aws

User Guide

AWS Emulator



Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Emulator: User Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS Emulator? 1
Use cases 1
Examples 2
Limitations 3
How AWS Emulator keeps improving 4
Features of AWS Emulator 4
Learn more 4
Getting started 5
Setting up 6
Review the license 6
With an AI coding agent 6
To install the skill 7
Linux 7
To install AWS Emulator 7
To verify the package before you install it (optional) 8
To start and stop AWS Emulator 10
To use a different port 10
To limit the memory of the emulator on Linux 11
To uninstall AWS Emulator 11
Docker 12
To build the AWS Emulator image 12
To run AWS Emulator in a container 13
To remove the AWS Emulator image 14
Embedded mode 15
To add AWS Emulator to a Rust project 15
To use embedded mode in a test 16
To turn on authentication in embedded mode 17
Point your SDK at the emulator 18
How the emulator handles credentials and HTTPS 18
To point your SDK at a local emulator 18
Host names that your SDK changes 21
To check that you're using the emulator 22
Trusting the emulator certificate in each SDK 23
Checking your project 25

Command reference ... **26**
Synopsis ... 26
Options ... 26
Output ... 28
Control routes ... 28
Local files ... 29
Credentials and authentication ... 30
When the port is in use ... 32
Stopping the emulator ... 32
Fidelity ... **33**
What is fidelity? ... 33
How we determine the fidelity score ... 34
Reading the fidelity scores ... 34
Telling us about a difference ... 35
Versions and compatibility ... **36**
Version numbers ... 36
Finding and installing a specific version ... 37
Security fixes ... 37
What stays compatible ... 38
How fidelity changes affect your tests ... 38
Security ... **40**
Don’t use AWS Emulator to test security ... 40
Shared responsibility ... 41
What we do ... 41
What you do ... 42
Reporting a security issue ... 42
Telemetry ... **44**
What is collected ... 44
How telemetry is sent ... 45
Turning telemetry off ... 46
Use a command-line option ... 46
Use an environment variable ... 46
Use a configuration file ... 47
Use the embedded configuration API ... 47
How the settings combine ... 48
Related resources ... **49**

What is AWS Emulator?

AWS Emulator is a local emulator of the AWS APIs. You use it to test your software without calling AWS, on your own machine, on a development server, or in continuous integration (CI). You don't need an AWS account. Your requests go to the emulator, not to AWS. So you work with emulated resources instead of AWS resources. AWS Emulator has no additional service fee. You pay only for AWS resources when you deploy on AWS.

The emulator is a local program that serves an AWS API endpoint. It has no graphical interface. After you start the emulator, you set environment variables to point the AWS CLI or an AWS SDK at its endpoint. Then you run your code as usual. In embedded mode, you pass the emulator's HTTP client to your SDK instead.

The emulator doesn't send your requests, responses, or resource data to AWS. By default, it sends anonymous usage and error data, which you can turn off. For more information, see *Telemetry*.

On Linux, you install AWS Emulator from an RPM or Debian package for x86_64 or aarch64, and run it as a command-line tool. On macOS and Windows, you run it in Docker, from an image that you build with our Dockerfile. For tests written in Rust with the AWS SDK for Rust, you can also run the emulator inside your test process. For more information, see *Getting started*.

Topics

- Use cases
- Examples
- Limitations
- How AWS Emulator keeps improving
- Features of AWS Emulator
- Learn more

Use cases

AWS Emulator helps you to test your application code locally before deploying to AWS. You can use it for the following:

- **Local testing** – Test how your code works with AWS services, including calls that go across services, on your own machine.

- **Continuous integration and delivery (CI/CD)** – Run integration tests in your pipeline with no AWS account and no AWS resources.
- **Prototyping with AI coding agents** – Build and interact with applications locally that will run on AWS.

To learn which AWS APIs the emulator supports and how closely it matches AWS, see *Fidelity*. To check whether your own project is expected to work, see the section called "Checking your project".

Examples

The following examples use the AWS CLI with an emulator that runs on your machine. To set up your shell, see the section called "To point your SDK at a local emulator".

Example – Create a resource and get it again

The emulator keeps the resources that you create. The following AWS CLI command creates an Amazon S3 bucket named `amzn-s3-demo-bucket`.

```
$ aws s3api create-bucket --bucket amzn-s3-demo-bucket
{
    "Location": "/amzn-s3-demo-bucket"
}
```

When you list your buckets later, the emulator returns the bucket that you created.

```
$ aws s3api list-buckets --query 'Buckets[].Name'
[
    "amzn-s3-demo-bucket"
]
```

Example – Get the same validation error as on AWS

Amazon DynamoDB table names must be at least 3 characters long. The following AWS CLI command tries to create a table named `ab`. The emulator rejects the `CreateTable` call with the same error as AWS.

```
$ aws dynamodb create-table --table-name ab \
```

```
    --attribute-definitions AttributeName=id,AttributeType=S \
    --key-schema AttributeName=id,KeyType=HASH \
    --billing-mode PAY_PER_REQUEST
An error occurred (ValidationException) when calling the CreateTable operation: 1
 validation error detected: Value 'ab' at 'tableName' failed to satisfy constraint:
 Member must have length greater than or equal to 3
```

Limitations

AWS Emulator is for building and testing. Don't use it for production workloads. It has the following limitations:

- **Policies** – The emulator doesn't evaluate AWS Identity and Access Management (IAM) policies and resource-based policies the way AWS does. Don't use it to test security. For more information, see *Security*.
- **One Region and one account** – The emulator accepts any Region and account ID in a request, but it doesn't keep resources separate by Region or account. Some services share resources across all Regions, and others keep them per Region. Use one Region and one account for all requests to an emulator. To test more than one Region, run a separate emulator for each Region, each on its own port.
- **Fidelity** – How closely the emulator matches AWS varies by service and feature. For more information, see *Fidelity*.
- **Verified in one Region** – We check the emulator's behavior against AWS in the US West (Oregon) Region (`us-west-2`) only, except for global services, which we check in the US East (N. Virginia) Region (`us-east-1`). Differences in service and feature availability across Regions aren't reflected.
- **Your machine's resources** – The emulator runs on your machine and keeps all resources in memory. How much it can hold depends on your machine, and the emulator deletes all resources when it stops.
- **Test runs, not long-running use** – We haven't tested the emulator for long-running use or load testing. Start the emulator for your tests and stop it afterwards, or set `--idle-timeout` so that it stops by itself. Restarting also gives your tests a clean state.

Some services have other gaps. To find them, see the fidelity scores that we publish with each release on the AWS Emulator releases page on GitHub.

How AWS Emulator keeps improving

AI agents keep the emulator up to date with AWS. The emulator improves continuously, with updates many times a day, so it keeps up as AWS services change and grow. For what stays the same between releases, see *Versions and compatibility*.

