p. 5)
- Cryptographic attestation (p. 5)
- Attestation document (p. 5)
- Platform configuration registers (p. 5)
- KMS proxy (p. 5)
- Vsock socket (p. 6)

Enclave

An enclave is a virtual machine with its own kernel, memory, and CPUs. It is created by partitioning memory and vCPUs from a Nitro-based parent instance. An enclave has no external network connectivity, and no persistent storage. The enclave's isolated vCPUs and memory can't be accessed by the processes, applications, kernel, or users of the parent instance.

Enclave ID

An enclave ID is a unique identifier across AWS. It consists of the parent instance ID and an identifier for each enclave created by the instance. For example, an enclave created by a parent instance with an ID of i-1234567890abcdef0 could have an enclave ID of i-1234567890abcdef0-enc9876543210abcde.

Parent instance

The parent instance is the Amazon EC2 instance that is used to allocate CPU cores and memory to the enclave. The resources are allocated to the enclave for the duration of its lifetime. The parent instance is the only instance that can communicate with its enclave.

Enclave image file

An enclave image file (.eif) includes a Linux operating system, libraries, and enclave applications that will be booted into an enclave when it is launched.
AWS Nitro Enclaves CLI

The AWS Nitro Enclaves CLI (Nitro CLI) is a command line tool that is used to create, manage, and terminate enclaves. The Nitro CLI must be installed and used on the parent instance. For more information, see Nitro Enclaves Command Line Interface (p. 65).

AWS Nitro Enclaves SDK

The AWS Nitro Enclaves SDK is an open-source library that you can use to develop enclave applications, or to update existing applications to run in an enclave. The SDKs also integrate with AWS KMS and provide built-in support for cryptographic attestation and other cryptographic operations. For more information, see Nitro Enclaves application development (p. 33).

Cryptographic attestation

Cryptographic attestation is the process that an enclave uses to prove its identity and build trust with an external service. Attestation is accomplished using a signed attestation document that is generated by the Nitro Hypervisor. The values in an enclave's attestation document can be used as a condition for an authorization decision by an external party. AWS KMS allows you to use attestation document values in conditions keys to grant access to specific cryptographic operations. For more information, see Cryptographic attestation (p. 26).

Attestation document

An attestation document is generated and signed by the Nitro Hypervisor. It contains information about the enclave, including platform configuration registers (PCRs), a cryptographic nonce, and additional information that you can define. It can be used by an external service to verify the identity of an enclave and to establish trust. You can use the attestation document to build your own cryptographic attestation mechanisms, or you can use it with AWS KMS, which provides built-in support for authorizing cryptographic requests based on values in the attestation document. For more information, see Cryptographic attestation (p. 26).

Platform configuration registers

Platform configuration registers (PCRs) are cryptographic measurements that are unique to an enclave. Some PCRs are automatically generated when the enclave is created, and they can be used to verify that no changes have been made to the enclave since it was created. You can also manually create additional PCRs that can be used to ensure that the enclave is running on the instance on which you expect it to run. PCRs are included in the attestation document that is generated by the Nitro Hypervisor. You can use PCRs to create condition keys for AWS KMS keys. For more information, see Where to get an enclave’s measurements (p. 26).

KMS proxy

The KMS proxy is used by enclaves running on a parent instance to call AWS KMS through the parent instance’s networking. The proxy ships with Nitro CLI and it runs on the parent instance. The proxy
is required only if you use AWS KMS as your key management service and you perform AWS KMS operations (kms-decrypt, kms-generate-data-key, and kms-generate-random) using the Nitro Enclaves SDK. Sessions with KMS are established logically between AWS KMS and the enclave itself, and all session traffic is protected from the parent instance and from other enclaves.

Vsock socket

Vsock is a local communication channel between a parent instance and its enclaves. It is the only channel of communication that an enclave can use to interact with external services. An enclave launched from a parent instance will share the vsock with other enclaves launched from the same parent instance. An enclave's vsock address is defined by a context identifier (CID) that you can set when launching an enclave. Each enclave running on a parent instance gets a unique CID. The CID used by the parent instance is always 3.

On Linux, Vsock utilizes standard, well-defined POSIX socket APIs, such as connect, listen, and accept. On Windows, the Vsock uses the standard Windows sockets (Winsock2) API.
Getting started: Hello enclave

The following tutorial walks you through the basics of using AWS Nitro Enclaves. It shows you how to launch an enclave-enabled parent instance, how to build an enclave image file, how to validate that an enclave is running, and how to terminate an enclave when it is no longer needed.

The tutorial uses the Hello Enclaves sample application.

Important
The steps for Windows and Linux parent instances are mostly similar. However, the nitro-cli build-enclave command referenced in Step 2: Build the enclave image file is not supported on Windows instances. If you are using a Windows instance, you must complete this step on a Linux instance and then transfer the enclave image file (.eif) to your Windows parent instance before continuing with the remainder of the tutorial.

Steps
- Step 1: Prepare the enclave-enabled parent instance (p. 7)
- Step 2: Build the enclave image file (p. 8)
- Step 3: Run the enclave (p. 9)
- Step 4: Validate the enclave (p. 9)
- Step 5: Terminate the enclave (p. 10)

Step 1: Prepare the enclave-enabled parent instance

Launch the parent instance that you will use to create the enclave, and prepare the instance to run Nitro Enclaves.

To prepare the parent instance

1. Launch the instance using the `run-instances` command and set the `--enclave-options` parameter to `true`. At a minimum, you must also specify a Windows or Linux AMI and a supported instance type. For more information, see Requirements (p. 2).

   The following example command launches an m5.xlarge instance using the Amazon Linux 2 Kernel 5.10 AMI.

   ```
   $ aws ec2 run-instances
   --image-id ami-0b5eea76982371e91
   --count 1
   --instance-type m5.xlarge
   --key-name your_key_pair
   --enclave-options 'Enabled=true'
   ```

2. Connect to the parent instance. For more information about connecting to an instance, see the following topics in the Amazon EC2 User Guide.

   - Connect to your Linux instance
   - Connect to your Windows instance
3. Install the AWS Nitro Enclaves CLI on the parent instance.
   - If you are using a Linux parent instance, you must preallocate the memory and vCPUs. For the purposes of this tutorial, you must preallocate at least 2 vCPUs and 512 MiB of memory. For more information, see Installing the Nitro Enclaves CLI on Linux (p. 65).
   - If you are using a Windows parent instance, see Installing the Nitro Enclaves CLI on Windows (p. 67).

Step 2: Build the enclave image file

**Important**

Only Linux-based operating systems can run inside an enclave. Therefore, you must use a Linux instance to build your enclave image file .eif. As a result of this, the nitro-cli build-enclave command referenced in this section is not supported on Windows instances. If you are using a Windows parent instance, you must complete this step on a Linux instance and then transfer the resulting enclave image file (.eif) to your Windows parent instance. In this case, you must launch a temporary Linux instance and install the AWS Nitro Enclaves CLI on that instance. For more information, see Installing the Nitro Enclaves CLI on Linux (p. 65). After you have launched the temporary Linux instance and you have installed the AWS Nitro Enclaves CLI, connect to that instance and perform the steps described here. After you have completed the steps, transfer the enclave image file (.eif) to your Windows parent instance, reconnect to your Windows parent instance and continue with Step 3: Run the enclave.

The Hello Enclave application is located in the /usr/share/nitro_enclaves/examples/hello directory.

To build the enclave image file

1. Build a docker image from the application. The following command builds a Docker image named hello with a tag of latest.

   ```bash
   $ docker build /usr/share/nitro_enclaves/examples/hello -t hello
   ```

2. Run the following command to verify that the Docker image has been built.

   ```bash
   $ docker image ls
   ```

3. Convert the Docker image to an enclave image file by using the nitro-cli build-enclave (p. 68) command. The following command builds an enclave image file named hello.eif.

   ```bash
   $ nitro-cli build-enclave --docker-uri hello:latest --output-file hello.eif
   ```

Example output

```
Start building the Enclave Image...
Enclave Image successfully created.
"Measurements": {
    "HashAlgorithm": "Sha384 { ... }",
    "PCR0": "2bb885ee2104203393c0d2f335f14061a9cd9154e4ede772a6a474d3679348fb33c4917d54fee3f11f9e7a49a0ef305c",
    "PCR1": "aca6e62ffbf5f7deccac452d7f8ceee1b94048af62af6c16c8a6b6c9feed8c38010c75a669f9a36ea69032f0b9739218955",
    "PCR2": "40686da34bb450210dccc234fbb95826ecf81f67e4496b6182ba8eb7ab018977dce07448cd0b7e44346dce1e283e5e64"
}
```
Step 3: Run the enclave

You can now use the hello.eif enclave image file to run the enclave. In this tutorial, you will run an enclave with 2 vCPUs and 512 MiB of memory using the hello.eif enclave image file. You will also create the enclave in debug mode.

Note

Enclaves booted in debug mode generate attestation documents with PCRs that are made up entirely of zeros (00000000000000000000000000000000000000000). These attestation documents cannot be used for cryptographic attestation.

Use the nitro-cli run-enclave (p. 70) command. Specify the vCPUs, memory, and the path to the enclave image file, and include the --debug-mode option.

```
$ nitro-cli run-enclave --cpu-count 2 --memory 512 --enclave-cid 16 --eif-path hello.eif --debug-mode
```

Example output

```
Start allocating memory...
Started enclave with enclave-cid: 16, memory: 512 MiB, cpu-ids: [1, 3]
[
  "EnclaveID": "i-05f6ed443aEXAMPLE-enc173dfe3eEXAMPLE",
  "ProcessID": 7077,
  "EnclaveCID": 16,
  "NumberOfCPUs": 2,
  "CPUIDs": [
    1,
    3
  ],
  "MemoryMiB": 512
]
```

The enclave is now running. In this sample output, the enclave is created with an ID of i-05f6ed443aEXAMPLE-enc173dfe3eEXAMPLE, and an enclave context identifier (EnclaveCID) of 16. The EnclaveCID is like an IP address for the local socket (vsock) between the parent instance and the enclave.

Step 4: Validate the enclave

Now that you have created the enclave, you can use the nitro-cli describe-enclaves (p. 74) command to verify that it is running.

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

```
{
  "EnclaveID": "i-05f6ed443aEXAMPLE-enc173dfe3eEXAMPLE",
  "ProcessID": 7077,
}
```
Step 5: Terminate the enclave

If you no longer need the enclave, you can use the `nitro-cli terminate-enclave` command to terminate it.

```bash
$ nitro-cli terminate-enclave --enclave-id i-05f6ed443aEXAMPLE-enc173dfe3eEXAMPLE
```

Example output

```json
Successfully terminated enclave i-05f6ed443aEXAMPLE-enc173dfe3eEXAMPLE. 
{
  "EnclaveID": "i-05f6ed443aEXAMPLE-enc173dfe3eEXAMPLE",
  "Terminated": true
}
```
Using enclaves

The following section provides an overview of working with AWS Nitro Enclaves.

Topics

• Enclaves workflow (p. 11)
• Building an enclave image file (p. 12)
• Creating an enclave (p. 14)
• Working with multiple enclaves (p. 15)
• Using Enclaves with Amazon EKS (p. 17)

Enclaves workflow

The following topic explains some of the roles and basic workflows of AWS Nitro Enclaves, using AWS KMS as the key management service, and Amazon S3 as the data storage service.

Topics

• Involved parties (p. 11)
• Data and environment preparation (p. 11)
• Attestation and data decryption (p. 12)

Involved parties

A typical Nitro Enclaves use case involves multiple parties. Each party is responsible for completing certain tasks to ensure that the enclave is operational. A typical use case includes the following parties:

• **Data owner**—Owns the AWS KMS key and the secret data. The owner is responsible for creating the KMS key in AWS KMS, encrypting the secret data, and making the encrypted data and the encrypted data key available.

• **Parent instance administrator**—Owns the parent instance and manages the enclave's lifecycle. This party launches the parent instance and then creates the enclave using the enclave image file or Docker image, which is provided by the application developer. The parent instance administrator should not have permission to perform cryptographic actions using the KMS key, and they should not have permission to change the KMS key policy. The parent instance however, will need permissions to call `kms-decrypt` using the KMS key, but the request will only succeed if it is made from inside the enclave, and it includes values that match the condition keys in the KMS key policy.

• **Application developer**—Develops the applications that run in the enclave and on the parent instance. The developer packages the application into an enclave image file or Docker image and provides it to the parent instance administrator, who uses it to create the enclave. The application developer might also develop applications that run on the parent instance itself.

Data and environment preparation

The following section provides an overview of the data encryption process, attestation set up, and enclave creation process.
Attestation and data decryption

The following section provides an overview of the attestation and data decryption process.

1. Download the encrypted data and the encrypted data key from Amazon S3 to the parent instance.
2. Transfer the encrypted data and the encrypted data key to the enclave over the vsock socket.
3. Call the kms-decrypt Nitro Enclaves SDK, which sends the encrypted data key and the attestation document to AWS KMS. The attestation document includes the enclave’s PCRs and public key. The request is sent over the vsock socket to the parent instance, and the parent instance forwards the request to AWS KMS via the AWS KMS proxy.
4. AWS KMS receives the request and verifies that the attached attestation document is signed by the Nitro Hypervisor. AWS KMS then compares the PCRs in the attestation document with the PCRs in the condition keys in the policy of the requested KMS key.
5. If the PCRs in the attestation document match the PCRs in the condition keys of the KMS key policy, AWS KMS encrypts the plaintext data key with the enclave's public key from the attestation document.
6. The encrypted plaintext data key is returned to the parent instance over the KMS proxy, and the parent instance sends it to the enclave over the vsock socket.
7. The encrypted plaintext data key is decrypted using the enclave's private key.
8. The plaintext data key is used to decrypt the encrypted data.
9. The data is now ready to be processed inside the enclave.

Building an enclave image file

After you have developed an enclave application, you are ready to package it as an enclave image file (.eif). An enclave image file provides the information that is required to launch an enclave. It contains
This section explains how to create an enclave image file.

First, you need to package the enclave application and its dependencies into a Docker image. A Docker image is a read-only template that provides instructions for creating a Docker container. Nitro Enclaves uses Docker images as a convenient file format for packaging your applications. Docker images are typically used to create Docker containers. However, in this case, you use the Docker image to create an enclave image file instead. For more information about Docker, see the following resources:

- Docker overview
- Orientation and setup
- Build and run your image
- Best practices for writing Docker images

After you have packaged your enclave application into a Docker image, you need to convert the Docker image to an enclave image file. To do this, use the `nitro-cli build-enclave` AWS Nitro Enclaves CLI command.

**Important**

The `nitro-cli build-enclave` command is not supported on Windows instances. If you are using a Windows instance, you must complete this step on a temporary Linux instance and then transfer the resulting enclave image file (.eif) to your Windows parent instance. After you have launched the temporary Linux instance and you have installed the AWS Nitro Enclaves CLI, connect to that instance and run the `nitro-cli build-enclave` command. After you have run the command, transfer the enclave image file (.eif) to your Windows parent instance where you will create the enclave.

The `nitro-cli build-enclave` command creates an enclave image file and provides the enclave's measurements. The enclave image file is used to launch the enclave on the parent instance, and the measurements are used to set up the attestation process. For more information, see [Cryptographic attestation](p. 26).

For example, to create an enclave image file from the hello-world sample Docker image, use the following command.

```shell
$ nitro-cli build-enclave --docker-uri hello-world:latest  --output-file hello-world.eif
```

This command creates an enclave image file named `hello-world.eif`, along with the following output.

```
Start building the Enclave Image...
Enclave Image successfully created.
{
  "Measurements": {
    "HashAlgorithm": "Sha384 { ... }",
    "PCR0": "7fb5EXAMPLEcbb68ed99a13d7122abfc0666b926a79d537EXAMPLEe445c84217f59cfdd36c08b2c79552928702EXAMPLE",
    "PCR1": "235cEXAMPLEb6b993c915505f3220e2d82b51aff830ad1EXAMPLEeeec1bf0b4ae749d511c663f464cde9f718aeEXAMPL
```
Creating an enclave

After your enclave applications have been packaged as an enclave image file (.eif), you are ready to create the enclave.

To create the enclave, you need to do the following:

Steps

- Launch the parent instance (p. 14)
- Create the enclave (p. 15)

Launch the parent instance

First, you need to launch the parent instance. The parent instance is the instance from which you allocate the resources for the enclave. You also use this instance to manage the lifecycle of the enclave. For more information about the supported instance types and sizes, see Requirements (p. 2).

After you launch the parent instance, make a note of the instance ID. You'll need it to generate PCR4, which is needed for attestation. For more information, see Where to get an enclave's measurements (p. 26).

You can launch the parent instance using the Amazon EC2 console or the AWS CLI.

Amazon EC2 console

To launch the parent instance using the Amazon EC2 console

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose Launch Instance.
3. On the Choose an AMI page, choose a Linux or Windows operating system.
4. On the Choose an Instance Type page, select a supported instance type and then choose Next: Configure Instance Details.
5. On the Configure Instance Details page, for Nitro Enclaves, choose Enable, and then choose Review and Launch.
6. On the Review Instance Launch page, review the settings, and then choose Launch to choose a key pair and to launch your instance.