Features of AWS Emulator

AWS Emulator includes the following features:

- An `aws-emulator` skill for AI coding agents, which installs, starts, and connects the emulator, and checks whether your project works on it. For more information, see the section called "With an AI coding agent".
- Works with the AWS CLI and AWS SDKs. You point their endpoint at the emulator with environment variables, and your code doesn't change.
- An embedded mode for tests written in Rust with the AWS SDK for Rust, with separate state for each emulator that you create
- Source code on GitHub, in the aws-emulator repository. For the license, see the section called "Review the license".
- Fidelity tests on GitHub, in the aws-emulator-evaluation-suite repository

Learn more

To learn more, see the following resources:

- *Fidelity*
- *Getting started*
- *Related resources*

Getting started with AWS Emulator

You can get started with AWS Emulator in four ways. Choose the path that fits how you work, your operating system, and how you test:

- **With an AI coding agent** – Install the `aws-emulator` skill, and ask your agent to install, start, and connect the emulator for you. You need an AI coding agent and the AWS CLI version 2.35.0 or later. For more information, see the section called “With an AI coding agent”.
- **Linux package** – Install the emulator from an RPM or Debian package and run it as a local process. Use this path on Linux. For more information, see the section called “Linux”.
- **Docker** – Build a container image that installs the emulator, and run it as a local container. Use this path on Windows and macOS, or on Linux when you prefer containers. For more information, see the section called “Docker”.
- **Embedded mode** – Run the emulator inside your Rust test process. Use this path when you want per-test isolated state and parallelism. For more information, see the section called “Embedded mode”.

The Linux and Docker paths start a local process that listens for HTTPS requests on port 9297 of `127.0.0.1`. You point the AWS CLI or your AWS SDK at that endpoint by setting environment variables. For more information, see the section called “Point your SDK at the emulator”.

Before you begin, complete the prerequisites in the section called “Setting up”.

Contents

- Setting up AWS Emulator
- Getting started with an AI coding agent
- Getting started on Linux
- Getting started with Docker
- Getting started with embedded mode
- Pointing your AWS SDK at AWS Emulator
- Checking whether your project works with AWS Emulator

Setting up AWS Emulator

Before you start, check the following. The prerequisites depend on how you run the emulator. You don't need an AWS account to download or run the emulator.

- **An AWS SDK or the AWS CLI** – You use the emulator through the AWS CLI or an AWS SDK, the same way that you use AWS.
- **AI coding agent path** – An AI coding agent, such as Kiro, Cursor, or Claude Code, and the AWS CLI version 2.35.0 or later to install the `aws-emulator` skill. For more information, see the section called "With an AI coding agent".
- **Linux package path** – A 64-bit Linux system on x86_64 or Arm (aarch64) with glibc 2.17 or later, such as Amazon Linux 2 or Amazon Linux 2023. It must be able to install RPM or Debian packages. To check your glibc version, run `ldd --version`. For more information, see the section called "Linux".
- **Docker path** – Any system that runs Docker, including Windows, macOS, and Linux. On macOS and Windows, use this path. The emulator runs natively only on Linux. For more information, see the section called "Docker".
- **Embedded mode** – A Rust toolchain with Cargo, a C compiler, and the AWS SDK for Rust. For more information, see the section called "Embedded mode".

Review the license

The source code is in the AWS Emulator repository on GitHub. The source code is under the CC0 1.0 Universal (CC0 1.0) public domain dedication, except the code in the `vendor/` directory, which is under the Apache License 2.0. For the full terms, see the license files in the repository.

Getting started with an AI coding agent

The `aws-emulator` skill teaches your AI coding agent how to work with AWS Emulator. With the skill, your agent can do the following for you:

- Install the emulator, using the Linux package or Docker, depending on your operating system
- Start the emulator, and point the AWS CLI or your AWS SDK at it
- Fix connection, certificate, and signing errors
- Check whether your project is expected to work on the emulator. For more information, see the section called "Checking your project".

The skill uses the same steps that this guide describes for the Linux package and Docker paths. Before your agent writes outside its working folders, such as when it runs `sudo` or edits your shell profile, it asks you first.

To install the skill

1. Check that you have the AWS CLI version 2.35.0 or later.

   ```
   aws --version
   ```

2. Install the skill. The following command installs it for all the AI coding agents that the AWS CLI finds on your machine, such as Kiro, Cursor, and Claude Code.

   ```
   aws agent-toolkit add-skill --skill-name aws-emulator --region us-east-1
   ```

   For more information, see Managing skills with the AWS CLI in the *Agent Toolkit for AWS User Guide*.

3. Ask your agent to set up the emulator. For example, ask "Install AWS Emulator and point my project at it."

Getting started on Linux

On Linux, you install AWS Emulator as a native package and run it as a local process. The package installs the emulator in `/opt/aws-emulator/` and adds the `aws-emulator` command to `/usr/bin/`, so the command is on your `PATH`.

First, check what you need in the section called “Setting up”.

To install AWS Emulator

1. Download and install the package for your system. The commands differ by package type and processor.

   RPM package (x86_64)

   ```
   sudo dnf install -y https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/latest/aws-emulator-x86_64.rpm
   ```

RPM package (aarch64)

```
sudo dnf install -y https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/
latest/aws-emulator-aarch64.rpm
```

Debian package (amd64)

```
curl -fsSLo aws-emulator.deb https://downloads.aws.com/aws-emulator/packages/aws-
emulator/LTS/latest/aws-emulator_amd64.deb
sudo apt-get install -y ./aws-emulator.deb
```

Debian package (arm64)

```
curl -fsSLo aws-emulator.deb https://downloads.aws.com/aws-emulator/packages/aws-
emulator/LTS/latest/aws-emulator_arm64.deb
sudo apt-get install -y ./aws-emulator.deb
```

On Amazon Linux 2, use `yum` for the RPM commands on this page.

The package also installs the license and notice files for AWS Emulator.

These commands install the latest release. To install a specific version, see the section called “Finding and installing a specific version”. To check the package before you install it, see the section called “To verify the package before you install it (optional)”.

2. Confirm the installation. The command prints the version. For what it shows, see the section called “Version numbers”.

```
aws-emulator -v
```

To verify the package before you install it (optional)

AWS signs each package. The signature is in a file next to the package, with the same name plus `.sig`. To check that a package comes from AWS and wasn’t changed, verify it before you install it. You need GnuPG (gpg), which most Linux distributions include.

1. Download the package and its signature file. The following example uses the RPM package for x86_64. For another package, change the file name in `PACKAGE_FILE`.

```
PACKAGE_FILE=aws-emulator-x86_64.rpm
curl -fsSLO https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/latest/
$PACKAGE_FILE
curl -fsSLO https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/latest/
$PACKAGE_FILE.sig
```

2. Import the public key.

```
curl -fsSLo aws-publishing.asc https://downloads.aws.com/keys/gpg/current.asc
gpg --import aws-publishing.asc
```

3. Check the fingerprint of the key.

```
gpg --fingerprint ssm-publishing@amazon.com
```

   The fingerprint must be 98BC F336 A3BE 53B7 34E9 67C5 2E5B F49B D940 F624. If it's different, don't install the package.

4. Verify the signature.

```
gpg --verify $PACKAGE_FILE.sig $PACKAGE_FILE
```

   The output includes Good signature from "SSM Publishing <ssm-publishing@amazon.com>". It also warns that the key isn't certified with a trusted signature. You can ignore this warning, because you checked the fingerprint. If the output says BAD signature, delete both files, and don't install the package.

5. Install the file that you verified.

   RPM package

```
sudo dnf install -y ./$PACKAGE_FILE
```

   Debian package

```
sudo apt-get install -y ./$PACKAGE_FILE
```

To start and stop AWS Emulator

1. Start the emulator.

   ```
   aws-emulator
   ```

   The emulator keeps running in that terminal. The startup line of its log gives the address and port that it serves on, and you can rely on it in scripts. For what the log contains, see the section called “Output”.

   By default, the emulator serves HTTPS on `127.0.0.1:9297`, so only your own machine can reach it. For all options, see *Command reference*.

2. In another terminal, set the environment variables for your SDK, and then run your code. For example, set `AWS_CONFIG_FILE` to `~/.aws/emulator/9297/config` and `AWS_SHARED_CREDENTIALS_FILE` to `~/.aws/emulator/9297/emulator_credentials`. For the full list, see the section called “To point your SDK at a local emulator”. For Java, also see the section called “To trust the emulator certificate in Java”.
3. To stop the emulator, press **Ctrl+C** in its terminal, or send the process a `SIGTERM` signal. The emulator stops cleanly and deletes the files that it created in `~/.aws/emulator/<port>/`. It also deletes all emulated resources, because it keeps them only in memory.

To use a different port

If another program already uses port 9297, the emulator can’t start. For the error that it prints, see the section called “When the port is in use”. To fix this, choose another port when you start the emulator.

- To use a specific port, run `aws-emulator -p <port>`. For example, run `aws-emulator -p 9300`.
- To let the operating system choose a free port, run `aws-emulator -p 0`. Then read the port from the `endpoint_url` field of the startup line in the log.

You can run several emulators side by side. Give each one a different port with `-p`, or use `-p 0` for each one. Each emulator keeps its own files in `~/.aws/emulator/<port>/`, and its own resources. When you set the environment variables for an emulator, use that emulator’s port in each one.

To limit the memory of the emulator on Linux

The emulator keeps emulated resources in memory. It uses about 160 MiB when idle and grows as your tests create resources. You can cap its memory with a systemd scope. This needs a systemd user session on cgroup v2. Amazon Linux 2 and many containers don't have one. On those systems, use Docker instead, as described in the section called "To run AWS Emulator in a container".

1. Start the emulator with the following `systemd-run` command. It sets a limit of 2 GiB, and `MemorySwapMax=0` keeps the emulator out of swap.

   ```
   systemd-run --user --scope -p MemoryMax=2G -p MemorySwapMax=0 aws-emulator
   ```

   The emulator runs in the foreground, as it does without the limit.

2. If the emulator goes over the limit, the kernel stops it with exit code 137. You lose the emulated resources. To start again with empty state, run the command again.

To uninstall AWS Emulator

1. Stop any running emulator.
2. Remove the package.

   RPM systems

   ```
   sudo dnf remove -y aws-emulator
   ```

   Debian systems

   ```
   sudo apt-get remove -y aws-emulator
   ```

3. (Optional) Delete the files the emulator keeps in your home directory.

   > **Important**
   >
   > This command deletes all the files that the emulator keeps in your home directory. Do this only when you're done with the emulator.

```
rm -rf ~/.aws/emulator
```

Getting started with Docker

On any system that runs Docker, including Windows and macOS, you can run the emulator in a Docker container. We don't publish a prebuilt image. You build one from a short Dockerfile that installs the Linux package.

First, install Docker. For the other prerequisites, see the section called "Setting up".

To build the AWS Emulator image

1. In an empty directory, create a file named `Dockerfile` with the following content.

   ```
   FROM public.ecr.aws/amazonlinux/amazonlinux:2023-minimal
   ADD https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/latest/aws-emulator-x86_64.rpm /tmp/aws-emulator.rpm
   RUN rpm -i /tmp/aws-emulator.rpm && rm /tmp/aws-emulator.rpm \
       && mkdir -p /home/emulator && chown 1000 /home/emulator
   USER 1000
   ENV HOME=/home/emulator
   EXPOSE 9297
   ENTRYPOINT ["aws-emulator", "-b", "0.0.0.0"]
   ```

   On computers with Apple silicon or other Arm processors, use the Arm package instead. In the ADD line, replace `aws-emulator-x86_64.rpm` with `aws-emulator-aarch64.rpm`.

   The `-b 0.0.0.0` option makes the emulator listen on all addresses inside the container, so Docker can forward its port. The `docker run` command in the next procedure opens the port only to your own machine.

2. Build the image.

   ```
   docker build -t aws-emulator .
   ```

To run AWS Emulator in a container

1. Create the directory that the container shares with your machine. On Linux, if you skip this step, Docker makes the directory owned by root, and your SDK can't read it.

   macOS and Linux

   ```
   mkdir -p ~/.aws/emulator
   ```

   Windows PowerShell

   ```
   New-Item -ItemType Directory -Force "$env:USERPROFILE\.aws\emulator"
   ```

2. Start the container.

   macOS and Linux

   ```
   PORT=9297
   docker run --rm --memory=2g --memory-swap=2g -p 127.0.0.1:$PORT:$PORT \
     -e HOME="$HOME" -v "$HOME/.aws/emulator:$HOME/.aws/emulator" \
     aws-emulator -p $PORT
   ```

   Windows PowerShell

   ```
   $port = 9297
   docker run --rm --memory=2g --memory-swap=2g -p "127.0.0.1:${port}:${port}" `
     -v "$env:USERPROFILE\.aws\emulator:/home/emulator/.aws/emulator" `
     aws-emulator -p $port
   ```

   The `-v` option lets your SDK read the credentials and certificate that the emulator creates. It also keeps the telemetry ID between runs, so each run isn't counted as a new install. On Linux, also add `--user "$(id -u):$(id -g)"` after `docker run` so the files belong to you. To turn off telemetry in the container, see the section called "Turning telemetry off".

   On macOS and Linux, `-e HOME` makes the emulator use the same paths in the container as on your computer. So the paths in its `config` file and startup line are correct on your computer. On Windows, they're paths in the container.