AWS CLI

To launch a parent instance using the AWS CLI

Use the run-instances command and set the --enclave-options parameter to Enabled=true.

For example, the following command launches a single m5.2xlarge instance using an AMI with an ID of ami-12345abcdef67890a1 and a key pair named my_key, and it enables Nitro Enclaves.

$ C:\> aws ec2 run-instances --image-id ami-12345abcdef67890a1 --count 1 --instance-type m5.2xlarge --key-name my_key --enclave-options 'Enabled=true'
Create the enclave

After you have launched the parent instance, you can create the enclave using the enclave image file (.eif). When you create the enclave, it boots the enclave application and its dependencies from the enclave image file into the enclave.

**Note**

You must have the Nitro Enclaves CLI installed on the parent instance in order to create the enclave. For more information, see [Nitro Enclaves Command Line Interface](#).

**To create the enclave**

On the parent instance, use the `nitro-cli run-enclave` CLI command and, at a minimum, specify the following:

- The number of vCPUs to allocate to the enclave
- The amount of memory (in MiB) to allocate to the enclave
- An enclave image file

For example, the following command creates an enclave with 4 vCPUs, 1600 MiB of memory, a context ID of 10, and it uses an enclave image file named `sample.eif`, which is located in the same directory from which the command is being run.

```
$ C:\> nitro-cli run-enclave --cpu-count 2 --memory 1600 --eif-path sample.eif --enclave-cid 10
```

**Example output**

```json
Instance CPUs [1, 3] going offline
Started enclave with enclave-cid: 10, memory: 1600 MiB, cpu-ids: [1, 3]
Sending image to cid: 10 port: 7000
{
  "EnclaveID": "i-abc12345def67890a-enc9876abcd543210ef12",
  "EnclaveCID": 10,
  "NumberOfCPUs": 2,
  "CPUIDs": [
    1,
    3
  ],
  "MemoryMiB": 1600
}
```

**Working with multiple enclaves**

You can create up to four separate enclaves from a single Amazon EC2 parent instance. Consider the following before using multiple enclaves.

- When launching a parent instance, choose an instance type that has enough vCPUs and memory for both the parent instance and the additional enclaves. If multi-threading is enabled, you must leave at least 2 vCPUs for the parent instance. If multi-threading is not enabled, you must leave at least 1 vCPU for the parent instance. For example, if multi-threading is enabled and you intend to run 4
enclaves with 4 vCPUs each, you must select an instance type that has at least 18 vCPUs (2 for the parent instance and 16 for the enclaves).

• When you install the Nitro CLI, you must configure the allocator service to preallocate enough vCPUs and memory for all of the enclaves. For example, if you intend to run 3 enclaves with 4 vCPUs and 2 GiB memory each, you must preallocate 12 vCPUs and 6 GiB of memory. For more information, see step 7 in Install AWS Nitro Enclaves CLI (p. 65).

• Each enclave communicates with the parent instance over vsock. Each enclave has its own vsock address that is defined by a context identifier (CID). There is no vsock connection between the enclaves.

• Each enclave has its own unique ID.

• Each enclave can be individually terminated by specifying its enclave ID.

• Each enclave can be configured with a different number of vCPUs or amount of memory.

• Each enclave on a parent instance can be created from the same or a different enclave image file.

The following image illustrates an example of using multiple enclaves. In this example, there is a single parent instance with 3 running enclaves. The parent instance is a m6g.2xlarge, which has 8 vCPUs and 32 GiB memory, running Amazon Linux 2. The parent instance has a CID of 3, and enclaves 1, 2, and 3 have unique CIDs of 14, 15, 16 respectively. Each enclave has a unique enclave ID; each ID is prefixed with the parent instance ID. Enclaves 1 and 3 were launched with the same enclave image file (FileA.eif), while enclave 2 was launched with a different enclave image file (FileB.eif). Enclave 1 has been launched with 2 vCPUs and 1600 MiB memory, enclave 2 with 1 vCPU and 700 MiB memory, and enclave 3 with 2 vCPUs and 2000 MiB memory. In total, the enclaves have been allocated with 5 vCPUs and 4300 MiB (4.2 GiB) of memory, which leaves the parent instance with 3 vCPUs and 27.8 GiB of memory. Each enclave has a vsock channel to communicate with the parent instance.
Using Enclaves with Amazon EKS

You can use Amazon Elastic Kubernetes Service to orchestrate, scale, and deploy Nitro Enclaves from a Kubernetes pod. Kubernetes is an open source platform for container orchestration. The following diagram provides a conceptual overview of how Nitro Enclaves integrates with Amazon EKS.
Important
All pods and containers in the same Amazon EKS node or Amazon EC2 instance that has the Nitro Enclaves Kubernetes device plugin installed will be able to communicate with the enclave that is attached to that parent Amazon EC2 instance.

For more information about Amazon EKS see the Amazon Elastic Kubernetes Service User Guide.

This tutorial shows how to create an Amazon EKS cluster with a managed node group and one enclave-enabled node. It shows how to install the Nitro Enclaves Kubernetes device plugin, how to prepare the Hello Enclaves sample application for deployment, and how to deploy the prepared Hello Enclaves Docker image to the cluster.

There are two key components to this process:

• Launch templates. The process requires a launch template that is properly configured. The launch template must have enclaves enabled and an it must include specific user data that is provided in Step 1: Create a launch template (p. 19). This launch template will be used to create the enclave-enabled nodes in the cluster.

• The Nitro Enclaves Kubernetes device plugin. This device plugin gives your pods and containers the ability to create and terminate enclaves using the Nitro Enclaves Command Line Interface (p. 65). The device plugin works with both Amazon EKS and self-managed Kubernetes nodes.

Tip
To simplify the build process, we also provide an open source tool called enclavectl that you can use to build and deploy your enclave applications to an Amazon EKS cluster. Using enclavectl, you can create an enclave-enabled Amazon EKS cluster and install the Nitro Enclaves device plugin as a daemonset. We also provide some sample applications and a tutorial to demonstrate how to build and run your own enclave applications in an Amazon EKS cluster. For more information about enclavectl, see the Nitro Enclaves with Kubernetes GitHub repo.

Topics
• Prerequisites (p. 18)
• Step 1: Create a launch template (p. 19)
• Step 2: Create Kubernetes cluster and node (p. 20)
• Step 3: Install the Nitro Enclaves Kubernetes device plugin (p. 21)
• Step 4: Prepare the image (p. 21)
• Step 5: Deploy the application to the cluster (p. 22)

Prerequisites

• This tutorial assumes familiarity with Kubernetes concepts. For more information, see the Kubernetes documentation.

• The following tools are required to complete this tutorial:
  • bash shell
  • AWS CLI version 2. For more information about installing the AWS CLI, see Getting started with the AWS CLI.
  • eksctl, a simple command line tool for creating and managing Kubernetes clusters on Amazon EKS. For more information, see Installing or updating eksctl.
  • Docker, an open platform used for packaging your enclave applications into images that can be deployed into containers on your worker nodes. For more information, see Get Docker.
  • jq, a command line JSON processor. For more information, see Download jq.
Step 1: Create a launch template

Create a launch template that will be used to launch the enclave-enabled worker nodes (Amazon EC2 instances) in the cluster. You can create the launch template using either the `create-launch-template` AWS CLI command or the Amazon EC2 console.

When you create the launch template, you must do the following:

1. Specify a supported instance type (p. 2).
2. Enable Nitro Enclaves.
3. Specify the following user data, which automates the AWS Nitro Enclaves CLI installation, and preallocates the memory and the vCPUs for enclaves on the instance.

   The CPU_COUNT and MEMORY_MIB variables in the user data specify the number of vCPUs and amount of memory (in MiB) respectively. For the purpose of this tutorial, the user data below specifies 2 vCPUs and 768 MiB of memory.

   ```
   Note
   You can specify a custom number of vCPUs and amount of memory, depending on your workload and instance type. When you create an enclave on the worker node, the requested memory and vCPUs can't exceed the values that you specified here.
   You can also include any other instructions in the user data that are required for your application.
   ```

   ```
   MIME-Version: 1.0
   Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="
   
   --==MYBOUNDARY==
   Content-Type: text/x-shellscript; charset="us-ascii"
   
   #!/bin/bash -e
   readonly NE_ALLOCATOR_SPEC_PATH="/etc/nitro_enclaves/allocator.yaml"
   # Node resources that will be allocated for Nitro Enclaves
   readonly CPU_COUNT=2
   readonly MEMORY_MIB=768
   
   # This step below is needed to install nitro-enclaves-allocator service.
   amazon-linux-extras install aws-nitro-enclaves-cli -y
   
   # Update enclave's allocator specification: allocator.yaml
   sed -i "s/cpu_count:.*/cpu_count: $CPU_COUNT/g" $NE_ALLOCATOR_SPEC_PATH
   sed -i "s/memory_mib:.*/memory_mib: $MEMORY_MIB/g" $NE_ALLOCATOR_SPEC_PATH
   
   # Restart the nitro-enclaves-allocator service to take changes effect.
   systemctl restart nitro-enclaves-allocator.service
   
   echo "NE user data script has finished successfully."
   
   --==MYBOUNDARY==
   ```

   Note
   If you create the launch template using the AWS CLI, you must provide the user data as base64-encoded text.

4. Note the launch template ID (for example, lt-01234567890abcdef; you'll need it in the following step.
For more information about creating a launch template, see Create a launch template in the Amazon EC2 User Guide.

Step 2: Create Kubernetes cluster and node

Create the cluster with node groups and worker nodes. In this tutorial, we use the eksctl command line tool to create an Amazon EKS cluster with one managed node group with one worker node using the launch template created in the previous step.

To create the cluster, node group, and worker node

1. Create a cluster configuration file named cluster_config.yaml and add the following configuration, which specifies the following:
   - The name of the cluster (metadata:name)
   - The AWS Region in which to create the cluster (region)
   - The name of the node group (managedNodeGroups:name)
   - The ID and version of the launch template to use to create the worker nodes (id and version)
   - The number of nodes in the node group (desiredCapacity)

   For the purpose of this tutorial, the configuration file creates a cluster named ne-cluster in us-east-1, it creates 1 node in a node group named ne-group using version 1 of launch template lt-01234567890abcdef.

   ```yaml
   apiVersion: eksctl.io/v1alpha5
   kind: ClusterConfig
   metadata:
     name: ne-cluster
     region: us-east-1
     version: "1.22"
   managedNodeGroups:
     - name: ne-group
       launchTemplate:
         id: lt-01234567890abcdef
         version: "1"
         desiredCapacity: 1
   
   For more information about cluster configuration files, see Using Config Files in the eksctl documentation.

2. Create the cluster, node group, and worker node using the cluster configuration file. Run the following eksctl command and specify the cluster configuration file created in the previous step.

   ```bash
   $ eksctl create cluster -f cluster_config.yaml
   
   After the cluster is created, you will see output similar to the following.

   ```
   ...  
   [##] EKS cluster "ne-cluster" in us-east-1 region is ready
   ```

   Note

   When you create an Amazon EKS cluster using eksctl, the tool automatically preconfigures kubectl so that it can find and access the Amazon EKS cluster. If you create a self-managed
cluster using other tooling, you might need to manually configure \texttt{kubectl} so that it can communicate with your cluster. To check the \texttt{kubectl} configuration, run \texttt{kubectl cluster-info}. If you see a URL response, \texttt{kubectl} is correctly configured to access your cluster. For more information, see Organizing Cluster Access Using kubeconfig Files in the Kubernetes documentation.

**Step 3: Install the Nitro Enclaves Kubernetes device plugin**

Deploy the Nitro Enclaves Kubernetes device plugin to the cluster and then enable it on each worker node in the cluster using \texttt{kubectl}. The plugin enables the pods on each worker node to access the Nitro Enclaves device driver. The plugin is deployed to the Kubernetes cluster as a daemonset.

**To deploy and enable the Nitro Enclaves Kubernetes device plugin**

1. Deploy the Nitro Enclaves Kubernetes device plugin to the cluster using the following command.

   ```bash
   ```

2. Get the name of the worker node on which to install the Nitro Enclaves Kubernetes device plugin using the following command.

   ```bash
   $ kubectl get nodes
   ```

   The following is example output.

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>ROLES</th>
<th>AGE</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip-123-123-123-123.us-east-1.compute.internal fb459a0</td>
<td>Ready</td>
<td>&lt;none&gt;</td>
<td>21h</td>
<td>v1.22.15-eks-</td>
</tr>
</tbody>
</table>

3. Enable the Nitro Enclaves Kubernetes device plugin on the worker node. Use the following \texttt{kubectl} command and specify the node name from the previous step.

   ```bash
   $ kubectl label node \texttt{node_name} aws-nitro-enclaves-k8s-dp=enabled
   ```

   For example:

   ```bash
   $ kubectl label node \texttt{ip-123-123-123-123.us-east-1.compute.internal} aws-nitro-enclaves-k8s-dp=enabled
   ```

**Step 4: Prepare the image**

Nitro Enclaves uses Docker images as a convenient file format for packaging your applications. You must build the Docker image that includes your enclave application and any other commands that are needed to run the application. This Docker image will be deployed to the worker node in the following step.

AWS provides a command line tool, \texttt{enclavectl} that automates the steps that are needed to build an enclave image file and to package your enclave image file into a Docker image. Additionally, the tool includes features that automate Amazon EKS cluster and node group creation, and application deployment. For an end-to-end tutorial on how to use the \texttt{enclavectl} tool to automate cluster creation, application packaging, and application deployment, see the aws-nitro-enclaves-with-k8s readme file.
Note
You can also perform these steps manually using Docker and the Nitro CLI. For more information, see Building an enclave image file (p. 12).

In this tutorial, we use the enclavectl tool to package the Hello Enclaves sample application into a Docker image.

To prepare the image
1. The enclavectl utility can be found in the aws-nitro-enclaves-with-k8s GitHub repo. Clone the GitHub repo and navigate into the directory.

   ```
   $ git clone git@github.com:aws/aws-nitro-enclaves-with-k8s.git && cd aws-nitro-enclaves-with-k8s
   ```

2. Source the env.sh script to add the enclavectl tool to your PATH variable.

   ```
   $ source env.sh
   ```

3. Configure the enclavectl for the tutorial. The settings.json file includes some default parameters that are used only if you create a cluster using enclavectl. Since the cluster was created manually in the previous steps, the parameters in the settings.json are not used; but you must run this command to configure the tool before using it.

   ```
   $ enclavectl configure --file settings.json
   ```

4. Build the Hello Enclaves enclave image file and package it into a Docker image. The required files are located in the /container/hello directory. Use the enclavectl build command and specify the name of the directory.

   ```
   $ enclavectl build --image hello
   ```

   Tip
   You can also use the enclavectl tool to package your own enclave applications into a Docker image. To do this, you must create a new directory with the name of your application in the /container directory. For example, /aws-nitro-enclaves-with-k8s/container/my-app. Then, you must create your Dockerfile and an enclave_manifest.json file in this directory. Then, when you run the enclavectl build command, for --image specify the name of the directory that you created. For example, enclavectl build --image my-app. For more information about how to use the enclavectl tool to package you applications, see How to create your own application.

5. The Docker image is created with a name in the following format: hello-unique_uuid. To view the full name of the image, run the following command.

   ```
   $ docker image ls | grep hello
   ```

Step 5: Deploy the application to the cluster

Finally, you need to deploy the application to your cluster.

To deploy the application to the cluster

1. Create a deployment specification. Create a new file named deployment_spec.yaml and add the following content.
Step 5: Deploy the application to the cluster

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: unique_deployment_name
spec:
  replicas: 1
  selector:
    matchLabels:
      app: application_name
  template:
    metadata:
      labels:
        app: application_name
    spec:
      containers:
        - name: unique_container_name
          image: docker_image_name:image_tag
          command: ["docker_image_entry_point"]
          resources:
            limits:
              aws.ec2.nitro/nitro_enclaves: "1"
              hugepages-2Mi: 768Mi
              cpu: 250m
            requests:
              aws.ec2.nitro/nitro_enclaves: "1"
              hugepages-2Mi: 768Mi
          volumeMounts:
            - mountPath: /dev/hugepages
              name: hugepage
              readOnly: false
          volumes:
            - name: hugepage-2mi
              emptyDir:
                medium: HugePages-2Mi
            - name: hugepage-1gi
              emptyDir:
                medium: HugePages-1Gi
          tolerations:
            - effect: NoSchedule
              operator: Exists
            - effect: NoExecute
              operator: Exists
```

The deployment specification must include the following Nitro Enclaves specific sections:

- **limits:**
  - `aws.ec2.nitro/nitro_enclaves: "1"
  - `hugepages-2Mi: 768Mi`

The `limits` section defines the resource limits for the container. A container can't use more resources than what is defined in the limits. For more information, see [Requests and limits](#). `aws.ec2.nitro/nitro_enclaves` is the resource name of the enclaves device driver defined in the device plugin. When the device plugin is registered, it advertises this name to kubelet. The resource name can be requested as part of a specification any time. In the template above, we specify 1 so that only one application can use the enclaves device driver at the same time. You can modify this value depending on your requirements.

- `hugepages` specifies the huge page size limit for your application. Nitro Enclaves uses large contiguous memory regions and therefore requires huge pages support. In the template above,
the huge page size limit for the application is set to 768 MiB of memory. Keep in mind that the nitro-enclaves-allocator service, which was installed to the node through the user data specified in the launch template, already allocated a huge page size based the value specified for MEMORY_MIB in the user data. For example, if the value for MEMORY_MIB in the user data is 1024, the nitro-enclaves allocator allocated one page of 1 GiB huge page type for the whole node. In this case, the field in the deployment spec be defined as hugepages-1Gi: 1Gi.

For more information, see Managing huge pages.

requests:
  aws.ec2.nitro/nitro_enclaves: "1"
  hugepages-2Mi: 768Mi

The requests section is used to define the node on which to place the pod. For more information, see Requests and limits.

In the template above, we request a pod that has one enclave device (aws.ec2.nitro/nitro_enclaves: "1"), and a huge page size of 768 MiB (hugepages-2Mi: 768Mi).

The following is an example deployment specification for the Hello Enclaves application created in the previous steps.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello_deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
      - name: hello_container
        image: 123456789012.dkr.ecr.eu-central-1.amazonaws.com/hello-b0f89e0a-7d83-4928-853c-9a3f941fa769:latest
        command: ["/home/run.sh"]
        resources:
          limits:
            aws.ec2.nitro/nitro_enclaves: "1"
            hugepages-2Mi: 768Mi
            cpu: 250m
          requests:
            aws.ec2.nitro/nitro_enclaves: "1"
            hugepages-2Mi: 768Mi
        volumeMounts:
        - mountPath: /dev/hugepages
          name: hugepage
          readOnly: false
        volumes:
        - name: hugepage
          emptyDir:
            medium: HugePages-2Mi
      tolerations:
      - effect: NoSchedule
        operator: Exists
      - effect: NoExecute
```
2. Apply the deployment specification to the cluster and deploy the application. Use the `kubectl` apply command and specify the deployment specification file.

```bash
$ kubectl apply -f deployment_spec.yaml
```

**Tip**
The `enclavectl` tool automates and simplifies the steps required to deploy an application to a cluster. You can use the `enclavectl run --image image_name` command to automatically generate a deployment specification for your application and to automatically deploy it to your cluster. For example, `enclavectl run --image hello`. If you prefer automatically generate a deployment specification for your application, but deploy it manually, add the `--prepare-only` flag. For example, `enclavectl run --image hello --prepare-only`. Doing this will generate the deployment specification but it will not deploy the application to the cluster. Once the deployment specification has been generated, you can deploy the application using the `kubectl apply` command.
Cryptographic attestation

Attestation is a unique feature available to Nitro Enclaves. The enclave uses the attestation process to prove its identity and build trust with an external service.

The attestation process uses a series of measurements that are unique to an enclave. You can use these measurements to create access policies in external services to grant the enclave access to special cryptographic operations. For more information, see Where to get an enclave's measurements (p. 26).

Using the Nitro Enclaves SDK, an enclave can request a signed attestation document from the Nitro Hypervisor that includes its unique measurements. This document can be attached to requests from the enclave to an external service. The external service can validate the measurements included in the attestation document against the values in the access policy to determine whether to grant the enclave access to the requested operation. For more information, see How to get an enclave's attestation document (p. 30).

Topics
- Integration with AWS KMS (p. 26)
- Where to get an enclave's measurements (p. 26)
- How to get an enclave's attestation document (p. 30)
- Using cryptographic attestation with AWS KMS (p. 30)
- Getting started with cryptographic attestation: KMS Tool sample application (p. 32)

Integration with AWS KMS

Nitro Enclaves includes built-in support for attestation with AWS KMS. AWS KMS has the ability to ingest attestation documents that are presented by an enclave. Using the AWS KMS APIs included in the Nitro Enclaves SDK, you can perform AWS KMS actions, such as Decrypt, GenerateDataKey, and GenerateRandom from within the enclave. For more information about using the KMS APIs with Nitro Enclaves, see the Nitro Enclaves SDK GitHub repo and the How Nitro Enclaves uses AWS KMS in the AWS Key Management Service Developer Guide.

For more information about how to use attestation with AWS KMS, see Using cryptographic attestation with AWS KMS (p. 30). If you are using a third-party external service, you must implement your own access policies and mechanisms for attestation using the attestation document and the enclave's measurements.

Where to get an enclave's measurements

An enclave's measurements includes a series of hashes and platform configuration registers (PCRs) that are unique to the enclave. An enclave has six measurements:

<table>
<thead>
<tr>
<th>PCR</th>
<th>Hash of ...</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCRO</td>
<td>Enclave image file</td>
<td>A contiguous measure of the contents of the image file, without the section data.</td>
</tr>
</tbody>
</table>
### PCR0, PCR1, and PCR2

<table>
<thead>
<tr>
<th>PCR</th>
<th>Hash of ...</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCR1</td>
<td>Linux kernel and bootstrap</td>
<td>A contiguous measurement of the kernel and boot ramfs data.</td>
</tr>
<tr>
<td>PCR2</td>
<td>Application</td>
<td>A contiguous, in-order measurement of the user applications, without the boot ramfs.</td>
</tr>
<tr>
<td>PCR3</td>
<td>IAM role assigned to the parent instance</td>
<td>A contiguous measurement of the IAM role assigned to the parent instance. Ensures that the attestation process succeeds only when the parent instance has the correct IAM role.</td>
</tr>
<tr>
<td>PCR4</td>
<td>Instance ID of the parent instance</td>
<td>A contiguous measurement of the ID of the parent instance. Ensures that the attestation process succeeds only when the parent instance has a specific instance ID.</td>
</tr>
<tr>
<td>PCR8</td>
<td>Enclave image file signing certificate</td>
<td>A measure of the signing certificate specified for the enclave image file. Ensures that the attestation process succeeds only when the enclave was booted from an enclave image file signed by a specific certificate.</td>
</tr>
</tbody>
</table>

Some of the measures are exposed when the enclave image file is built, while others need to be manually generated based on information about the parent instance.

#### Topics

- [PCR0, PCR1, and PCR2 (p. 27)]
- [PCR3 (p. 28)]
- [PCR4 (p. 28)]
- [PCR8 (p. 28)]

### PCR0, PCR1, and PCR2

PCR0, PCR1, and PCR2 are exposed when the enclave image file (.eif) is built. In other words, they are provided as part of the output of the `nitro-cli build-enclave (p. 68)` command.

For example, when building the enclave image file for the `hello-world` sample application, the output includes the following:

```
Enclave Image successfully created.
{
    "Measurements": {
        "HashAlgorithm": "Sha384 ( ... )",
        "PCR0": "7fb5c55bc22ecb68ed99a13d7122abfc866b926a79d5379bc58b9445c84217f59cfdd36c08b2c79552928702efe23e4",
        "PCR1": "235c9e6050abf6b993c915505f3220e2d82b51aff830ad14cbec2e0e1b60b4ae748d311c6634e644c9e718acc4286",
        "PCR2": "0f0ac32c300289e872e6ac4d19b05ac4a9b020c9829564ff3978610750ce6a86f7edff24e3c0a4a4452ff8a9ea79d"
    }
```

---

27
To further strengthen the security posture of the enclave, you can create and attach an instance profile to the parent instance. After you create the instance profile and associate an IAM role with it, you can generate a SHA384 hash based on the Amazon resource name (ARN) of the IAM role that's associated with the instance profile. You can then use the hash as PCR3 in the condition keys for your AWS KMS key policies. Doing this ensures that only enclaves running on an instance that has the correct IAM role can perform specific AWS KMS actions against a KMS key. For more information, see Using instance profiles in the IAM User Guide.

You can generate the hash using any tool that is capable of converting a string to a SHA384 hash.

For example, the following command generates a SHA384 hash for an IAM role with an ARN of arn:aws:iam::123456789012:role/Webserver.

```
Note
In this example, the hash is padded with 48 null (\0) characters.
```

```
$ ROLEARN="arn:aws:iam::123456789012:role/Webserver"; \
    python -c"import hashlib, sys; \n    h=hashlib.sha384(); h.update(b'\0'*48); \n    h.update($ROLEARN\".encode('utf-8')); \n    print(h.hexdigest())"
```

Example output

```
$ 78fce75db17cd4e0a3fb8dad3ad128ca5e77edbb2b2c7f75329dcd99aa5f6ef4fc1f1a452e315b9e98f9e312e6921e6
```

**PCR4**

PCR4 is based on a SHA384 of the instance ID of the parent instance. Therefore, you can generate the PCR after you have launched the parent instance.

You can generate this hash using any tool that is capable of converting a string to a SHA384 hash.

For example, the following command generates a SHA384 hash for a parent instance with an instance ID of i-1234567890abcdef0.

```
Note
In this example, the hash is padded with 48 null (\0) characters.
```

```
$ INSTANCE_ID="i-1234567890abcdef0"; \
    python -c"import hashlib, sys; \n    h=hashlib.sha384(); h.update(b'\0'*48); \n    h.update($INSTANCE_ID\".encode('utf-8')); \n    print(h.hexdigest())"
```

Example output

```
$ 08f996bd43e047a9eb51e7f548bfee7e164fd7dc8f65541f2ac09d6545ac812719327281c401a67a10fcba87ae79ce0
```

**PCR8**

You can also sign the enclave image file using your signing certificate and your private key.
PCR8 is exposed only when building a signed enclave image file (.eif). In other words it is provided as part of the output of the `nitro-cli build-enclave (p. 68)` command when the `--private-key` and `--signing-certificate` options are specified. Doing this creates a signed enclave image file.

Using PCR8 ensures that only enclaves booted from an enclave image file signed by a specific certificate can perform specific AWS KMS actions against a KMS key. It also enables you to build more flexible condition keys that remain effective even if the enclave image or parent instance is changed. We recommend that you use PCR3 and PCR8 together for the best flexibility.

You can use OpenSSL to generate a private key and signing certificate that can be used to sign an enclave image file.

### To generate a private key and signing certificate

1. Generate the private key.

   ```bash
   $ openssl ecparam -name secp384r1 -genkey -out key_name.pem
   ```

   This command generates the private key needed for the `--private-key` option.

2. Generate a certificate signing request (CSR). You can customize the request information if needed.

   ```bash
   $ openssl req -new -key key_name.pem -sha384 -nodes -subj "/CN=AWS/C=US/ST=WA/L=Seattle/O=Amazon/OU=AWS" -out csr.pem
   ```

3. Generate a certificate based on the CSR. Specify the CSR, the private key, a name for the certificate, and the number of days for which the certificate is to remain valid.

   ```bash
   $ openssl x509 -req -days 20 -in csr.pem -out certificate.pem -sha384 -signkey key_name.pem
   ```

   This command generates the signing certificate needed for the `--signing-certificate` option.

For example, when building the enclave image file for the `hello-world` sample application and specifying a private key and signing certificate, the output includes the following.

```bash
$ nitro-cli build-enclave --docker-uri hello-world:latest --output-file hello-signed.eif --private-key key_name.pem --signing-certificate certificate.pem
```

Enclave Image successfully created.

```json

```
How to get an enclave's attestation document

An enclave's attestation document is generated by the Nitro Hypervisor. You can request an enclave's attestation document from inside the enclave only, using the `get-attestation-document` API, which is included in the Nitro Enclaves SDK. For more information, see AWS Nitro Enclaves SDK Github repository.

**Important**
Enclaves booted in debug mode generate attestation documents with PCRs that are made up entirely of zeros (00000000000000000000000000000000). These attestation documents cannot be used for cryptographic attestation.

Using cryptographic attestation with AWS KMS

This section explains how to set up attestation to work with AWS Key Management Service. AWS KMS integrates with Nitro Enclaves to provide built-in attestation support.

**Secret data preparation**

Before using Nitro Enclaves with AWS KMS, it is important that you encrypt your sensitive data before sending it to the parent instance or the enclave. This section provides an overview of the steps needed to prepare your sensitive data for processing inside the enclave.

1. Create a AWS KMS key. For more information, see Creating Keys in the AWS Key Management Service Developer Guide.
2. Generate a plaintext and encrypted data key using the KMS key. For more information, see generate-data-key in the AWS KMS AWS CLI Command Reference.
3. Encrypt the secret data under the KMS key using the plaintext data key and a client-side cryptographic library, such as the AWS Encryption SDK. For more information, see Encrypt data with a data key in the AWS Key Management Service Developer Guide. You must to modify the KMS key policy to grant the IAM principal that you're using in your client permission to call the GenerateDataKey API action.
4. Upload the encrypted secret data and the encrypted data key to a storage location, such as Amazon S3. If you’re using the AWS Encryption SDK, the encrypted data key is automatically included in the header of the encrypted message.

**KMS key preparation**

After you have created your KMS key and you have encrypted your sensitive data under it, you need to ensure that only the enclave can access it to decrypt the encrypted data.

AWS KMS enables you to create KMS key policies with condition keys that are based on an enclave's measurements. For more information about using condition keys in KMS key policies, see AWS KMS condition keys for AWS Nitro Enclaves in the AWS Key Management Service Developer Guide.

The Nitro Enclaves SDK includes some APIs (kms-decrypt, kms-generate-data-key, and kms-generate-random) that integrate with AWS KMS. When these APIs are called against a specific key, the enclave's attestation document, which includes its measurements, is attached to the request. AWS KMS receives the request and validates the measurements in the provided attestation document against the measurements specified in the condition keys of the KMS key policy. It uses this information to determine whether the enclave should be granted permission to perform the requested action using the requested KMS key.
To prepare AWS KMS for attestation you must have the enclave's measurements. When you have the measurements, you can create a KMS key policy that includes condition keys that are based on those measurements.

AWS KMS provides `kms:RecipientAttestation:ImageSha384` and `kms:RecipientAttestation:PCR` condition keys that enable you to create attestation-based condition keys for KMS key policies. These policies ensure that AWS KMS only allows operations using the KMS key if the enclave provides a signed attestation document that contains measurements that match the measurements specified in the KMS key policy's condition keys. For more information about the condition keys, see `kms:RecipientAttestation:ImageSha384` and `kms:RecipientAttestation:PCR` in the AWS Key Management Service Developer Guide.

For example, the following KMS key policy allows enclaves running on instances that have the data-processing instance profile to use the KMS key for the Decrypt, GenerateDataKey, and GenerateRandom actions. The condition key allows the operation only when measurements in the attestation document in the request matches the measurements in the condition. If the request doesn't include an attestation document, the role doesn't have permission to call the operation because this condition cannot be satisfied.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable enclave data processing",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/data-processing"
            },
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey",
                "kms:GenerateRandom"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "kms:RecipientAttestation:ImageSha384": "EXAMPLE8abcdef7abcdef6abcdef5abcdef4abcdef3abcdef2abcdef1abcdef1abcdef0abcdef1abcdEXAMPLE",
                    "kms:RecipientAttestation:PCR0": "EXAMPLEbc2ecbb68ed99a13d7122abf0666b926a79d5379bc58b9445c84217f59cf0EXAMPLE",
                    "kms:RecipientAttestation:PCR1": "EXAMPLEf050abf6b993c915505f3220e2d82b51af830ad14cbecc2eccc1f0f4aef749cEXAMPLE",
                    "kms:RecipientAttestation:PCR2": "EXAMPLEc300289e872e6ac4d19bbb5ac4a9b020c98295643ff5978610750ce6a86f76EXAMPLE",
                    "kms:RecipientAttestation:PCR3": "EXAMPLEe11de9baee597508183477f097ae385d4a2c885aa655432365b53b812694e25EXAMPLE",
                    "kms:RecipientAttestation:PCR4": "EXAMPLE6b9b3d89a53b13f5dfd14a1049ec0b80a9ae4b159ad6e479e9f7f512f33e83EXAMPLE",
                    "kms:RecipientAttestation:PCR8": "EXAMPLE34a884328944cd806127c7784677ab60a154249fd21546a217299ccfabe1bfEXAMPLE"
                }
            }
        }
    ]
}
```
Getting started with cryptographic attestation: KMS Tool sample application

The AWS Nitro Enclaves SDK ships with a sample application, called KMS Tool, that demonstrates the cryptographic attestation process. The KMS Tool sample application is supported on both Windows and Linux parent instances.

KMS Tool includes two applications:

- **kmstool-instance**—An application that runs on the parent instance. It connects to kmstool-enclave (over the vsock socket), passes credentials to the enclave, along with a base64-encoded message for decryption.
- **kmstool-enclave**—An application that runs in an enclave. It uses the Nitro Enclaves SDK to call AWS KMS in order to decrypt the base64-encoded message received from the application running on the parent instance.

For instructions on how to set up and use the KMS Tool sample application, see the tutorial in the AWS Nitro Enclaves SDK Github repository. This tutorial shows you how to:

- Launch an enclave-enabled parent instance.
- Build a Docker image from a Docker file.
- Convert a Docker image to an enclave image file.
- Create an AWS KMS key.
- Add attestation-based condition keys to a KMS key policy.
- Create an enclave using an enclave image file.

**Tip**
The tutorial also discusses some best practices for preparing your enclave and KMS key for attestation. You can use this sample application as a reference for building your own enclave applications and for preparing your enclave and KMS keys for attestation.
Nitro Enclaves application development

An enclave application is an application that is designed and developed to run inside the isolated enclave environment. An enclave application typically consists of at least two components:

- An application that runs on the parent instance
- An application that runs inside the enclave

Due to the isolated environment of the enclave, the only channel of communication between applications that are running on the instance and applications that are running in the enclave is the vsock socket.

Topics
- Nitro Enclaves Developer AMI (p. 33)
- Nitro Enclaves SDK (p. 33)
- Nitro Enclaves application development on Linux instances (p. 33)
- Nitro Enclaves Application development on Windows instances (p. 35)

Nitro Enclaves Developer AMI

AWS provides a Nitro Enclaves Developer AMI that contains the tools and components needed to develop enclave applications and to build enclave image files. It also contains samples applications, such as hello-enclave, vsock_sample and kmstool, to demonstrate how to use and develop your own enclave applications. For more information, see AWS Nitro Enclaves Developer AMI.

Nitro Enclaves SDK

The Nitro Enclaves SDK is a set of open-source libraries that you can use to develop your enclave applications. The SDKs also integrate with AWS KMS and provide built-in support for attestation and cryptographic operations. For more information about the SDKs and how to use them, see the Nitro Enclaves SDK Github repository.

Nitro Enclaves application development on Linux instances

This section provides information for Nitro Enclaves application development on Linux instances.

Getting started with the vsock: Vsock tutorial

Note
The vsock sample application is supported on Linux instances only.
The vsock sample application is a simple client-server application that exchanges information between the parent instance and the enclave using the vsock socket.

The vsock sample application includes a client application and a server application. The client application runs on the parent instance, while the server application runs in the enclave. The client application sends a simple text message over the vsock to the server application. The server application listens to the vsock and prints the message to the console.

The vsock sample application is available in both Rust and Python. This tutorial shows you how to set up and run the Rust vsock sample application. For more information about setting up and running the Python vsock sample application, see the AWS Nitro Enclaves samples GitHub repository.

**Note**
The application source is also freely available from the AWS Nitro Enclaves samples GitHub repository. You can use the application source as a reference for building your own applications.

**Prerequisites**

To compile the Rust vsock sample application, you must have Cargo, Rust's build system and package manager installed, and you must add the x86_64-unknown-linux-musl target. To install and configure Rust, use the following commands.