The `--memory` and `--memory-swap` options limit the emulator to 2 gibibytes (GiB) of memory, with no swap. If the container goes over the limit, it stops with exit code 137, and you lose the emulated resources.

The startup message shows paths inside the container, such as `/home/emulator/.aws/emulator/config.toml`. Because of the `-v` option, that file is `config.toml` in the `.aws/emulator` folder that you mounted. Edit it there, if needed.

Use `-p 127.0.0.1:9297:9297`, not `-p 9297:9297`. The short form opens the emulator to your whole network. Other computers could then reach it.

To pass other emulator options, add them at the end of the command. For all options, see *Command reference*.

3. In another terminal, set the environment variables for your SDK, as shown in the section called "To point your SDK at a local emulator".
4. Run your code.
5. To stop the emulator, press **Ctrl+C** in its terminal. The emulator deletes its files in `~/.aws/emulator/<port>/`, and Docker removes the container. You lose the emulated resources, because the emulator keeps them in memory.

To use another port, change `PORT` (`$port` in PowerShell). The command uses the port on both sides of Docker's `-p` option and in the emulator's `-p` option. They must match, because the emulator writes its port into the files that it creates.

To run several emulators at the same time, run the command in a separate terminal for each one, with a different port. Then use that emulator's port in the environment variables for your SDK.

To keep each emulator's credentials and certificate, change the port in all three places. If you change only the port on your computer, as in `-p 127.0.0.1:9300:9297`, the containers write to the same subdirectory of `~/.aws/emulator`. They then replace and delete each other's files.

Don't use the emulator's `-p 0` option in a container. Docker can't forward a port that the emulator picks when it starts.

To remove the AWS Emulator image

1. Stop each emulator that's running. Press **Ctrl+C** in its terminal.

2. Remove the image.

```
docker image rm aws-emulator
```

3. (Optional) Delete the shared directory.

> **Important**
>
> This step deletes all the files that the emulator keeps in your home directory. Do this only when you're done with the emulator.

macOS and Linux

```
rm -rf ~/.aws/emulator
```

Windows PowerShell

```
Remove-Item -Recurse -Force "$env:USERPROFILE\.aws\emulator"
```

Getting started with embedded mode

Embedded mode runs the emulator in the same process as your tests. Each test gets its own emulator and its own data. This lets you run tests in parallel. Your code calls the emulator directly, not over the network. You don't set an endpoint URL or a certificate.

Embedded mode works with the AWS SDK for Rust. Telemetry is on by default in embedded mode, as it is when the emulator runs on its own. To turn it off, see *Telemetry*.

To add AWS Emulator to a Rust project

Add the `aws-emulator-embedded` crate to the `[dev-dependencies]` section of your `Cargo.toml` file. This keeps the emulator out of the binary that you ship. You get the crate from GitHub. In the following example, `[dependencies]` holds the AWS SDK that your code uses. `[dev-dependencies]` adds the emulator.

```
[dependencies]
aws-config = "1"
```

```
aws-sdk-s3 = "1"

[dev-dependencies]
aws-emulator-embedded = { git = "https://github.com/aws/aws-emulator", rev =
 "<commit>" }
tokio = { version = "1", features = ["macros", "rt-multi-thread"] }
```

Replace `<commit>` with the commit of the release that you want to use. Each release on the AWS Emulator releases page on GitHub names its commit. Pin to a commit rather than a tag, because a commit can't move. For more information about versions, see *Versions and compatibility*.

To use embedded mode in a test

1. Create an `AwsEmulatorEmbedded` object with `AwsEmulatorEmbedded::new(EmbeddedOptions::default())`. You can also call `AwsEmulatorEmbedded::default()`, which does the same thing.
2. When you build your SDK configuration, pass `emulator.http_client()` to `.http_client()`.
3. Set the Region to `us-east-1`, and pass `emulator.aws_credentials_provider()` to `.credentials_provider()`.
4. Create a service client from that configuration, and run your test.

Set the Region and the credentials provider as shown. If you leave them out, the SDK looks for them in your environment. If it doesn't find them, your test fails.

The following example creates an Amazon S3 bucket, writes an object, and reads it back.

```
use aws_emulator_embedded::{AwsEmulatorEmbedded, EmbeddedOptions};
use aws_sdk_s3::primitives::ByteStream;

#[tokio::test]
async fn put_then_get_returns_the_body() {
    // Start an emulator for this test only. Its state is not shared with other tests.
    let emulator = AwsEmulatorEmbedded::new(EmbeddedOptions::default());

    // Send SDK requests to the emulator instead of to AWS.
    let config = aws_config::defaults(aws_config::BehaviorVersion::latest())
        .http_client(emulator.http_client())
        .region(aws_config::Region::new("us-east-1"))
```

```
        .credentials_provider(emulator.aws_credentials_provider())
        .load()
        .await;

    let s3 = aws_sdk_s3::Client::new(&config);

    // Use the client as you would against AWS.
    s3.create_bucket().bucket("emulator-demo-bucket").send().await.unwrap();
    s3.put_object()
        .bucket("emulator-demo-bucket")
        .key("k")
        .body(ByteStream::from_static(b"hi"))
        .send()
        .await
        .unwrap();

    let got = s3.get_object().bucket("emulator-demo-
bucket").key("k").send().await.unwrap();
    assert_eq!(got.body.collect().await.unwrap().to_vec(), b"hi");
}
```

Each emulator that you create has its own state. All clients that you build from one emulator share its state. Two emulators never share state.

To turn on authentication in embedded mode

Authentication is off by default in embedded mode. The emulator serves any request, signed or not. The credentials provider gives the SDK placeholder keys.

To make the emulator check signatures, set `Authentication::On` in `EmbeddedOptions`. Then API requests must be signed with the emulator's credentials, or with access keys that you create in the emulator through IAM or AWS STS. Operations that AWS serves without a signature don't need one. The emulator rejects other requests. In embedded mode, the emulator doesn't write a credentials file.

```
use aws_emulator_embedded::{Authentication, AwsEmulatorEmbedded, EmbeddedOptions};

let emulator = AwsEmulatorEmbedded::new(
    EmbeddedOptions::default().authentication(Authentication::On),
);
```

Keep passing `emulator.aws_credentials_provider()` to your SDK configuration. With authentication on, the provider supplies the emulator's credentials, and the SDK signs requests with them.

Pointing your AWS SDK at AWS Emulator

To use AWS Emulator, you configure the AWS CLI or your AWS SDK to send requests to the emulator instead of to AWS. When you run the emulator as a local process (Linux or Docker), you set environment variables. When you run it embedded in Rust tests, you replace the SDK HTTP client. For embedded mode, see the section called "Embedded mode".