```
$ curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh
```

You must disconnect from the instance and then reconnect before running the following commands.

```
$ rustup target add x86_64-unknown-linux-musl

$ sudo yum -y install gcc
```

**To try the vsock sample application**

1. Download the application source and navigate into the directory.

```
$ git clone https://github.com/aws/aws-nitro-enclaves-samples.git

$ cd aws-nitro-enclaves-samples/vsock_sample/rs
```

2. Compile the application code using Cargo.

```
$ cargo build --target=x86_64-unknown-linux-musl --release
```

The compiled binary is located in `vsock_sample/rs/target/x86_64-unknown-linux-musl/release/vsock-sample`.

3. Navigate two levels up.

```
$ cd ../..
```

4. Create a new file named Dockerfile and add the following.

```
# start the Docker image from the Alpine Linux distribution
FROM alpine:latest
# copy the vsock-sample binary to the Docker file
COPY vsock_sample/rs/target/x86_64-unknown-linux-musl/release/vsock-sample .
```
# start the server application inside the enclave
CMD ./vsock-sample server --port 5005

5. Convert the Docker file to an enclave image file.

```bash
$ nitro-cli build-enclave --docker-dir ./ --docker-uri vsock-sample-server --output-file vsock_sample.eif
```

6. Boot the enclave using the enclave image file. You need to access the enclave console to see the server application output, so you must include the `--debug-mode` option.

```bash
$ nitro-cli run-enclave --eif-path vsock_sample.eif --cpu-count 2 --enclave-cid 6 --memory 256 --debug-mode
```

Make a note of the enclave ID, because you'll need this to connect to the enclave console.

7. Open the enclave console. The console provides a view of what's happening on the server side of the application.

```bash
$ nitro-cli console --enclave-id enclave_id
```

8. Open an SSH terminal window for the parent instance. You'll use this terminal to run the client application.

9. In the parent instance terminal, run the client application. When you start the client application, it automatically sends some text over the vsock to the server application running in the enclave. Watch the enclave console terminal for the output.

```bash
$ ./aws-nitro-enclaves-samples/vsock_sample/rs/target/x86_64-unknown-linux-musl/release/vsock-sample client --cid 6 --port 5005
```

10. When the server application receives the text over the vsock, it prints the text to the console.

```bash
$ [ 0.127079] Freeing unused kernel memory: 476K
[ 0.127631] nsm: loading out-of-tree module taints kernel.
[ 0.128055] nsm: module verification failed: signature and/or required key missing - tainting kernel
[ 0.154010] random: vsock-sample: uninitialized urandom read (16 bytes read)
Hello, world!
```

Now that you understand how the sample application works, download and customize the source code to suit your use case.

---

**Nitro Enclaves Application development on Windows instances**

This section provides information for Nitro Enclaves application development on Windows instances.

**Topics**
- Considerations for using Nitro Enclaves on a Windows parent instance (p. 36)
- Nitro Enclaves for Windows release notes (p. 36)
- Subscribe to notifications of new versions (p. 38)
- Working with the vsock socket in Windows (p. 39)
Considerations for using Nitro Enclaves on a Windows parent instance

The EC2 parent instance and the enclaves operate as separate virtual machines. This means that each of them (the parent instance and all of its enclaves) must run its own operating system. The parent instance, supports both Linux and Windows (2012 R2 and later) operating systems. However, the enclaves support only operating systems that support the Linux boot protocol. This means that even if you have a Windows parent instance, you must run a Linux environment inside your enclaves.

This also means that you must use a Linux-based instance to build your enclave image file (.eif).

Keep the following in mind when using a Windows parent instance.

- Only Windows 2012 R2 and later is supported on the parent instance.
- You must run a Linux-based environment inside the enclave.
- The Hello enclaves sample application is supported on Windows parent instances, but the enclave image file (.eif) must be built on a Linux instance. For more information, see Getting started: Hello enclave (p. 7).
- The KMS Tool sample application is supported on Windows parent instances, but the enclave image file (.eif) must be built on a Linux instance. For more information, see Getting started with cryptographic attestation: KMS Tool sample application (p. 32).
- On Windows, the vssock uses the standard Windows sockets (Winsock2) API. For more information, see Working with the vssock socket in Windows (p. 39).
- AWS Certificate Manager for Nitro Enclaves is not supported with Windows parent instances.
- To use the AWS Nitro Enclaves CLI software on your parent instance, you must install the AWSNitroEnclavesWindows package using AWS Systems Manager Distributor. For more information, see Installing the Nitro Enclaves CLI on Windows (p. 67).
- The nitro-cli build-enclave command is not supported on Windows parent instances. For more information, see nitro-cli build-enclave (p. 68).

Nitro Enclaves for Windows release notes

This section describes Nitro Enclaves (for Windows) features, improvements, and bug fixes by release date.
### Release Notes

<table>
<thead>
<tr>
<th>Release date</th>
<th>Version</th>
<th>Updates and Bug Fixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 27, 2023</td>
<td>1.2.1</td>
<td>The release fixed a bug related to terminating multiple enclaves.</td>
</tr>
</tbody>
</table>
| May 4, 2022 | 1.2.0 | The release added the following commands, arguments, and output for Nitro CLI:  
- Added `pcr` and `describe-eif` commands.  
- Added `--enclave-name` argument for `run-enclave`, `console`, and `terminate-enclave` commands.  
- Added `--disconnect-timeout` argument for `console` command.  
- Added `--config` argument and `--attach-console` flag to `run-enclave` command  
- Updated `describe-enclaves` and `run-enclave` commands to display `EnclaveName`.  
- Added `--metadata` flag to `describe-enclaves` command.  
The release added the following bug fixes and enhancements:  
- Improved Nitro CLI error messages.  
- Fixed bugs in `vsock select()` when it blocks or returns certain calls.  
- Fixed bug in `vsock shutdown()` on nonblocking sockets, which can result in connection reset errors. |
| July 27, 2021 | 1.1.0 | The release added the following bug fixes and enhancements:  
- Improved `vsock` error codes and Nitro CLI error messages.  
- Improved `vsock` driver stability when enabling and disabling the `vsock` device.  
- Improved Nitro CLI efficiency during failed enclave startups. |
### Subscribe to notifications of new versions

Amazon SNS can notify you when new versions of Nitro Enclaves for Windows are released. Use one of the following procedures to subscribe to these notifications.

**Amazon SNS console**

**To subscribe to notifications using the Amazon SNS console**

2. In the navigation bar, change the Region to **US West (Oregon)**, if necessary. You must select this Region because the SNS notifications that you are subscribing to are in this Region.
3. In the navigation pane, choose **Subscriptions**.
4. Choose **Create subscription**.
5. In the **Create subscription** dialog box, do the following:
   b. For **Protocol**, choose **Email**.
   c. For **Endpoint**, type an email address that you can use to receive the notifications.
   d. Choose **Create subscription**.
6. You'll receive a confirmation email. Open the email and follow the directions to complete your subscription.

**AWS Tools for PowerShell Core**

**To subscribe to notifications using the Tools for Windows PowerShell**

Use the following command.

```
```

**AWS Command Line Interface**

**To subscribe to notifications using the AWS CLI**

Use the following command.

```
C:\> aws sns subscribe \
  --protocol email \
```
If you no longer want to receive these notifications, use the following procedure to unsubscribe.

**To unsubscribe to notifications using the Amazon SNS console**

2. In the navigation bar, change the Region to US West (Oregon).
3. In the navigation pane, choose Subscriptions.
4. Select the check box for the subscription and then choose Delete. When prompted for confirmation, choose Delete.

**Working with the vsck socket in Windows**

This topic provides information that is specific to working with the vsck socket on Windows instances.

**Topics**

- Terminology (p. 39)
- AWS vsck socket implementation (p. 39)
- Using the Winsock2 functions with vsck sockets (p. 40)
- Unsupported Winsock2 functions (p. 43)
- Known issues (p. 43)

**Terminology**

**Service Provider Interface**

The Service Provider Interface (SPI) is a library registered with the Windows Sockets 2 (Winsock2) API to support a new address family for the vsck socket. The vsck SPI is available in a 64-bit version only. Only 64-bit applications can use vsck sockets.

**Port**

The port component of an address. This is a 32-bit unsigned value.

**Local address**

The address of a vsck socket on the host on which the application is running. The address includes a context identifier (CID) and a port. The CID and port value pairs are concatenated with a ‘.’ when written as a string. For example, a host with CID of 3 that listens on port 1234 has a listening address of 3.1234.

**Peer**

A host that this host is communicating with over the vsck socket.

**Remote address**

The address of the vsck socket of the peer. The address includes a context identifier (CID) and a port. The CID and port value pairs are concatenated with a ‘.’ when written as a string. For example, a host with CID of 3 that listens on port 1234 has a listening address of 3.1234.

**AWS vsck socket implementation**

The following are considerations for vsck socket implementation using Winsock2.
Topics

• Build-time dependencies (p. 40)
• Runtime (p. 40)
• Loopback support (p. 40)

Build-time dependencies

Some value definitions are required to create and interact with vsock sockets. These include
the definitions of AF_VSOCK, sockaddr_vm, and some reserved values for CIDs and ports. It is
recommended that you include these definitions by including the VirtioVsock.h header in your code.
For more information, about the header, see the Nitro Enclaves SDK Github repository.

Runtime

To create a Winsock2 socket with the AF_VSOCK address family, the vsock SPI must be registered with
Winsock2. The vsock SPI is registered during the AWS Nitro Enclaves installation. Currently, the vsock
SPI is available in a 64-bit version and supports only 64-bit applications. For more information about
installing AWS Nitro Enclaves on a Windows instance, see Install AWS Nitro Enclaves CLI (p. 67).

Loopback support

Loopback is not supported with vsock sockets. Attempts to connect() to a CID that belongs to the
same host could result in an error.

Using the Winsock2 functions with vsock sockets

This section highlights differences between the Winsock2 functions and the AWS implementation for the
vsock SPI.

Note

Functions not listed below follow the Winsock2 implementation and behave as described in the
Winsock2 API documentation without any AWS specific nuances. For a list of the unsupported
Winsock2 functions, see Unsupported Winsock2 functions (p. 43).

Topics

• WSAAccept()/accept() (p. 41)
• WSAAddressToString() (p. 41)
• WSABind()/bind() (p. 41)
• WSAConnect()/connect() (p. 41)
• WSAEventSelect() (p. 41)
• WSAGetPeerName() (p. 41)
• WSAGetSockName() (p. 41)
• WSAGetSockOpt()/getsockopt() (p. 41)
• WSAlctl()/ioctlsocket() (p. 42)
• WSAListen()/listen() (p. 42)
• WSASend()/send() (p. 42)
• WSASetSockOpt()/setsockopt() (p. 42)
• WSASocket()/socket() (p. 42)
• WSAStringToAddress() (p. 42)
• WSARecv()/recv() (p. 42)
**WSAAccept() / accept()**

If a vsock transport reset or device disable event occurs after receiving a connection request, but before `accept()` is called, `accept()` returns an invalid socket and `WSAGetLastError()` returns the value `WSAECONNRESET`.

**WSAAddressToString()**

Converts `sockaddr_vm` to a string in the `CID.Port` format.

**WSABind() / bind()**

To create a connection using a specific local port, you must call `bind()` with a valid local CID and the desired local port before calling `connect()`. An enclave-enabled Amazon EC2 instance is always assigned local CID of 3. A socket bound to `SOCKADDR_VM_CID_ANY` can only be used for listening and cannot be used with `connect()`. `SO_REUSEADDR` and `SO_EXCLUSIVEADDRUSE` are not configurable. AWS vsock sockets behave as if `SO_EXCLUSIVEADDRUSE` is enabled for all sockets. That is, if any socket is bound to a local CID.port pair, no other socket can bind to that same local CID.port except as an `accept()` from a listening socket that is bound to `SOCKADDR_VM_CID_ANY.port` or CID.port. Additionally, sockets cannot be bound to `SOCKADDR_VM_CID_ANY.port` when any other socket on the host is bound to an address with the same local port value and any CID.

**WSAConnect() / connect()**

Outgoing connection requests have a non-configurable 1 second timeout before the peer responds with a connection acceptance packet. When using `WSAConnect()`, caller data and callee data are not supported. Specifying caller data results in an error and specifying callee data returns a length of 0. QOS options are also not supported and return an error if specified.

**WSAEventSelect()**

`FD_OOB`, `FD_QOS`, `FD_GROUP_QOS`, `FD_ROUTING_INTERFACE_CHANGE`, and `FD_ADDRESS_LIST_CHANGE` will never be signaled in the current implementation. However, they do not return an error if specified.

**WSAGetPeerName()**

Gets the peer's socket address as a `sockaddr_vm`.

**WSAGetSockName()**

Gets the local socket address as a `sockaddr_vm`.

**WSAGetSockOpt() / getsockopt()**

Only the following `optname` parameters are supported:

- `SO_LINGER`
- `SO_DONTLINGER`
- `SO_RCVBUF`
- `SO_ACCEPTCONN`
- `SO_PROTOCOL_INFOW`
- `SO_TYPE`

For more information, see [getsockopt function](https://docs.microsoft.com/en-us/windows/win32/api/winsock2/) on the Windows app developer documentation website.
**WSAIoctl()/ioctlsocket()**

Only the following `ioctl`s are supported:

- FIONBIO
- FIONREAD

For more information, see [ioctlsocket function](https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2_ioctlsocket) on the Windows app developer documentation website.

**WSAListen()/listen()**

Setting a backlog size of 0 or less sets the backlog size to 0. Setting the backlog size to a value greater than the implementation-specific maximum backlog size, which is currently 2048, sets the backlog size to the implementation-specific maximum backlog size. Reducing the backlog size while connection requests exist on a listening socket rejects some of the connection requests until the number of connections is equal to, or less than, the new backlog size.

**WSASend()/send()**

No flags are supported for these functions. The flag values must be set to 0. If you specify a different value, an error is returned.

**WSASetSockOpt()/setsockopt()**

This function follows the Winsock2 implementation. However, only the following options are supported:

- SO_LINGER
- SO_DONTLINGER
- SO_RCVBUF

SO_RCVBUF has a minimum value of 4096 bytes and a maximum value of 2 MB. Requested buffer sizes are rounded down to a power of 2, or to 2 MB if the value is greater than 2 MB. Received buffer size defaults to 256 KB.

For more information, see [ioctlsocket function](https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2_ioctlsocket) on the Windows app developer documentation website.

**WSASocket()/socket()**

This function returns a new `SOCKET`. With the AWS vsock SPI, this `SOCKET` is also a `HANDLE` that allows you to call functions, such as `ReadFile` and `WriteFile` directly on the `SOCKET`.

This function must be called with `af = AF_VSOCK` and `type = SOCK_STREAM`. Only the `WSA_FLAG_OVERLAPPED` flag is supported when calling `WSASocket()`, which allows overlapped IO on the `SOCKET` that is returned. If `socket()` is called, the `WSA_FLAG_OVERLAPPED` flag is set. For more information about overlapped creation of sockets, see [socket function](https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2_socket) on the Windows app developer documentation website.

**WSAStringToAddress()**

Converts a string in the format of CID.Port to a `sockaddr_vm`.

**WSARcv()/recv()**

Only the MSG_PEEK flag is supported for this function.
Unsupported Winsock2 functions

The following Winsock2 functions are not supported with AWS vsock sockets.

- WSACancelBlockingCall()
- WSAAsyncSelect()
- WSAGetQosByName()
- WSAJoinLeaf()
- WSARcvDisconnect()
- WSASendDisconnect()
- WSARcvFrom()
- WSASendTo()

Known issues

Some IOs cannot be canceled

When calling WSASend(), WSARcv(), or WSAIoctl() on an overlapped socket, either with lpOverlapped omitted or with lpCompletionRoutine specified, the IO cannot be canceled by the user using CancelIo or CancelIoEx. CancelIoEx returns an error with GetLastError() returning ERROR_NOT_FOUND. All IOs can be canceled by calling closesocket().
Attestation in the Nitro Enclaves world

The purpose of attestation is to prove that an enclave is a trustworthy entity, based on the code and configuration that is running within a particular enclave. The root of trust for the enclave resides within the AWS Nitro system, which provides attestation documents to the enclave.

The root of trust component for the attestation is the Nitro Hypervisor, which contains information about the enclave, such as its platform configuration registers (PCRs). The Nitro Hypervisor is able to produce an attestation document that contains details of the enclave, including the enclave signing key, a hash of the enclave image, a hash of the parent instance ID, and a hash of the ARN of the attached IAM role.

Attestation documents are signed by the AWS Nitro Attestation Public Key Infrastructure (PKI), which includes a published certificate authority that can be incorporated into any service.

The attestation document

An enclave can request an attestation document from the Nitro hypervisor that it can use to verify its identity with an external service. The attestation document that is generated by the Nitro system is encoded in Concise Binary Object Representation (CBOR), and it is signed using CBOR Object Signing and Encryption (COSE).

For more information about CBOR, see RFC 8949: Concise Binary Object Representation (CBOR). For more information about the COSE implementation, see the COSE for AWS Nitro Enclaves Github repository.

Attestation document specification

The following shows the structure of an attestation document.

```plaintext
AttestationDocument = {
  module_id: text,               ; issuing Nitro hypervisor module ID
  timestamp: uint .size 8,       ; UTC time when document was created, in milliseconds since UNIX epoch
}  
```
Attestation document validation

When you request an attestation document from the Nitro Hypervisor, you receive a binary blob that contains the signed attestation document. The signed attestation document is a CBOR-encoded, COSE-signed (using the COSE_Sign1 signature structure) object. The overall validation process includes the following steps:

1. Decode the CBOR object and map it to a COSE_Sign1 structure.
2. Extract the attestation document from the COSE_Sign1 structure.
3. Verify the certificate's chain.
4. Ensure that the attestation document is properly signed.

Attestation documents are signed by the AWS Nitro Attestation PKI, which includes a root certificate for the commercial AWS partitions. The root certificate can be downloaded from [https://aws-nitro-enclaves.amazonaws.com/AWS_NitroEnclaves_Root-G1.zip](https://aws-nitro-enclaves.amazonaws.com/AWS_NitroEnclaves_Root-G1.zip), and it can be verified using the following SHA256 checksum:

8cf60e2b2efca96c6a9e71e851d00c1b6991cc09eadbe64a6a1d1b1eb9faff7c
The root certificate is based on an AWS Certificate Manager Private Certificate Authority (AWS Private CA) private key and it has a lifetime of 30 years. The subject of the PCA has the following format.

CN=aws.nitro-enclaves, C=US, O=Amazon, OU=AWS

Topics

- COSE and CBOR (p. 46)
- Semantical validity (p. 46)
- Certificate validity (p. 47)
- Certificate chain validity (p. 47)

**COSE and CBOR**

Usually, the COSE_Sign1 signature structure is used when only one signature is going to be placed on a message. The parameters dealing with the content and the signature are placed in the protected header rather than having the separation of COSE_Sign. The structure can be encoded as either tagged or untagged, depending on the context it will be used in. A tagged COSE_Sign1 structure is identified by the CBOR tag 18.

The CBOR object that carries the body, the signature, and the information about the body and signature is called the COSE_Sign1 structure. The COSE_Sign1 structure is a CBOR array. The array includes the following fields.

```json
[
  protected:   Header,
  unprotected: Header,
  payload:     This field contains the serialized content to be signed,
  signature:   This field contains the computed signature value.
]
```

In the context of an attestation document, the array includes the following.

```plaintext
18/* COSE_Sign1 CBOR tag is 18 */
(1: -35), /* This is equivalent with {algorithm: ECDS 384} */
{}, /* We have nothing in unprotected */
$ATTESTATION_DOCUMENT_CONTENT /* Attestation Document */,
  signature /* This is the signature */
)
```


**Semantical validity**

An attestation document will always have its CA bundle in the following order.

```
[  ROOT_CERT - INTERM_1 - INTERM_2 .... - INTERM_N]
      0  1  2    ....    N - 1
```

Keep this ordering in mind, as some existing tools, such as Java's CertPath from [Java PKI API Programmer's Guide](https://docs.oracle.com/javase/8/docs/technotes/guides/security/x509/CertPath.html), might require them to be ordered differently.
To validate the certificates, start from the attestation document CA bundle and generate the required chain, where `TARGET_CERT` is the certificate in the attestation document.

```
[TARGET_CERT, INTERM_N, ..... , INTERM_2, INTERM_1, ROOT_CERT]
```

For more information about the optional fields in the attestation document, see the Nitro Enclaves Attestation Process.

## Certificate validity

For all of the certificates in the chain, you must ensure that the current date falls within the validity period specified in the certificate.

## Certificate chain validity

In general, a chain of multiple certificates might be needed, comprising a certificate of the public key owner signed by one CA, and zero or more additional certificates of CAs signed by other CAs. Such chains, called certification paths, are required because a public key user is only initialized with a limited number of assured CA public keys. Certification path validation procedures for the internet PKI are based on the algorithm supplied in X.509. Certification path processing verifies the binding between the subject distinguished name and/or subject alternative name and subject public key. The binding is limited by constraints that are specified in the certificates that comprise the path and inputs that are specified by the relying party. The basic constraints and policy constraint extensions allow the certification path processing logic to automate the decision making process.

**Note**

CRL must be disabled when doing the validation.

Using Java, starting from the root path and the generated certificate chain, the chain validation is as follows.

```java
validateCertsPath(certChain, rootCertificate) {
    /* The trust anchor is the root CA to trust */
    trustAnchors.add(rootCertificate);

    /* We need PKIX parameters to specify the trust anchors and disable the CRL validation */
    validationParameters = new PKIXParameters(trustAnchors);
    certPathValidator = CertPathValidator.getInstance(PKIX);
    validationParameters.setRevocationEnabled(false);

    /* We are ensuring that certificates are chained correctly */
    certPathValidator.validate(certPath, validationParameters);
}
```
Nitro Enclaves application: AWS Certificate Manager for Nitro Enclaves

Note

- ACM for Nitro Enclaves is available for Linux instances only. It is currently not supported on Windows instances.
- AWS Certificate Manager for Nitro Enclaves is currently not supported in Asia Pacific (Osaka) and Asia Pacific (Jakarta).

AWS Certificate Manager (ACM) for Nitro Enclaves allows you to use public and private SSL/TLS certificates with your web applications and web servers running on Amazon EC2 instances with AWS Nitro Enclaves. SSL/TLS certificates are used to secure network communications and to establish the identity of websites over the internet, as well as resources on private networks.

Previously, when running a web server on an EC2 instance, you would have created SSL certificates and stored them as plaintext on your instance. With ACM for Nitro Enclaves, you can now bind AWS Certificate Manager certificates to an enclave and use those certificates directly with your web server, without exposing the certificates in plaintext form to the parent instance and its users.

ACM for Nitro Enclaves removes the time-consuming and error-prone manual process of purchasing, uploading, and renewing SSL/TLS certificates. ACM for Nitro Enclaves creates secure private keys, distributes the certificate and its private key to your enclave, and manages certificate renewals. With ACM for Nitro Enclaves, the certificate's private key remains isolated in the enclave, preventing the instance, and its users, from accessing it.

Currently, ACM for Nitro Enclaves works with NGINX servers and Apache HTTP servers running on Amazon EC2 instances to install the certificate and seamlessly replace expiring certificates. Support for additional web servers will be added over time.

Note
ACM for Nitro Enclaves uses the standardized PKCS11 cryptographic interface between the parent instance and the enclave. Any application that supports the PKCS11 protocol can be adapted to use ACM for Nitro Enclaves for protecting certificates and keys. ACM for Nitro Enclaves also includes a “helper” p11-kit based module for using the PKCS11 protocol over the Nitro Enclaves vsock socket.

Topics
- Pricing and billing (p. 48)
- Installing and configuring ACM for Nitro Enclaves (p. 49)
- Updating ACM for Nitro Enclaves (p. 59)
- Uninstalling ACM for Nitro Enclaves (p. 59)

Pricing and billing

Public SSL/TLS certificates that you provision through ACM for Nitro Enclaves are available at no additional cost. You pay only for the AWS resources that you create to run your application, such as
Amazon EC2 instances. Private certificates are available at no additional cost per certificate when you use and pay for ACM Private CA.

Installing and configuring ACM for Nitro Enclaves

Steps

- **Step 1: Create the ACM certificate** (p. 49)
- **Step 2: Prepare the enclaves-enabled parent instance** (p. 50)
- **Step 3: Prepare the IAM role** (p. 51)
- **Step 4: Associate the role with the ACM certificate** (p. 51)
- **Step 5: Grant the role permission to access the certificate and encryption key** (p. 52)
- **Step 6: Attach the role to the instance** (p. 52)
- **Step 7: Configure the web server to use ACM for Nitro Enclaves** (p. 53)
- **Using multiple certificates** (p. 57)

Prerequisites

The user performing this configuration must have permission to use the `ec2:AssociateEnclaveCertificateIamRole`, `ec2:GetAssociatedEnclaveCertificateIamRoles`, and `ec2:DisassociateEnclaveCertificateIamRole` actions. To grant the user the required permissions, use the following IAM policy.

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "ec2: AssociateEnclaveCertificateIamRole",
        "ec2: GetAssociatedEnclaveCertificateIamRoles",
        "ec2: DisassociateEnclaveCertificateIamRole"
      ],
      "Resource": [
        "arn:aws:acm:region:account_id:certificate/*",
        "arn:aws:iam::account_id:role/*"
      ],
      "Effect": "Allow"
    }
  ]
}
```

**Step 1: Create the ACM certificate**

Create the AWS Certificate Manager (ACM) certificate that you want use with your NGINX or Apache HTTP server. ACM for Nitro Enclaves supports both private and public certificates. For more information about creating a certificate, see the following resources in the *AWS Certificate Manager User Guide*.

- **Requesting a Public Certificate**
- **Requesting a Private Certificate**

After you have created the certificate, make a note of the certificate ARN; as you'll need it later.
Step 2: Prepare the enclaves-enabled parent instance

Launch the enclave enabled instance (p. 14) that you will use as the parent instance. You can use either the ACM for Nitro Enclaves AMI from AWS Marketplace, or you can install ACM for Nitro Enclaves and the web server using RPM packages.

Tip
After you launch the instance, make a note of the instance ID, as you’ll need it later.

Option 1: Using ACM for Nitro Enclaves AMI

To launch an instance using the ACM for Nitro Enclaves AMI from AWS Marketplace

1. Open the ACM for Nitro Enclaves page in the AWS Marketplace.
2. Find the ACM for Nitro Enclaves AMI for your Region, and note the AMI ID. You need the AMI ID for the next step.
3. Launch the instance using the AMI from the AWS Marketplace and enable it for Nitro Enclaves using the following command.

```
aws ec2 run-instances --image-id ami_id --count 1 --instance-type supported_instance_type --key-name your_key_pair --enclave-options 'Enabled=true'
```

Option 2: Using RPM packages

To install ACM for Nitro Enclaves from the Amazon Linux Extras repository

1. Connect to the instance.
2. Enable the aws-nitro-enclaves-cli topic in the Amazon Linux Extras library.

```
sudo amazon-linux-extras enable aws-nitro-enclaves-cli
```
3. Install your preferred web server. Do one of the following:

- **NGINX**

  Enable the nginx1 topic in the Amazon Linux Extras library and install NGINX from the Amazon Linux Extras library.

  ```
sudo amazon-linux-extras enable nginx1
```

  ```
sudo amazon-linux-extras install nginx1 -y
```

- **Apache**

  Install and configure the Apache HTTP server with SSL/TLS support.

  ```
sudo yum -y install httpd mod_ssl
```
4. Install ACM for Nitro Enclaves from the Amazon Linux Extras library.

```
sudo yum install aws-nitro-enclaves-acm -y
```
Step 3: Prepare the IAM role

To grant the instance permission to use the ACM certificate, you must create an IAM role with the required permissions. The IAM role is later attached to the instance and the ACM certificate.

Create a JSON file named `acm-role` and add the following policy statement.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "ec2.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

Use the `create-role` command to create a role named `acm-role`, and specify the path to the JSON policy file.

```
$ aws iam create-role --role-name acm-role --assume-role-policy-document file://acm-role.json
```

After you have created the role, make a note of the role ARN, as you’ll need it in the next step.

Step 4: Associate the role with the ACM certificate

Attach the IAM role that you created in the previous step to the ACM certificate. To do this, use the `associate-enclave-certificate-iam-role` command, and specify the ARN of the role to attach, and the ARN of the certificate to attach it to.

```
$ aws ec2 --region region associate-enclave-certificate-iam-role --certificate-arn certificate_ARN --role-arn role_ARN
```

For example

```
$ aws ec2 --region us-east-1 associate-enclave-certificate-iam-role --certificate-arn arn:aws:acm:us-east-1:123456789012:certificate/d4c3b2a1-e5d0-4d51-95d9-1927fEXAMPLE --role-arn arn:aws:iam::123456789012:role/acm-role
```

Example output

```
{
   "CertificateS3BucketName": "aws-ec2-enclave-certificate-us-east-1",
   "EncryptionKmsKeyId": "a1b2c3d4-354d-4e51-9190-b12ebEXAMPLE"
}
```

After running the command, make a note of `CertificateS3BucketName` and `EncryptionKmsKeyId`, as you’ll need them for the next step.
Step 5: Grant the role permission to access the certificate and encryption key

You must now grant the IAM role (acm-role) permission to do the following:

- Retrieve the ACM certificate from the Amazon S3 bucket returned in the previous step
- Perform kms:Decrypt using the AWS KMS key returned in the previous step
- Retrieve information about itself, including its path, GUID, and ARN.

Create a JSON file named `acm-role-policies.json`, add the following policy statement, and specify the values of CertificateS3BucketName and EncryptionKmsKeyId from the previous step.

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": [
            "s3:GetObject"
         ],
         "Resource": ["arn:aws:s3:::CertificateS3BucketName/*"]
      },
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": [
            "kms:Decrypt"
         ],
         "Resource": "arn:aws:kms:region:::key/EncryptionKmsKeyId"
      },
      {
         "Effect": "Allow",
         "Action": "iam:GetRole",
         "Resource": "arn:aws:iam::123456789012:role/acm-role"
      }
   ]
}
```

Use the `put-role-policy` command to add the additional policies to the acm-role role, and specify the path to the JSON policy file.

```
$ aws iam put-role-policy --role-name acm-role --policy-name acm-role-policy --policy-document file://acm-role-policies.json
```

Step 6: Attach the role to the instance

You must attach the IAM role to the instance to give it permission to use the certificate.

Create a new instance profile named acm-instance-profile using the `create-instance-profile` command.

```
$ aws iam create-instance-profile --instance-profile-name acm-instance-profile
```

Example output

```
{}
```
Add the acm-role that you created earlier to the acm-instance-profile that you just created. Use the `add-role-to-instance-profile` command.

```
sudo mv /etc/nitro_enclaves/acm.example.yaml /etc/nitro_enclaves/acm.yaml
```

Associate the instance profile with the instance that you launched previously. Use the `associate-iam-instance-profile` command and specify the instance profile to attach and the instance to attach it to.

```
sudo mv /etc/nitro_enclaves/acm.example.yaml /etc/nitro_enclaves/acm.yaml
```

**Step 7: Configure the web server to use ACM for Nitro Enclaves**

Configure the NGINX or Apache HTTP web server to use the ACM certificate. Choose the correct procedure depending on the web server you’re using.

**NGINX**

To configure NGINX

1. SSH into the instance that you launched previously.
2. Nitro Enclaves ships with a sample ACM for Nitro Enclaves configuration file that you can use as a starting point for your own configuration. To use the sample configuration file, rename the configuration file from `/etc/nitro_enclaves/acm.example.yaml` to `/etc/nitro_enclaves/acm.yaml`.