How the emulator handles credentials and HTTPS

When you run AWS Emulator as a local process, by default it requires API requests to be signed with Signature Version 4 (SigV4), like AWS does. Signature checks protect you from requests that a web page or another local program sends to the emulator without your knowledge. For which keys the emulator accepts and the errors that it returns, see the section called "Credentials and authentication". For how this protects your machine, see the section called "What we do".

When the emulator starts, it generates a new access key ID and secret access key. It writes them, a `config` file for your SDK, and its certificate (`ca.pem`, and `truststore.p12` for Java) to a subdirectory of `~/.aws/emulator/` named for its port, for example `~/.aws/emulator/9297/`. The keys work only with this run of the emulator. For what each file contains and when it's deleted, see the section called "Local files".

To turn off signature checks, start the emulator with `--auth off` (`-a off`). The emulator then serves every request and doesn't write the `emulator_credentials` file. Before you use this option, see the section called "What you do".

To point your SDK at a local emulator

1. Start the emulator. For more information, see *Getting started*.
2. In the shell where you run your code, set the following environment variables. If you use a port other than 9297, replace 9297 in each value.

   macOS and Linux

   ```
   export AWS_ENDPOINT_URL=https://127.0.0.1:9297
   export AWS_SHARED_CREDENTIALS_FILE=~/.aws/emulator/9297/emulator_credentials
   ```

```
export AWS_CONFIG_FILE=~/.aws/emulator/9297/config
export AWS_CA_BUNDLE=~/.aws/emulator/9297/ca.pem
export AWS_EC2_METADATA_DISABLED=true
export AWS_DISABLE_HOST_PREFIX_INJECTION=true
export AWS_REGION=us-east-1
export AWS_DEFAULT_REGION=us-east-1
```

Windows PowerShell

```
$env:AWS_ENDPOINT_URL = "https://127.0.0.1:9297"
$env:AWS_SHARED_CREDENTIALS_FILE = "$env:USERPROFILE\.aws\emulator
\9297\emulator_credentials"
$env:AWS_CONFIG_FILE = "$env:USERPROFILE\.aws\emulator\9297\config"
$env:AWS_CA_BUNDLE = "$env:USERPROFILE\.aws\emulator\9297\ca.pem"
$env:AWS_EC2_METADATA_DISABLED = "true"
$env:AWS_DISABLE_HOST_PREFIX_INJECTION = "true"
$env:AWS_REGION = "us-east-1"
$env:AWS_DEFAULT_REGION = "us-east-1"
```

`AWS_EC2_METADATA_DISABLED` stops your SDK from asking the Amazon EC2 instance metadata service for credentials. This keeps your SDK from using real credentials when you test on an EC2 instance.

`AWS_DISABLE_HOST_PREFIX_INJECTION` stops the AWS CLI and the AWS SDK for Python (Boto3) from adding a prefix to the host name of the endpoint. For other SDKs, see the section called “Host names that your SDK changes”.

Setting `AWS_CONFIG_FILE` replaces your own AWS config file for that shell. Settings that you keep there, such as retry or Amazon S3 settings, don’t apply while you test against the emulator. The emulator writes its `config` file again each time it starts, so don’t add your settings to it. Instead, copy the file, add your settings to the copy, and set `AWS_CONFIG_FILE` to the copy. If you change the emulator’s port, make the copy again.

Use one Region for all your requests. To test several Regions, run one emulator for each Region. For more information, see the section called “Limitations”.

`AWS_CA_BUNDLE` makes the AWS CLI, Boto3, and the AWS SDK for Go trust the emulator’s certificate. It also works when the emulator runs in Docker, where the certificate path in `config` is a path in the container.

3. Unset any variables that can supply real AWS credentials. Environment variables take precedence over the credentials file. If you leave them set, your SDK signs requests with your real credentials, and the emulator rejects those requests.

   macOS and Linux

   ```
   unset AWS_ACCESS_KEY_ID AWS_SECRET_ACCESS_KEY AWS_SESSION_TOKEN AWS_PROFILE
   ```

   Windows PowerShell

   ```
   Remove-Item Env:AWS_ACCESS_KEY_ID, Env:AWS_SECRET_ACCESS_KEY, Env:AWS_SESSION_TOKEN,
    Env:AWS_PROFILE -ErrorAction SilentlyContinue
   ```

4. If your SDK doesn't read AWS_CA_BUNDLE, set the variable for your language from the table in the section called "Trusting the emulator certificate in each SDK".

5. (Optional) Check that the emulator is ready. The health check returns HTTP 200 with the body ok. It uses the same port as the API, over HTTPS.

   ```
   curl --cacert ~/.aws/emulator/9297/ca.pem https://127.0.0.1:9297/__admin/health
   ```

   In a script, you can wait for the emulator before you run your tests. The following loop tries once per second, for up to 30 seconds.

   ```
   for i in $(seq 30); do
     curl -sf --cacert ~/.aws/emulator/9297/ca.pem https://127.0.0.1:9297/__admin/health
    && break
     sleep 1
   done
   ```

6. Run your code. Your SDK reads the variables that you set, so your code needs no endpoint or credentials. For example, the following Python code lists the Amazon S3 buckets in the emulator.

   ```
   import boto3

   s3 = boto3.client("s3")
   print(s3.list_buckets()["Buckets"])
   ```

> **Important**
>
> Make sure that your tests send requests to the emulator, not to AWS. If `AWS_ENDPOINT_URL` isn't set in the shell that runs your tests, your code calls AWS with whatever credentials it finds. Set the variables in the same script that runs your tests, and don't keep real credentials in the environment of your test jobs.

Host names that your SDK changes

The emulator serves all services on one endpoint, `https://127.0.0.1:9297`. For some requests, your SDK changes the host name of the endpoint before it sends the request. A request that goes to a different host name doesn't reach the emulator.

- **Host prefixes** – For some operations, the SDK adds a prefix to the host name. For example, it adds `data-` for AWS Cloud Map `DiscoverInstances`, `sync-` for AWS Step Functions `StartSyncExecution`, and `stream-` for Amazon CloudWatch Logs `StartLiveTail`. The request then goes to a host such as `data-127.0.0.1`, which doesn't exist. To prevent this, turn off host prefix injection in your SDK, as shown in the following table.
- **Amazon S3** – The endpoint is an IP address. So your SDK puts the bucket name in the path of each Amazon S3 request, not in the host name. The emulator doesn't accept requests that put the bucket name in the host name. Keep `127.0.0.1` in `AWS_ENDPOINT_URL`. If you use `localhost` instead, the AWS SDKs for JavaScript, Java, and Go put the bucket name in the host name by default.
- **Amazon S3 Control** – Your SDK always puts your account ID in the host name of an Amazon S3 Control request, such as `000000000000.127.0.0.1`. You can't turn this off, so Amazon S3 Control requests don't reach the emulator.