```
$ sudo mv /etc/nitro_enclaves/acm.example.yaml /etc/nitro_enclaves/acm.yaml
```
3. Specify the ARN of the certificate that you associated with the IAM role that is attached to the parent instance. Using your preferred text editor, open `/etc/nitro_enclaves/acm.yaml`. In the Acm section, for certificate_arn, specify the ARN of the certificate. Save and close the file.

4. Configure NGINX to use the pkcs11 SSL engine by setting the top-level `ssl_engine` directive.

Using your preferred text editor, open `/etc/nginx/nginx.conf`. Add the following line below `pid /run/nginx.pid;`:

```yaml
ssl_engine pkcs11;
```

Example

```yaml
# For more information on configuration, see:
#   * Official English Documentation: http://nginx.org/en/docs/

user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

ssl_engine pkcs11;
```

5. Enable the TLS server and configure the server to use your certificate.

In `/etc/nginx/nginx.conf`, scroll to the bottom of the file and do the following:

- Uncomment all of the lines below `Settings for a TLS enabled server`.
- In the first block, for `server_name`, specify the host name, or the common name (CN), that you specified when you created the certificate.
- In the second block, remove the following lines.

```yaml
ssl_certificate "/etc/pki/nginx/server.crt";
ssl_certificate_key "/etc/pki/nginx/private/server.key";
ssl_ciphers PROFILE=SYSTEM;
```

And add the following line.

```yaml
ssl_protocols TLSv1.2;
```

- Add the following as a new block below the second block.

```yaml
# Set this to the stanza path configured in /etc/nitro_enclaves/acm.yaml
include "etc/pki/nginx/nginx-acm.conf";
```

The completed section should appear as follows.

```yaml
# Settings for a TLS enabled server.

server {
    listen 443 ssl http2;
    listen [::]:443 ssl http2;
    server_name example.com;
    root /usr/share/nginx/html;
    ssl_protocols TLSv1.2;
    
    # Set this to the stanza path configured in /etc/nitro_enclaves/acm.yaml
    include "etc/pki/nginx/nginx-acm.conf";

    # Settings for a TLS enabled server.
    
    server {
        listen 443 ssl http2;
        listen [::]:443 ssl http2;
        server_name example.com;
        root /usr/share/nginx/html;
        
        ssl_protocols TLSv1.2;
    }
```
Step 7: Configure the web server to use ACM for Nitro Enclaves

ssl_session_cache shared:SSL:1m;
ssl_session_timeout 10m;
ssl_prefer_server_ciphers on;

# Set this to the stanza path configured in /etc/nitro_enclaves/acm.yaml
include "/etc/pki/nginx/nginx-acm.conf";

# Load configuration files for the default server block.
include /etc/nginx/default.d/*.conf;

error_page 404 /404.html;
    location = /40x.html {
    }
error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }

6. Start the ACM for Nitro Enclaves service and ensure that it starts automatically at instance boot.

$ sudo systemctl start nitro-enclaves-acm.service

$ sudo systemctl enable nitro-enclaves-acm

7. Test that the ACM for Nitro Enclaves is working as expected.

If you used a public certificate, use the following command.

$ curl https://host_name_or_IP

If you used a private certificate, you must add the host name to /etc/hosts in the following format: 127.0.0.1 host_name, for example 127.0.0.1 example.com. And you must specify the certificate chain to use to validate the certificate. For more information about generating the certificate chain for your certificate, see Exporting a Private Certificate in the AWS Certificate Manager User Guide.

$ curl --cacert path_to_pem_file https://host_name_or_IP

A successful test displays the NGINX index.htm. The ACM for Nitro Enclaves service continuously polls and fetches the ACM certificate data and updates NGINX accordingly. It does this by generating an NGINX config snippet and including it in the main nginx.conf.

If you renew the ACM certificate by running acm renew-certificate, the ACM for Nitro Enclaves automatically reconfigures the enclave and the NGINX web server. You can use the following command to check the log for update announcements and to diagnose possible issues.

$ journalctl -u nitro-enclaves-acm.service

If you encounter any unexpected errors, you can also check the NGINX service log for more details.

$ journalctl -u nginx.service
Apache

To configure Apache HTTP server

1. SSH into the instance that you launched previously.
2. Nitro Enclaves ships with a sample ACM for Nitro Enclaves configuration file that you can use as a starting point for your own configuration. Rename the sample ACM for Nitro Enclaves configuration file from `/etc/nitro_enclaves/acm-httpd.example.yaml` to `/etc/nitro_enclaves/acm.yaml`.

```bash
$ sudo mv /etc/nitro_enclaves/acm-httpd.example.yaml /etc/nitro_enclaves/acm.yaml
```
3. Specify the ARN of the certificate that you associated with the IAM role that is attached to the parent instance. Using your preferred text editor, open `/etc/nitro_enclaves/acm.yaml`. In the Acm section, for certificate_arn, specify the ARN of the certificate. Save and close the file.
4. Using your preferred text editor, open `/etc/httpd/conf.d/httpd-acm.conf`. For ServerName, specify the host name, or the common name (CN), that you specified when you created the certificate, and configure the remaining settings as needed. The following is an example of a minimal SSL/TLS configuration:

```xml
<VirtualHost *:443>
  ServerName www.example.com
  SSLEngine on
  SSLProtocol -all +TLSv1.2
  SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
  SSLCertificateFile /etc/pki/tls/certs/localhost.crt
</VirtualHost>
```

**Note**
The SSLCertificateFile and SSLCertificateKeyFile entries must be present in the configuration file. These entries will be automatically updated with the URIs after starting the ACM for Nitro Enclaves service.
5. Apache HTTP server ships with a configuration file that you can use. To use the configuration file, rename it from `/etc/httpd/conf.d/ssl.conf` to `/etc/httpd/conf.d/httpd-acm.conf`.

```bash
$ sudo mv /etc/httpd/conf.d/ssl.conf /etc/httpd/conf.d/httpd-acm.conf
```
6. Start the ACM for Nitro Enclaves service and ensure that it starts automatically at instance boot.

```bash
$ sudo systemctl start nitro-enclaves-acm.service
$ sudo systemctl enable nitro-enclaves-acm
```
7. Test that the ACM for Nitro Enclaves is working as expected.
   - If you used a public certificate, use the following command.

```bash
$ curl https://host_name_or_IPM
```
   - If you used a private certificate, you must add the host name to `/etc/hosts` in the following format: `127.0.0.1 host_name`, for example `127.0.0.1 example.com`. And you must specify the certificate chain to use to validate the certificate. For more information about generating the certificate chain for your certificate, see [Exporting a Private Certificate](#) in the [AWS Certificate Manager User Guide](#).
Using multiple certificates

You can also add multiple ACM certificates; one for each PKCS#11 token. For each additional certificate that you need to add, repeat Step 4: Associate the role with the ACM certificate (p. 51) in order to associate your IAM role with the additional ACM certificates.

Then to add more PKCS#11 tokens, open /etc/nitro_enclaves/acm.yaml with your preferred text editor, and under the token section, add another label block and specify a label name, the ARN of the additional certificate, and a path for the NGINX stanza or Apache HTTP configuration file respectively. For example, the following snippet shows the format to be used for two ACM certificates (the initial certificate and two additional certificates):

NGINX

tokens:
  - label: nginx-acm-token
    # Configure a managed token, sourced from an ACM certificate.
    source:
      Acm:
      # The certificate ARN
      certificate_arn: "arn:aws:acm:us-east-1:123456789012:certificate/d4c3b2a1-e5d0-4d51-95d9-1927fEXAMPLE"
    target:
      NginxStanza:
      # Path to the nginx stanza to be written by the ACM service whenever # the certificate configuration changes (e.g. after a certificate renewal). # This file must be included from the main nginx config
      path: /etc/pki/nginx/nginx-acm.conf
      # Stanza file owner (i.e. the user nginx is configured to run as).
      user: nginx
      # PKCS#11 token 2
  - label: token_2_name
    source:
      Acm:
Using multiple certificates

certificate_arn: "certificate_2_ARN"
target:
  NginxStanza:
    path: /etc/pki/nginx/nginx-acm-2.conf
    user: nginx

Apache

tokens:
  # A label for this PKCS#11 token
  - label: token_1_name
    # Configure a managed token, sourced from an ACM certificate.
    source:
      Acm:
        # The certificate ARN
        # Note: this certificate must have been associated with the IAM role assigned
to the instance
        # on which ACM for Nitro Enclaves is run.
        certificate_arn: "certificate_1_ARN"
    target:
      Conf:
        # Path to the server configuration file to be written by the ACM service whenever the
        # certificate configuration changes (e.g. after a certificate renewal). The
        SSLCertificateKeyFile
        # and optionally the SSLCertificateFile directives shall be populated.
        path: /etc/httpd/conf.d/httpd-acm.conf
        # Configuration file owner (i.e. the user httpd is configured to run as).
        user: apache
        # Attestation period (seconds)
        refresh_interval_secs: 43200

  - label: token_2_name
    # Configure a managed token, sourced from an ACM certificate.
    source:
      Acm:
        # The certificate ARN
        # Note: this certificate must have been associated with the IAM role assigned
to the instance
        # on which ACM for Nitro Enclaves is run.
        certificate_arn: "certificate_2_ARN"
    target:
      Conf:
        # Path to the server configuration file to be written by the ACM service whenever the
        # certificate configuration changes (e.g. after a certificate renewal). The SSLCertificateKeyFile and optionally
        SSLCertificateFile directives shall be populated.
        path: /etc/httpd/conf.d/httpd-acm-2.conf
        # Configuration file owner (i.e. the user httpd is configured to run as).
        user: apache
        # Attestation repeat period (seconds)
        refresh_interval_secs: 43200

Note
You also need to update the /etc/nginx/nginx.conf configuration (NGINX) or the /etc/httpd/conf.d/httpd-acm.conf configuration file (Apache) to include the additional ACM certificates. For more information about configuring NGINX for multiple domains and about different use cases, refer to the NGINX documentation or Apache HTTP server documentation.

After you have completed the necessary configuration, run the following command to restart the Start the ACM for Nitro Enclaves service.
Updating ACM for Nitro Enclaves

If you have already installed ACM for Nitro Enclaves, use the following command to update it to the latest version.

```
$ sudo systemctl restart nitro-enclaves-acm.service
```

Uninstalling ACM for Nitro Enclaves

If you no longer want to use ACM for Nitro Enclaves, use the following steps to uninstall it.

**To uninstall ACM for Nitro Enclaves**

1. Stop the web server.
   - **NGINX**
     ```
     $ sudo systemctl stop nginx
     ```
   - **Apache**
     ```
     $ sudo systemctl stop httpd
     ```

2. Stop the ACM for Nitro Enclaves service.
   ```
   $ sudo systemctl stop nitro-enclaves-acm.service
   ```

   ```
   $ sudo yum remove aws-nitro-enclaves-acm
   ```
Security

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that are built to meet the requirements of the most security-sensitive organizations.

Topics

• Shared responsibility (p. 60)
• Amazon EC2 security (p. 60)
• Enclave security (p. 60)
• Logging API calls for the Nitro Enclaves with AWS CloudTrail (p. 61)

Shared responsibility

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security of the cloud and security in the cloud:

• **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to Amazon EC2, see AWS Services in Scope by Compliance Program.

• **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

Amazon EC2 security

The AWS Nitro Enclaves parent instance benefits from the standard security features and capabilities of Amazon EC2. The following documentation helps you understand how to apply the shared responsibility model when using Amazon EC2. It shows you how to configure Amazon EC2 to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon EC2 resources.

• Infrastructure security in Amazon EC2
• Amazon EC2 and interface VPC endpoints
• Resilience in Amazon EC2
• Data protection in Amazon EC2
• Identity and access management for Amazon EC2
• Amazon EC2 key pairs and Linux instances
• Amazon EC2 security groups for Linux instances
• Update management in Amazon EC2
• Compliance validation for Amazon EC2

Enclave security

Nitro Enclaves use the same Nitro Hypervisor technology that provides CPU and memory isolation for Amazon EC2 instances in order to isolate the vCPUs and memory for an enclave from a parent
instance. Enclaves provide only secure local socket connectivity with their parent instance. They have no persistent storage, SSH access, or external networking. Users cannot SSH into an enclave, and the data and applications inside the enclave cannot be accessed by the processes, applications, or users (root or admin) of the parent instance.

Nitro Enclaves also supports a cryptographic attestation feature, which allows you to verify an enclave's identity and ensure that only authorized code is running inside it. Attestation ensures that only authorized enclaves are able to decrypt sensitive data and perform specific cryptographic operations.

Nitro Enclaves integrates with AWS Key Management Service (KMS). AWS KMS makes it easy for you to create and manage cryptographic keys and control their use across a wide range of AWS services and in your applications. AWS KMS provides built-in attestation support that allows you to create condition keys for AWS KMS key policies that include an enclave's platform configuration registers. This ensures that only authorized enclaves are able to perform cryptographic operations using a specific KMS key.

Logging API calls for the Nitro Enclaves with AWS CloudTrail

AWS Nitro Enclaves is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user or role in Nitro Enclaves. CloudTrail captures AWS KMS API calls made from enclaves as events. If you create a trail, you can enable continuous delivery of the events to an Amazon Simple Storage Service (Amazon S3) bucket. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in Event history. You can use the information collected by CloudTrail to audit AWS KMS API calls made by enclaves.

For more information about CloudTrail, see the AWS CloudTrail User Guide.

Nitro Enclaves information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When supported event activity occurs in an enclave, that activity is recorded in a CloudTrail event along with other AWS service events in Event history. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your enclave, create a trail. A trail enables CloudTrail to deliver log files to an S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- Configuring Amazon SNS Notifications for CloudTrail

Nitro Enclaves supports logging the following Nitro Enclaves SDKs (that call AWS KMS APIs) as events in CloudTrail log files:

- kms-decrypt (Decrypt)
- generate-data-key (GenerateDataKey)
- generate-random (GenerateRandom)

Every event or log entry contains information about the origins of the request. Event logs generated by API calls from an enclave include the following additional fields that provide information about the identity of the enclave.
Understanding Nitro Enclaves log file entries

CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order. The following examples show CloudTrail log entries for the supported actions.

- **GenerateRandom**

  ```json
  {
  "eventVersion": "1.02",
  "userIdentity": {
      "type": "IAMUser",
      "principalId": "EX_PRINCIPAL_ID",
      "arn": "arn:aws:iam::111122223333:user/Alice",
      "accountId": "111122223333",
      "accessKeyId": "EXAMPLE_KEY_ID",
      "userName": "Alice"
    },
  "eventTime": "2014-11-04T00:52:37Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateRandom",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "AWS Internal",
  "requestParameters": null,
  "responseElements": null,
  "additionalEventData": {
    "recipient": {
      "attestationDocumentModuleId": "i-123456789abcde123-enc123456789abcde12",
      "attestationDocumentEnclaveImageDigest": "ee0d451a2ff9aaa9bccc07700b9cab123a0ac2386ef7e88ad5ea6c72ebabec840957328e2ec890b608c9b05cb8ebe6a"
    }
  },
  "requestID": "df1e3de6-63bc-11e4-bc2b-4198b6150d5c",
  "eventID": "239cb9f7-ae05-4c94-9221-6ea30eef0442",
  "readOnly": true,
  "resources": [],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
  }
  ```
• GenerateDataKey

```json
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::111122223333:user/Alice",
    "accountId": "111122223333",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "userName": "Alice"
  },
  "eventTime": "2014-11-04T00:52:40Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "AWS Internal",
  "requestParameters": {
    "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "numberOfBytes": 32
  },
  "responseElements": null,
  "additionalEventData": {
    "recipient": {
      "attestationDocumentModuleId": "i-123456789abcde123-enc123456789abcde12",
      "attestationDocumentEnclaveImageDigest": "ee0d451a2ff99aaa9bccc0d7bb9b9123a0ac2386ef7e88ad5ea6c72ebabea840957328e2ec890b408c9b06cb8ebe6a"
    },
    "requestID": "e0eb83e3-63bc-11e4-bc2b-41986150d5c",
    "eventID": "a9dea4f9-8395-46c0-942c-f509c02b71",
    "readOnly": true,
    "resources": [{
      "ARN": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "accountId": "111122223333"
    }],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
  }
}
```

• Decrypt