SDK	How to turn off host prefix injection
AWS CLI	Set `AWS_DISABLE_HOST_PREFIX_INJECTION` to `true`.
AWS SDK for Python (Boto3)	Set `AWS_DISABLE_HOST_PREFIX_INJECTION` to `true`, or create the client with `Config(inject_host_prefix=False)` .

SDK	How to turn off host prefix injection
AWS SDK for JavaScript v3	Create the client with `disableHostPrefix: true` .
AWS SDK for Java 2.x	Set `SdkAdvancedClientOption.DISABLE_HOST_PREFIX_INJECTION` to `true` in the client override configuration.

Set `AWS_DISABLE_HOST_PREFIX_INJECTION` only in the shell where you use the emulator. On AWS, operations that use a host prefix fail when the variable is set. For example, Cloud Map `DiscoverInstances` returns `UnknownOperationException`.

The following example uses the AWS CLI to find the instances of a Cloud Map service. Without `AWS_DISABLE_HOST_PREFIX_INJECTION`, the AWS CLI sends the request to `data-127.0.0.1`, and the request fails.

```
$ aws servicediscovery discover-instances --namespace-name my-namespace --service-name
 my-service
Could not connect to the endpoint URL: "https://data-127.0.0.1:9297/"
```

With `AWS_DISABLE_HOST_PREFIX_INJECTION` set to `true`, the request reaches the emulator, and the emulator returns the instances that you registered.

```
$ export AWS_DISABLE_HOST_PREFIX_INJECTION=true
$ aws servicediscovery discover-instances --namespace-name my-namespace --service-name
 my-service \
    --query 'Instances[].{InstanceId: InstanceId, Attributes: Attributes}'
[
    {
        "InstanceId": "my-instance",
        "Attributes": {
            "AWS_INSTANCE_IPV4": "10.0.0.1"
        }
    }
]
```

To check that you’re using the emulator

To confirm that your shell sends requests to the emulator, run the following command.

```
aws sts get-caller-identity
```

The emulator uses one account, `000000000000`. If you reach the emulator, the output shows `Account` as `000000000000` and `Arn` as `arn:aws:iam::000000000000:user/emulator`. If you see your own account ID, your requests go to AWS. Check the variables in the section called "To point your SDK at a local emulator".

Trusting the emulator certificate in each SDK

The emulator serves HTTPS with its own certificate. The AWS CLI and some SDKs read the certificate from `AWS_CA_BUNDLE`, which you set in the section called "To point your SDK at a local emulator". Others need an extra setting. The following table lists the setting for each client.

Client	Extra setting for HTTPS
AWS CLI version 2	None. Reads `AWS_CA_BUNDLE` .
AWS SDK for Python (Boto3)	None. Reads `AWS_CA_BUNDLE` .
AWS SDK for Go v2	None. Reads `AWS_CA_BUNDLE` .
AWS SDK for JavaScript v3 (Node.js)	`NODE_EXTRA_CA_CERTS` set to the path of `ca.pem`, for example `~/.aws/emulator/9297/ca.pem`
AWS SDK for Java 2.x	`JAVA_TOOL_OPTIONS` set to use the emulator's `truststore.p12` . For more information, see the section called "To trust the emulator certificate in Java".
AWS SDK for Rust	`SSL_CERT_FILE` set to the path of `ca.pem`, for example `~/.aws/emulator/9297/ca.pem`
AWS SDK for .NET	`SSL_CERT_FILE` set to the path of `ca.pem`, for example `~/.aws/emulator/9297/ca.pem`

The emulator creates a new certificate each time it starts. Settings that point to the path of `ca.pem` or `truststore.p12` keep working. If you copy or import the certificate, repeat that step after each start.

On Windows, you might need to import `ca.pem` into the Windows certificate store, depending on your SDK.

If your code must also reach other HTTPS servers, including AWS, see the section called “Reaching the emulator and other HTTPS servers from one process”.

To trust the emulator certificate in Java

Java doesn’t read `ca.pem`. Instead, the emulator writes the same certificate to `truststore.p12`, a PKCS12 trust store without a password, in the same folder as `ca.pem`. To use it, point the Java virtual machine (JVM) at it. For example, set `JAVA_TOOL_OPTIONS` in the shell that runs your tests. If you use a port other than 9297, replace 9297 in the path.

macOS and Linux

```
export JAVA_TOOL_OPTIONS="-Djavax.net.ssl.trustStore=$HOME/.aws/emulator/9297/
truststore.p12 -Djavax.net.ssl.trustStoreType=PKCS12"
```

Windows PowerShell

```
$env:JAVA_TOOL_OPTIONS = "-Djavax.net.ssl.trustStore=$env:USERPROFILE\.aws\emulator
\9297\truststore.p12 -Djavax.net.ssl.trustStoreType=PKCS12"
```

Every JVM that you start from that shell reads `JAVA_TOOL_OPTIONS`, including build tools such as Gradle and Maven. To limit the setting to your tests, pass the same options to the test JVM only. For example, use `jvmArgs` in a Gradle `test` task, or `argLine` in the Maven Surefire plugin.

This trust store replaces the JVM’s default trust store, so calls to other HTTPS servers, including AWS, fail with `PKIX path building failed`. To reach both, see the section called “Reaching the emulator and other HTTPS servers from one process”.

Reaching the emulator and other HTTPS servers from one process

Most of the settings in the section called “Trusting the emulator certificate in each SDK” replace the certificates that your code trusts by default, so your code then trusts only the emulator. This applies to `ca_bundle`, `AWS_CA_BUNDLE`, `SSL_CERT_FILE`, and the Java trust store. `NODE_EXTRA_CA_CERTS` in Node.js is the exception. It adds `ca.pem` to the certificates that Node.js trusts by default.

If your code must reach the emulator and other HTTPS servers, including AWS, from the same process, add `ca.pem` to the certificates that your code already trusts. For example, use a certificate bundle that contains your system's certificates and `ca.pem`, or a copy of your Java Development Kit (JDK) trust store that also contains `ca.pem`. The emulator creates a new certificate each time it starts, so update the trust store after each start.

Checking whether your project works with AWS Emulator

You can check which parts of your project you can rely on AWS Emulator for. The easiest way is the `aws-emulator` skill for AI coding agents. With the skill, your agent finds the AWS operations and resources that your project depends on. It checks them against the fidelity tests and fidelity scores for the emulator. It then tells you which parts of your project you can rely on the emulator for, and which ones might behave differently from AWS. If you find a behavior that differs from AWS, your agent can also draft an issue for you. It shows you the draft before it posts anything, so you can review it first.

To install the skill, see the section called "To install the skill". Then ask your agent, for example, "Will my app in this repo work on AWS Emulator?"