```json
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::111122223333:user/Alice",
    "accountId": "111122223333",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "userName": "Alice"
  },
  "eventTime": "2020-07-27T22:58:24Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "192.0.2.0",
  "userAgent": "AWS Internal",
  "requestParameters": {
    "encryptionAlgorithm": "SYMMETRIC_DEFAULT",
    "keyId": "12345ac3-fbbf-4143-abcd-28b39example"
  }
}
```
Understanding Nitro Enclaves log file entries
Nitro Enclaves Command Line Interface

The Nitro Enclaves CLI (Nitro CLI) is a command line tool for managing the lifecycle of enclaves. You can use the Nitro CLI to create, manage, and terminate enclaves. The Nitro CLI must be installed on the Amazon EC2 parent instance.

Topics
- Installing the Nitro Enclaves CLI on Linux (p. 65)
- Installing the Nitro Enclaves CLI on Windows (p. 67)
- Nitro Enclaves Command Line Interface reference (p. 67)
- Error codes (p. 80)

Installing the Nitro Enclaves CLI on Linux

The following instructions are for installing or uninstalling the AWS Nitro Enclaves CLI on or from a parent instance running Amazon Linux 2. For instructions for installing the Nitro CLI on different Linux distributions, see the Nitro CLI github repository.

Topics
- Install AWS Nitro Enclaves CLI (p. 65)
- Uninstall AWS Nitro Enclaves CLI (p. 66)

Install AWS Nitro Enclaves CLI

To install the Nitro CLI on an instance running AL2

1. Install the Nitro CLI.

   $ sudo amazon-linux-extras install aws-nitro-enclaves-cli -y

2. Install the Nitro Enclaves development tools needed to build enclave images. The development tools also includes some sample applications.

   $ sudo yum install aws-nitro-enclaves-cli-devel -y

3. Add your user to the ne user group.

   $ sudo usermod -aG ne $username

4. Add your user to the docker user group.

   $ sudo usermod -aG docker $username
5. For the changes to take effect, log out of the instance and then reconnect to it.

6. Verify that the Nitro CLI installed correctly.

   ```
   $ nitro-cli --version
   ```

   The command should return version information about the Nitro CLI.

7. Preallocate the memory and the vCPUs that you intend to use for the enclaves.

   **Important**
   Nitro Enclaves uses an allocator service to preallocate vCPUs and memory to the enclaves. The amount of vCPUs and memory to preallocate are defined in the allocator service configuration file (`/etc/nitro_enclaves/allocator.yaml`). By default, the configuration file is set up to preallocate 512 MiB of memory and 2 vCPUs for use by the enclaves. In some cases, you might need to manually update the configuration file to preallocate a different number vCPUs or amount of memory. For example:

   - If you launched an AWS Graviton-based instance with 2 vCPUs, you must configure the allocator service to preallocate only 1 vCPU.
   - If you launched an instance with 4 or more vCPUs, you can configure the allocator service to preallocate more vCPUs to the enclave.
   - If you are going to run multiple enclaves, you must configure the allocator service to preallocate enough vCPUs and memory for all of the enclaves. For example, to run 3 enclaves with 4 vCPUs and 2 GiB memory each, you must configure the allocator service to preallocate 12 vCPUs and 6 GiB of memory.

   If you need to change the configuration file, use your preferred text editor to open `/etc/nitro_enclaves/allocator.yaml`. Then, for `memory_mib` and `cpu_count`, specify the overall amount of memory (in MiB) and the number of vCPUs that you want to preallocate. Save and close the file and then run the command below.

   If you want to preallocate the default 512 MiB of memory and 2 vCPUs, you do not need to make any changes to the configuration file.

   Run the following command to allocate the resource specified in the configuration file and to ensure that they are automatically allocated every time the instance starts.

   ```
   $ sudo systemctl enable --now nitro-enclaves-allocator.service
   ```

   **Note**
   When you create an enclave, the requested memory and vCPUs must be less than or equal to the values that you specified here. If you need to create an enclave with more memory or vCPUs in the future, you must update the values in this file and restart the service.

8. Start the Docker service and ensure that it starts every time the instance starts.

   ```
   $ sudo systemctl enable --now docker
   ```

---

**Uninstall AWS Nitro Enclaves CLI**

If you no longer want to use AWS Nitro Enclaves, use the following command to uninstall the AWS Nitro Enclaves CLI.

```
$ sudo yum remove aws-nitro-enclaves-cli
```
Installing the Nitro Enclaves CLI on Windows

The AWS Nitro Enclaves CLI is packaged together with all of the components that are required to run Nitro Enclaves on a Windows parent instance. The package includes kernel drivers for the Enclave and vsock devices, a service provider interface for Winsock to support vsock sockets, the vsock-proxy, and the AWS Nitro Enclaves CLI.

The following instructions are for installing and uninstalling the AWS Nitro Enclaves CLI on or from a parent instance running Windows.

**Note**

You may get the following error when you install, uninstall, or update the Nitro CLI:

Installation failed with code 3010. This message indicates that a reboot is required to complete the installation. This error is likely caused by a component in use, such as a running enclave or a vsock-proxy process. To complete the installation, shut down all applications running on the instance and reboot it.

**Topics**

- Install AWS Nitro Enclaves CLI (p. 67)
- Uninstall AWS Nitro Enclaves CLI (p. 67)

---

**Install AWS Nitro Enclaves CLI**

To use the Nitro Enclaves on your parent instance, you must install the `AWSNitroEnclavesWindows` package using AWS Systems Manager Distributor.

Before you can install a package using the AWS Systems Manager Distributor, you must first complete the Distributor prerequisites.

After you have completed the prerequisites, install the `AWSNitroEnclavesWindows` package. For more information, see one of the following in the AWS Systems Manager User Guide:

- Installing or updating a package one time using the console
- Installing a package one time using the AWS CLI

You must reload the path environment variable from the updated environment in any PowerShell or command prompt already open on the instance. When you open a new PowerShell or command prompt, Windows automatically updates the path variable.

**Uninstall AWS Nitro Enclaves CLI**

If you no longer need to use Nitro Enclaves, you can remove the AWS Nitro Enclaves CLI from the parent instance by uninstalling the `AWSNitroEnclavesWindows` package using the AWS Systems Manager Distributor. For more information see Uninstall a package in the AWS Systems Manager User Guide.

Alternatively, you can uninstall the AWS Nitro Enclaves CLI using Programs and Features, which can be accessed using the Windows Control Panel.

---

**Nitro Enclaves Command Line Interface reference**

The following commands are available in the Nitro CLI. All of the Nitro CLI commands start with `nitro-cli`, followed by one of the following subcommands.
nitro-cli build-enclave

Converts a Docker image into an enclave image file (.eif). You can specify either a local directory containing a Dockerfile, or a Docker image in a Docker repository.

**Important**
This command is not supported on Windows. If you are using a Windows parent instance, you must run this command on a Linux computer and then transfer the enclave image file (.eif) to the Windows parent instance.

The command returns a set of measurements (SHA384 hashes) that are unique to the enclave image file. These measurements are provided in the form of platform configuration registers (PCRs). The PCRs are used during the enclave’s attestation process. For more information, see [Nitro Enclaves concepts](#) (p. 4).

For example, when using Nitro Enclaves with AWS Key Management Service (AWS KMS), you can specify these PCRs in condition keys for customer managed keys policies. When an application in the enclave performs an AWS KMS operation, AWS KMS compares the PCRs in the enclave’s signed attestation document with the PCRs specified in the condition keys of the KMS key policy before allowing the operation.

**Syntax**
```
nitro-cli build-enclave --docker-uri repository:tag --docker-dir /path_to/dockerfile_directory --output-file enclave_image_filename --private-key key.pem --signing-certificate certificate.pem
```

**Options**

---**docker-uri**

Specifies the uniform resource identifier (URI) of a Docker image in a Docker repository. The URI is specified using the repository:tag format.

Type: String

Required: Yes
**--docker-dir**

Specifies the path to a local directory containing a Dockerfile.

Type: String

Required: No

**--output-file**

The file name of the enclave image file that is created.

Type: String

Required: Yes

**--private-key**

The private key to use to sign the enclave image file. If you specify --private-key then you must also specify --signing-certificate. If you specify these parameter, the command creates a signed enclave image file. The command output will include an additional PCR, PCR8, which can be used in condition keys for KMS key policies. For more information, see Where to get an enclave's measurements (p. 26).

Type: String

Required: No

**--signing-certificate**

The signing key to use to sign the enclave image file. If you specify --signing-certificate then you must also specify --private-key. If you specify these parameter, the command creates a signed enclave image file. The command output will include and additional PCR, PCR8, which can be used in condition keys for KMS key policies. For more information, see Where to get an enclave's measurements (p. 26).

**Important**

Ensure that the specified certificate is still valid. If you attempt to start an enclave with an enclave image file that is signed with a certificate that is no longer valid, the nitro-cli run-enclave fails with errors E36, E39, and E11.

Type: String

Required: No

**Output**

**Measurements**

Cryptographic measurements (SHA384 hashes) that are unique to the enclave image file.

Type: String

**Example**

The following example converts a Docker image with a URI of `sample:latest` to an enclave image file named `sample.eif`.

**Command**
nitro-cli build-enclave --docker-uri sample:latest --output-file sample.eif

**Output**

Enclave Image successfully created.

```
{
   "Measurements": {
      "HashAlgorithm": "Sha384 { ... }",
      "PCR0": "EXAMPLE59044e337c00068c2c033546644e37aa4466b853ca486dd149f641f15071961db2a0827beccea9cade3EXAMPLE",
      "PCR1": "EXAMPLE7783d0c23167299fbe5a69622490a9bd0f82e94a0a1a48b0e7c56130c0c1e6555de7c0aa3d7901fbc58EXAMPLE",
      "PCR2": "EXAMPLE4b51589e8374b7f695b4649d1f1e9b528b05ab75a49f9a04a1ec36be81280caab0486f660b9207ac0EXAMPLE"
   }
}
```

**nitro-cli run-enclave**

Launches a new enclave. This command partitions the specified number of vCPUs and the amount of memory from the Amazon EC2 parent instance to create the enclave. You also need to provide an enclave image file (.eif) that contains the operating system and the applications that you want to run inside the enclave.

**Important**

If you attempt to start an enclave with an enclave image file that is signed with a certificate that is no longer valid, the `nitro-cli run-enclave` command fails with errors E36, E39, and E11.

**Topics**

- Syntax (p. 70)
- Options (p. 71)
- Output (p. 72)
- Examples (p. 73)

**Syntax**

```
nitro-cli run-enclave [--enclave-name enclave_name] [--cpu-count number_of_vcpus]
| --cpu-ids list_of_vcpu_ids] --memory amount_of_memory_in_MiB --eif-path path_to_enclave_image_file [--enclave-cid cid_number] [--debug-mode]
```

Or

```
nitro-cli run-enclave --config config_file.json
```

```
config_file.json
```

```
{
    "enclave_name": enclave_name,
    "cpu_count": number_of_vcpus,
    "cpu_ids": list_of_vcpu_ids,
    "memory_mib": amount_of_memory_in_MiB,
    "eif_path": "path_to_enclave_image_file",
    "enclave_cid": cid_number,
    "debug_mode": true|false
}
Options

--enclave-name

Specifies a unique name for the enclave. You can use this name to reference the enclave when using the `nitro-cli console` and `nitro-cli terminate-enclave` commands.

If you do not specify a name, the name of the enclave image file (.eif) is used as the enclave name.

Type: String
Required: No

--cpu-count

Specifies the number of vCPUs to allocate to the enclave.

**Note**
- Amazon EC2 instances support multithreading, which enables multiple threads to run concurrently on a single CPU core. Each thread is represented as a virtual CPU (vCPU) on the instance. For more information about vCPUs, see [Optimize CPU options](#) in the Amazon EC2 User Guide.
- If the parent instance is enabled for multithreading, you must specify an even number of vCPUs.

The number of vCPUs that you can allocate to an enclave depends on the size and configuration of the parent instance. If the parent instance is enabled for multithreading, you must leave at least 2 vCPUs for the parent instance. If multithreading is not enabled, you must leave at least 1 vCPU for the parent instance. For example, if your parent instance has 4 vCPUs and it is enabled for multithreading, you can allocate up to 2 vCPUs to the enclave.

You must specify either `--cpu-count` or `--cpu-ids`. If you specify this option, omit `--cpu-ids`.

Type: Integer
Required: Yes, if `--cpu-ids` is not specified

--cpu-ids

Specifies the IDs of the vCPUs to allocate to the enclave.

**Note**
- Amazon EC2 instances support multithreading, which enables multiple threads to run concurrently on a single CPU core. Each thread is represented as a virtual CPU (vCPU) on the instance. For more information about vCPUs, see [Optimize CPU options](#) in the Amazon EC2 User Guide.
- If the parent instance is enabled for multithreading, you must specify an even number of vCPUs.

The number of vCPUs that you can allocate to an enclave depends on the size and configuration of the parent instance. If the parent instance is enabled for multithreading, you must leave at least 2 vCPUs for the parent instance. If multithreading is not enabled, you must leave at least 1 vCPU for the parent instance. For example, if your parent instance has 4 vCPUs and it is enabled for multithreading, you can allocate up to 2 vCPUs to the enclave.

You must specify either `--cpu-count` or `--cpu-ids`. If you specify this option, omit `--cpu-count`. 
Type: String  
Required: Yes, if --cpu-count is not specified

**--memory**

Specifies the amount of memory (in MiB) to allocate to the enclave.

The amount of memory that you can allocate to an enclave depends on the size of the parent instance and the applications that you intend to run on it. The specified amount of memory cannot exceed the amount of memory provided by the parent instance. You must leave enough memory for the applications running on the parent instance. You must allocate a minimum of 64 MiB of memory to the enclave.

Type: Integer (MiB)

Required: Yes

**--eif-path**

Specifies the path to the enclave image file.

Type: String

Required: Yes

**--enclave-cid**

Specifies the context identifier (CID) for the enclave. The CID is the socket address used by the vsock socket. Only CIDs of 4 and higher can be specified. If you omit this option, a random CID is allocated to the enclave.

Type: Integer

Required: No

**--debug-mode**

Indicates whether to run the enclave in debug mode. Specify this option to enable debug mode, or omit it to disable debug mode.

If you enable debug mode, you can view the enclave's console in read-only mode using the `nitro-cli console` command. Enclaves booted in debug mode generate attestation documents with PCRs that are made up entirely of zeros (0000000000000000000000000000000000000000).  

Required: No

**--config**

The path to a .json configuration file that specifies the parameters for the enclave. If you specify --config, the specified json file must include the required and optional parameters as described above, and you must not specify any other parameters in the command itself.

Type: String

Required: No

**Output**

**EnclaveName**

The unique name of the enclave.
Type: String
EnclaveID
The unique ID of the enclave.

Type: String
ProcessID
The process identifier (PID) of the process holding the enclave's resources.

Type: String
EnclaveCID
The context ID (CID) of the enclave.

Type: Integer
NumberOfCPUs
The number of vCPUs allocated to the enclave from the parent instance.

Type: Integer
CPUIDs
The IDs of the vCPUs allocated to the enclave from the parent instance.

Type: Integer
MemoryMiB
The amount of memory (in MiB) allocated to the enclave from the parent instance.

Examples
Example 1: Inline parameters
The following example creates an enclave with 2 vCPUs, 1600 MiB of memory, and a context ID of 10. It also uses an enclave image file named sample.eif, which is located in the same directory from which the command is being run.