The fidelity tests don't cover every behavior yet. We're continuously adding new fidelity tests based on customer feedback and fidelity issues that we discover. If no fidelity test covers a behavior, the emulator might already handle it correctly, but no fidelity test confirms it yet. For more information about fidelity tests and fidelity scores, see *Fidelity*. To check a workflow yourself, without the skill, do the following:

1. List the AWS operations that your workflow calls.
2. Find the fidelity tests for those operations in the manifest file, `manifest.jsonl`, in the aws-emulator-evaluation-suite repository on GitHub.
3. Check the fidelity scores for those services on the releases page, for the release that you use. Each score shows how many of the service's fidelity tests pass.
4. For operations that no fidelity test covers, run your own tests against the emulator and against AWS, and compare the results.

If something acts differently from AWS, tell us. Your reports help us add fidelity tests that cover your scenario. For more information, see the section called "Telling us about a difference".

AWS Emulator command reference

The `aws-emulator` command starts an instance of AWS Emulator. You install it with the Linux package, or run it in a container that you build. For installation, see *Getting started*. The container takes the same options as the native command.

Topics

- Synopsis
- Options
- Output
- Control routes
- Local files
- Credentials and authentication
- When the port is in use
- Stopping the emulator

Synopsis

The command takes only options. It has no subcommands.

```
aws-emulator [options]
```

With no options, the emulator listens for HTTPS requests on `127.0.0.1:9297`. It serves the AWS APIs and its health route on that one port. It runs in the foreground until you stop it.

Options

The following table lists every option. Long options also accept the form `--option=value`, for example `--port=9300`. If you give an option that the emulator doesn't know, it prints an error and exits with code 2.

Option	Default	Description
`-b, --bind` *ip*	`127.0.0.1`	The IP address that the emulator listens on. The value must be an IP address, not a host name. The default

Option	Default	Description
		lets only your own machine reach the emulator. In a container, bind `0.0.0.0` so that Docker can forward the port. Use `0.0.0.0` only in a container, and publish the port on `127.0.0.1` .
`-p, --port` *port*	9297	The port for HTTPS requests, from 0 to 65535. Use `0` to let the operating system pick a free port. Then read the actual port from the startup line of the log. For more information, see the section called “Output”. To run several emulators side by side, give each one a different port.
`-a, --auth` *on|off*	on	Turns request authentication on or off. When it’s on, API requests must be signed with the keys of this run, except operations that AWS serves without a signature. When it’s off, the emulator serves every request and doesn’t write a credentials file. For more information, see the section called “Credentials and authentication”.
`-t, --telemetry` *on|off*	on	Turns telemetry on or off for this run. `off` always turns telemetry off. on doesn’t override an opt-out that you set in an environment variable or the emulator settings file (`config.toml`). For more information, see *Telemetry*.
`--idle-timeout` *seconds*	None	Stops the emulator after it receives no API call for the given number of seconds. The value is a whole number from 1 to 31536000 (one year). Use it in automated test runs so that a forgotten emulator doesn’t keep running. Without this option, the emulator runs until you stop it.
`-v, --version`	None	Prints the version, the release date, and the commit that the release was built from, and then exits.
`-h, --help`	None	Prints the list of options and the health route, and then exits.

Output

The emulator writes its log to standard output as JSON lines, one JSON object for each line. When it's ready, it writes one startup line, whose `message` field is `emulator server information`. The following table describes the fields in the startup line.

Field	Description
`endpoint_url`	The URL that your tools use to reach the emulator, for example `https://127.0.0.1:9297` .
`health_check_url`	The URL of the health route, for example `https://127.0.0.1:9297/__admin/health` .
`config_file`	The path of the config file that it wrote.
`ca_file`	The path of its certificate file (`ca.pem`).
`java_truststore_file`	The path of its trust store for Java (`truststore.p12`).

You can rely on this line and its fields. Find it by its `message` field, not by its position, because other lines can come before it. For example, if you start the emulator with `-p 0`, read the actual port from `endpoint_url`. For the files, see the section called "Local files". For how to use them with your tools, see the section called "Point your SDK at the emulator".

The emulator also writes one line of plain text to standard error. This telemetry notice says whether telemetry is on and how to turn it off. For more information, see *Telemetry*.

If the emulator listens on an address other than loopback, or if authentication is off, it also writes a warning line. This line says who can reach the emulator. The rest of the log isn't a stable interface. Its fields can change in any release, and so can the help text. To check whether the emulator is ready, use the health route. For more information, see the section called "Control routes".

Control routes

Besides the AWS APIs, the emulator serves one control route. It serves this route on the same port as the APIs, over HTTPS. There's no separate admin port.

Route	Description
`GET /__admin/health`	Returns HTTP 200 with the body ok when the emulator is working. Use it to wait for the emulator to start.

The following example checks the health of an emulator on the default port. It uses the certificate that the emulator wrote for that port.

```
curl --cacert ~/.aws/emulator/9297/ca.pem https://127.0.0.1:9297/__admin/health
```

To wait for the emulator in a script, use the loop in the section called “To point your SDK at a local emulator”.

The emulator doesn’t provide routes to reset state or to control state transitions. It keeps all state in memory. To start with empty state, restart the emulator.

Local files

When it starts, the emulator writes the following files to a subdirectory of `~/.aws/emulator/` named for its port, for example `~/.aws/emulator/9297/`.

File	Contents
`emulator_credentials`	The access key ID and secret access key for this run, in the format of an AWS shared credentials file. The emulator writes this file only when authentication is on.
`config`	A `[default]` profile in the format of an AWS config file. It sets `region` to `us-east-1`, and it sets `endpoint_url` and `ca_bundle` for this emulator. It also sets the Amazon S3 `signature_version` to `s3v4`.
`ca.pem`	The certificate that the emulator uses for HTTPS. The emulator creates a new certificate each time it starts.
`truststore.p12`	The same certificate as `ca.pem`, in a Public-Key Cryptography Standards #12 (PKCS12) trust store without a password, for Java.

File	Contents
	For more information, see the section called “To trust the emulator certificate in Java”.

The emulator creates each file so that only your user can read and write it (mode 0600). It deletes the files when it stops cleanly. A clean stop is **Ctrl+C**, `SIGTERM`, or an idle timeout set with `--idle-timeout`.

If the emulator crashes, the files stay. The next emulator that starts on the same port replaces them, so you don’t need to delete them yourself.

When you run the emulator in a container, the paths in `config` are paths inside the container. For the settings to use on your host, see the section called “To point your SDK at a local emulator”.

For the files that telemetry uses, see *Telemetry*.