Command
```
nitro-cli run-enclave --enclave-name my_enclave --cpu-count 2 --memory 1600 --eif-path sample.eif --enclave-cid 10
```

Output
```
Start allocating memory...
Started enclave with enclave-cid: 10, memory: 1600 MiB, cpu-ids: [1, 3]
{
    "EnclaveName": "my_enclave",
    "EnclaveID": "i-abc12345def67890a-enc9876abcd543210ef12",
    "ProcessID": 12345,
    "EnclaveCID": 10,
    "NumberOfCPUs": 2,
    "CPUIDs": [1, 3]
}
Example 2: Config file

The following example creates an enclave with 2 vCPUs, 1600 MiB of memory, and a context ID of 10. It also uses an enclave image file named sample.eif, which is located in the same directory from which the command is being run.

Command

```bash
nitro-cli run-enclave --config enclave_config.json
```

Content of `enclave_config.json`:

```json
{
  "enclave_name": "my_enclave",
  "cpu_count": 2,
  "memory_mib": 1600,
  "eif_path": "sample.eif",
  "enclave_cid": 10,
  "debug_mode": true
}
```

Output

```
Start allocating memory...
Started enclave with enclave-cid: 10, memory: 1600 MiB, cpu-ids: [1, 3]
{
  "EnclaveName": "my_enclave",
  "EnclaveID": "i-abc12345def67890a-enc9876abcd543210ef12",
  "ProcessID": 12345,
  "EnclaveCID": 10,
  "NumberOfCPUs": 2,
  "CPUIDs": [
    1,
    3
  ],
  "MemoryMiB": 1600
}
```

nitro-cli describe-enclaves

Describes an enclave.

Topics

- Syntax (p. 74)
- Output (p. 75)
- Example (p. 75)

Syntax

```bash
nitro-cli describe-enclaves
```
Output

**EnclaveName**

The unique name of the enclave.

Type: String

**EnclaveID**

The unique ID of the enclave.

Type: String

**ProcessID**

[Linux parent instances only] The process identifier (PID) of the process holding the enclave's resources.

Type: String

**EnclaveCID**

The unique context ID (CID) of the enclave. The CID is the socket address used by the vsock socket.

Type: Integer

**NumberOfCPUs**

The number of vCPUs allocated to the enclave from the parent instance.

Type: Integer

**CPUIDs**

[Linux parent instances only] The IDs of the vCPUs allocated to the enclave from the parent instance.

Type: Integer

**MemoryMiB**

The amount of memory (in MiB) allocated to the enclave from the parent instance.

Type: Integer

**State**

The current status of the enclave.

Possible values: running | terminating

Type: String

**Flags**

Indicates if the enclave is in debug mode. None indicates that debug mode is disabled. Debug indicates that debug mode is enabled.

Possible values: None | Debug

Type: String

Example

The following example describes an enclave.

Command
nitro-cli describe-enclaves

Output

```
[{
    "EnclaveName": "my_enclave",
    "EnclaveID": "i-abc12345def67890a-enc9876abcd543210ef12",
    "ProcessID": 12345,
    "EnclaveCID": 10,
    "NumberOfCPUs": 2,
    "CPUIDs": [
        1,
        3
    ],
    "MemoryMiB": 1600,
    "State": "RUNNING",
    "Flags": "NONE"
}]
```

nitro-cli console

Enters a read-only console for the specified enclave. This enables you to view the enclave's console output to assist with troubleshooting. You can use this command only on an enclave that was launched with the `--debug-mode` flag.

Topics
- Syntax (p. 76)
- Options (p. 76)
- Example (p. 77)

Syntax

```
nitro-cli console [--enclave-name enclave_name | --enclave-id enclave_id] [--disconnect-timeout number_of_seconds]
```

Options

--enclave-name

The unique name of the enclave. You must specify either `--enclave-name` or `--enclave-id`.

Type: String

Required: Yes, if `--enclave-id` is not specified

--enclave-id

The unique ID of the enclave. You must specify either `--enclave-id` or `--enclave-name`.

Type: String

Required: Yes, if `--enclave-name` is not specified
--disconnect-timeout

The number of seconds after which to automatically disconnect idle console sessions.

Type: Integer
Required: No

Example

The following command enters a read-only console for an enclave with an ID of i-05f6ed443ae428c95-enc173dfe3e2b1c87b. The session automatically disconnects if the connection is idle for 60 seconds.

Command

```
nitro-cli console --enclave-id i-05f6ed443ae428c95-enc173dfe3e2b1c87b --disconnect-timeout 60
```
Example

The following example describes an enclave image file named `sample.eif`.

**Command**

```bash
nitro-cli describe-eif --eif-path image.eif
```

**Output**

```
{
    "Measurements": {
        "HashAlgorithm": "Sha384 { ... }",
        "PCR0": "EXAMPLE59044e337c00068c2c033546641e37aa466b853ca486dd149f641f15071961db2a0827becc8aa9cde3EXAMPLE",
        "PCR1": "EXAMPLE7783d0c23167299f8be5a69622490a9bdf82e94a0a1a48b0e7c56130c0c1e6555de7c0aa3d7901fbc58EXAMPLE",
        "PCR2": "EXAMPLE4b51589e8374b7f695b4649d1f1e9b528b05ab75a49f9a0a4a1ec36be81280caab0486f660b9207ac0EXAMPLE"
    }
}
```

**nitro-cli pcr**

Returns the platform configuration register (PCR) value for a specified input file or PEM certificate. You can use this command to identify the files and signing certificate that were used to sign an enclave by comparing the command output with PCR values in the enclave's build measurements.

**Topics**

- Syntax (p. 78)
- Options (p. 78)
- Output (p. 79)
- Example (p. 79)

**Syntax**

```
nitro-cli pcr [--input path_to_file | --signing-certificate path_to_certificate]
```

**Options**

- **--input**

  Specifies the path to the file for which to generate the platform configuration register (PCR) value.

  Type: String

  Required: Yes, if `--signing-certificate` is not specified

- **--signing-certificate**

  Specifies the path to the PEM certificate for which to generate PCR8. This option is used to specifically request the PCR8 value by performing deserialisation of the certificate and PEM format validation.
**nitro-cli terminate-enclave**

Terminates a specific enclave or all enclaves owned by the current user.

To terminate a specific enclave, specify --enclave-name or --enclave-id. To terminate all enclaves, specify --all.

**Topics**
- Syntax (p. 79)
- Options (p. 79)
- Example (p. 80)

**Syntax**

```bash
nitro-cli terminate-enclave [--enclave-id enclave_id | --enclave-name enclave_name | --all]
```

**Options**

**--enclave-name**

The unique name of the enclave to terminate. You must specify either --enclave-name or --enclave-id.

Type: String
**Error codes**

This section lists the possible errors that the Nitro CLI can return.

---

**Example**

**Example: Terminate specific enclave**

The following example terminates an enclave with an ID of `i-abc12345def67890a-enc9876abcd543210ef12`.

**Command**

```
nitro-cli terminate-enclave --enclave-id i-abc12345def67890a-enc9876abcd543210ef12
```

**Output**

```
Successfully terminated enclave i-abc12345def67890a-enc9876abcd543210ef12.
{
  "EnclaveID": "i-abc12345def67890a-enc9876abcd543210ef12",
  "Terminated": true
}
```

**Example: Terminate all running enclaves**

The following example terminates all of the enclaves owned by the current user.

**Command**

```
nitro-cli terminate-enclave --all
```

**Output**

```
Successfully terminated enclave i-abc12345def67890a-enc9876abcd543210ef12.
{
  "EnclaveID": "i-abc12345def67890a-enc9876abcd543210ef12",
  "Terminated": true
}
```
E01

**Missing mandatory argument.** At least one mandatory argument has not been specified. Ensure that all mandatory arguments have been specified.

E02

**Conflicting arguments.** The command includes two or more incompatible arguments. Ensure that you specify only one of the conflicting arguments. For example, you cannot specify `--cpu-count` and `--cpu-ids` in the same `run-enclave` command.

E03

**Invalid argument provided.** A value of the incorrect type has been specified for one or more arguments. For example, a string was specified for an argument that expects an integer. Ensure that all values are of the expected type.

E04

**Socket pair creation failure.** The Nitro CLI attempted to open a stream pair with the enclave, but the stream initialization has failed. Either there is insufficient memory available for the Nitro CLI process, or the system-wide maximum number of open descriptors was reached. Retry the command. If that fails, reboot the instance and then retry the command.

E05

**Process spawn failure.** The Nitro CLI failed to spawn the enclave process while running the `run-enclave` command. Either the system has reached its maximum number of threads, or there is insufficient memory available to spawn the new process. Ensure that the system has enough free memory and then retry the command. If that fails, reboot the instance and then retry the command.

E06

**Daemonize process failure.** An error occurred while attempting to daemonize the newly spawned enclave process. Possible reasons are that the system has reached its maximum number of threads, there is insufficient memory available to spawn the new process, or the configuration of the Nitro CLI main process is not allowing the daemon creation process. Ensure that the system has enough free memory and then retry the command. If that fails, reboot the instance and then retry the command.

E07

**Read from disk failure.** The Nitro CLI failed to read content from the enclave's socket directory (typically `/var/run/nitro_enclaves/`) while running the `describe-enclave` command. Ensure that the directory exists and that it has the correct permissions. Alternatively, run the Nitro Enclaves configuration script to reconfigure the environment.

E08

**Unusable connection error.** The Nitro CLI is unable to connect to an enclave. Ensure that it exists and that it is in the `running` state.

E09

**Socket close error.** The Nitro CLI is unable to close the communication channel. The socket close operation was interrupted by another signal. Retry the command.

E10

**Socket connect set timeout error.** The Nitro CLI failed to configure a specific timeout for the specified socket. Ensure that the operation is being performed on a valid socket.

E11

**Socket error.** An unexpected error occurred with the socket.
E12

**Epoll error.** The Nitro CLI failed to register the enclave descriptor for event monitoring with `epoll`. Either the system has insufficient memory to handle the requested operation, or the per-user maximum number of watches was reached while trying to register a new descriptor on an epoll instance. Ensure that the system has enough free memory and then retry the command. If that fails, reboot the instance and then retry the command.

E13

**Inotify error.** The Nitro CLI failed to configure a socket for monitoring. Either the system has insufficient memory to handle the requested operation, or the user limit of inotify watches has been reached. Ensure that the system has enough free memory and then retry the command. If that fails, reboot the instance and then retry the command.

E14

**Invalid command.** An unknown command or command argument was specified. Verify the command and argument names.

E15

**Lock acquire failure.** The Nitro CLI failed to obtain a lock on an object with concurrent access, such as a structure containing information about a running enclave. A previous thread failed an operation while holding the lock. Retry the command. If that fails, reboot the instance and then retry the command.

E16

**Thread join failure.** The Nitro CLI failed to join a thread after it finished executing. Retry the command.

E17

**Serde error.** An error occurred while serializing or deserializing a command or command response. The JSON in the supplied command might not be valid. If you are supplying command arguments in the JSON file, ensure that the supplied JSON is valid.

E18

**File permissions error.** You do not have permission to modify the logging file (typically `/var/log/nitro_enclaves/nitro_enclaves.log`). Ensure that your user is part of the `neutral` user group. For more information, see `Installing the Nitro Enclaves CLI on Linux (p. 65)`.

E19

**File operation failure.** The system failed to perform the requested file operations. Ensure that the file on which the operation is performed exists and that you have permission to modify it.

E20

**Invalid CPU configuration.** The same CPU ID has been specified more than once for the `--cpu-ids` argument. Ensure that each vCPU ID is specified only once.

E21

**No such CPU available in the pool.** One or more of the specified CPU IDs does not exist in the CPU pool. Either retry the command and specify different vCPU IDs, or preallocate the environment resources so that the vCPU pool includes the vCPU IDs that you want to use. For more information, see `Installing the Nitro Enclaves CLI on Linux (p. 65)`.

E22

**Insufficient CPUs available in the pool.** The number of requested vCPUs is greater than the number of available vCPUs. Either specify a number of vCPUs less than or equal to the configured vCPU pool
size, or preallocate the environment resources so that the vCPU pool includes the number of vCPUs that you want to use. For more information, see Installing the Nitro Enclaves CLI on Linux (p. 65).

E23

**Malformed CPU ID error.** This error appears when a lscpu line is malformed and it reports an online CPUs list that is not valid. Ensure that the lscpu output is not corrupt.

E24

**CPU error.** A CPU line interval is not valid. Ensure that the lscpu output is not corrupt.

E25

**No such hugepage flag error.** The enclave process attempted to use a hugepage size that is not valid for initializing the enclave memory. Make sure that the Nitro CLI code has not been modified to include hugepage sizes that are not valid.

E26

**Insufficient memory requested.** Insufficient memory was requested for the enclave. The memory should be equal to or greater than the size of the enclave image file. Preallocate enough memory to ensure that the enclave image file fits in the enclave's memory. For more information, see Installing the Nitro Enclaves CLI on Linux (p. 65).

E27

**Insufficient memory available.** The amount of requested memory is greater than the amount of available memory. The enclave memory should not be greater than the size of the configured hugepage memory. For example, if you request 100 MiB of memory while the allocated hugepage memory is 80 MiB, the request fails. Preallocate enough memory for the enclave. For more information, see Installing the Nitro Enclaves CLI on Linux (p. 65). Alternatively, specify a smaller amount of memory with the run-enclave command.

E28

**Invalid enclave descriptor.** NE_CREATE_VM ioctl returned an error. Review the error backtrace for more information.

E29

**ioctl failure.** An unexpected ioctl error occurred. Review the error backtrace for more information.

E30

**ioctl image get load info failure.** The ioctl used for getting the memory load information failed. Review the error backtrace for more information.

E31

**ioctl set memory region failure.** The ioctl used for setting a given memory region has failed. Review the error backtrace for more information.

E32

**ioctl add vCPU failure.** The ioctl used for adding a vCPU failed. Review the error backtrace for more information.

E33

**ioctl start enclave failure.** The ioctl used for starting an enclave has failed. Review the error backtrace for more information.

E34

**Memory overflow.** An error occurred while loading the enclave image file in memory regions that will be conceded to the future enclave. For example, this can occur if the regions offset plus the enclave image file size exceeds the maximum address of the target platform.
E35

**E35**

**Elf file parsing error.** Failed to fill a memory region with a section of the enclave image file.

E36

**E36**

**Enclave boot failure.** The enclave failed to return a ready signal after booting. For example, if booting from an enclave image file that is not valid, the enclave process exits immediately, before returning a ready signal. Ensure that the enclave image file is not corrupt. Review the error backtrace for more information.

E37

**E37**

**Enclave event wait error.** Failed to monitor an enclave descriptor for events.

E38

**E38**

**Enclave process command not executed error.** At least one enclave process failed to provide the description information.

E39

**E39**

**Enclave process connection failure.** The enclave manager failed to connect to at least one enclave process for retrieving the description information.

E40

**E40**

**Socket path not found.** The Nitro CLI failed to build the corresponding socket path starting from a given enclave ID.

E41

**E41**

**Enclave process send reply failure.** The enclave process failed to report its status to the requesting command.

E42

**E42**

**Enclave mmap error.** Failed to allocate memory to the enclave. Make sure that the system has enough free memory available. Retry the command. If that fails, reboot the instance and then retry the command.

E43

**E43**

**Enclave munmap error.** Failed to unmap an enclave's memory. Make sure that the Nitro CLI code has not been modified to pass flags to the memory region unmapping operation that are not valid.

E44

**E44**

**Enclave console connection failure.** The Nitro CLI failed to establish a connection with a running enclave's console. Make sure that the enclave has been started with the --debug flag.

E45

**E45**

**Enclave console read error.** Failed to read from a running enclave's console. Retry the command.

E46

**E46**

**Enclave console write output error.** Failed to write the information retrieved from a running enclave's console to a stream. Retry the command.

E47

**E47**

**Integer parsing error.** Unable to connect to a running enclave's console because the CID could not be parsed. Use the nitro-cli describe-enclaves command to confirm the CID, and to ensure that it is a valid number.

E48

**E48**

**Elf building error.** An error occurred while building the enclave image file. Review the error backtrace for more information.
E49

**Docker image build error.** An error occurred while building the enclave image file because the specified Docker image could not be built. Review the error backtrace for more information.

E50

**Docker image pull error.** An error occurred while building the enclave image file because the specified Docker image could not be pulled. Review the error backtrace for more information.

E51

**Artifacts path environment variable not set.** An error occurred while building the enclave image file because the artifacts path environment variable has not been set.

E52

**Blobs path environment variable not set.** An error occurred while building the enclave image file because the blobs path environment variable has not been set. Retry the command.

E53

**Clock skew error.** Failed to measure the elapsed time between consecutive reads from a running enclave's console. Retry the command.

E54

**Signal masking error.** Failed to mask specific signals after creating an enclave process. Retry the command.

E55

**Signal unmasking error.** Failed to unmask specific signals after creating an enclave process. Retry the command.

E56

**Logger error.** An error occurred while initializing the underlying logging system. Review the error backtrace for more information.

E57

**Hasher error.** An I/O error occurred while initializing a hasher or while writing bytes to the hasher.

E58

**Naming error.** The specified enclave name does not exist.

E59

**ELF signature checker error.** An error occurred while validating the signing certificate.
# Document history

The following table describes important additions to the AWS Nitro Enclaves documentation. We also update the documentation frequently to address the feedback that you send us.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multiple enclaves per instance</strong></td>
<td>You can run up to four Nitro Enclaves on a single parent Amazon EC2 instance.</td>
<td>January 13, 2023</td>
</tr>
<tr>
<td><strong>Amazon EKS support</strong></td>
<td>You can use Amazon Elastic Kubernetes Service to orchestrate, scale, and deploy Nitro Enclaves from a Kubernetes pod.</td>
<td>November 28, 2022</td>
</tr>
<tr>
<td><strong>AWS Graviton support</strong></td>
<td>Nitro Enclaves now supports AWS Graviton-based Amazon EC2 instance types, except A1, G5g, Im4gn, Is4gen, and T4g.</td>
<td>October 20, 2022</td>
</tr>
<tr>
<td><strong>ACM for Nitro Enclaves</strong></td>
<td>ACM for Nitro Enclaves now supports Apache HTTP server.</td>
<td>September 14, 2022</td>
</tr>
<tr>
<td><strong>Nitro Enclaves 1.2.0 for Windows</strong></td>
<td>Nitro Enclaves 1.2.0 is now available for Windows.</td>
<td>May 4, 2022</td>
</tr>
<tr>
<td><strong>Amazon SNS topic for Nitro Enclaves for Windows updates</strong></td>
<td>You can subscribe to an Amazon SNS topic to receive notifications for new versions of Nitro Enclaves for Windows.</td>
<td>August 13, 2021</td>
</tr>
<tr>
<td><strong>Nitro Enclaves 1.1.0 for Windows</strong></td>
<td>Nitro Enclaves 1.1.0 is now available for Windows.</td>
<td>July 28, 2021</td>
</tr>
<tr>
<td><strong>Nitro Enclaves on Windows</strong></td>
<td>Nitro Enclaves supports the creation of isolated compute environments form parent Amazon EC2 instances running Windows operating system.</td>
<td>April 27, 2021</td>
</tr>
<tr>
<td><strong>Initial release</strong></td>
<td>Initial release of AWS Nitro Enclaves. Nitro Enclaves is an Amazon EC2 feature that allows you to create isolated execution environments, called enclaves, from Amazon EC2 instances.</td>
<td>October 28, 2020</td>
</tr>
</tbody>
</table>
AWS glossary

For the latest AWS terminology, see the AWS glossary in the AWS Glossary Reference.