Credentials and authentication

By default, the emulator requires API requests to be signed with Signature Version 4 (SigV4). Operations that AWS serves without a signature, such as AWS STS `AssumeRoleWithWebIdentity`, don’t need one. The emulator accepts both signed headers and presigned URLs. Each time it starts, it generates a new key pair. Currently, the keys have the following format:

- The access key ID is 20 characters long. It starts with `EMULATOR`, followed by 12 uppercase hexadecimal characters.
- The secret access key is 40 uppercase hexadecimal characters.

The keys work only with the emulator run that generated them. They don’t expire while the emulator runs. They stop working when the emulator stops, and they don’t work with an emulator on another port.

The emulator rejects unsigned requests and requests signed with other keys. Keys that you create in the emulator itself, through IAM (`CreateAccessKey`) or AWS STS, also work. Some services accept requests that aren’t signed with standard SigV4. The emulator doesn’t handle those requests the same way as AWS. For more information, see the section called “What we do”.

The emulator rejects a request with HTTP 403, or with HTTP 404 if it can't tell which service the request is for. The error depends on the reason, as shown in the following table.

Reason	Error
The request isn't signed	The error that the service returns on AWS, for example `InvalidClientTokenId` from AWS STS.
The request is signed with a key that the emulator doesn't accept	`AccessDenied`, with the message "Sign the request with the emulator credentials". This error also covers a malformed signature, a credential scope that names the wrong service, and an expired presigned URL.
The request uses a key that the emulator accepts, but the signature is wrong	`SignatureDoesNotMatch`. The secret key is wrong, or the body doesn't match the signature.
The emulator can't tell which service the request is for	HTTP 404 `UnknownService`. This often happens with an unsigned request, because the emulator usually reads the service from the signature. Sign the request with the emulator credentials, as described in the section called "To point your SDK at a local emulator".

The emulator rejects a request whose signed time is more than 15 minutes from its own clock, as AWS does. It also rejects presigned URLs after they expire.

Each emulator acts as one AWS account, `000000000000`. To check that your tools talk to the emulator, see the section called "To check that you're using the emulator".

If you start the emulator with `--auth off`, it serves every request, signed or not, and doesn't write a credentials file. Before you use this option, see the section called "What you do".

The size and format of the generated credentials can change in any release. Don't validate them in your code.

When the port is in use

If another process already uses the port, the emulator prints the following error, followed by an operating system error code. Then it exits with code 1.

```
emulator serve failed: server I/O error: Address already in use
```

The emulator writes no files in this case. The files of an emulator that already runs on that port stay as they are. To fix the error, choose another port with `-p`, or use `-p 0` to let the operating system pick a free port.

Stopping the emulator

The emulator runs in the foreground. To stop it, press **Ctrl+C** or send the process `SIGTERM`. If you set `--idle-timeout`, the emulator also stops by itself after that idle time.

In each case, the emulator stops cleanly. It sends its final telemetry batch if telemetry is on, and deletes its files for that port. The emulator keeps all emulated resources in memory, so they're gone when it stops.

Fidelity in AWS Emulator

AWS Emulator supports the APIs of AWS services that have an API model in the AWS API models repository on GitHub. How closely the emulator matches AWS for those APIs is called *fidelity*.

AWS offers a broad and powerful set of capabilities, and both the capabilities and the ways that builders use them keep growing, so we're continuously assessing and improving fidelity. We base that work on customer feedback and on differences that we discover ourselves. The next section defines the terms that we use to describe fidelity.

Topics

- What is fidelity?
- How we determine the fidelity score
- Reading the fidelity scores
- Telling us about a difference

What is fidelity?

Where the emulator implements an operation, it aims for full fidelity: the same results as AWS for valid calls, and the same errors for invalid ones. If it doesn't implement an operation yet, it returns HTTP 501 with the error code `NotImplemented`, and a test that calls it fails right away. We measure fidelity with fidelity tests.

A *fidelity test* is a short program that uses an AWS SDK to run a workflow against an AWS service and assert how AWS behaves: the result, the resulting state, and for invalid calls, the exact error. For example, a fidelity test might try to create an Amazon DynamoDB table with a name that's too short, and assert that the call is rejected with the error code and message that AWS returns. We build and verify fidelity tests in the US West (Oregon) Region (`us-west-2`), except for global services, which we verify in the US East (N. Virginia) Region (`us-east-1`). We publish the fidelity tests in the aws-emulator-evaluation-suite repository on GitHub, including the ones that the emulator doesn't pass yet.

Each fidelity test has labels that describe what it asserts, such as whether a call succeeds or AWS rejects it. One label says whether the test targets the *control plane* or the *data plane*. The control plane is the APIs that create, read or describe, update, delete, and list resources, such as creating an Amazon S3 bucket. The data plane is the primary function of a service, such as getting and putting

objects in an S3 bucket. For more information, see Control planes and data planes in the *AWS Whitepaper*. Today, the emulator matches AWS most closely for control plane APIs. Fidelity for data plane APIs, and support for infrastructure as code tools such as AWS CloudFormation, Terraform, and the AWS CDK, varies by service. For what each label means, see `docs/TAXONOMY.md` in the aws-emulator-evaluation-suite repository.

A *fidelity score* is, for each service, the number of fidelity tests that the emulator passes out of the total for that service, such as 40 of 50. We publish fidelity scores with each release of AWS Emulator.

To check whether your own project is expected to work on the emulator, see the section called "Checking your project".

How we determine the fidelity score

The emulator passes a fidelity test when its behavior matches AWS for every call in the workflow, including the status code, the resources that it creates, and other side effects. If AWS returns an error, the emulator passes only if it returns the same error.

We publish the fidelity tests in numbered versions. Later versions add tests, and can retire tests that become outdated.

The aws-emulator-evaluation-suite repository on GitHub holds the following:

- The fidelity tests
- A manifest file, `manifest.jsonl`, that lists the services and operations that each fidelity test calls, and its labels
- A small Python runner, `fidelity_tests_runner`, that runs the fidelity tests against AWS Emulator

Reading the fidelity scores

We publish the fidelity scores with each release, on the AWS Emulator releases page on GitHub. Each release names the AWS Emulator commit and the version of the fidelity tests that we used. Compare scores only when they use the same version of the fidelity tests.

A score is a snapshot. As the emulator improves and we add fidelity tests, both numbers grow. For example, 40 of 50 can become 45 of 60: the emulator now passes 5 more fidelity tests, and we added 10 new ones.

Fidelity tests that pass don't regress. The number of failing fidelity tests can still go up in the following situations:

- We add fidelity tests. A new fidelity test can find a difference that was already there. It shows up as a new failing test.
- We rebuild a fidelity test, or update it, when AWS changes how a service behaves. This can happen within a version of the fidelity tests.

We can also retire a fidelity test that becomes outdated. Then the total goes down.

A fidelity test that the emulator doesn't pass yet is a known gap that we publish on purpose and track. We keep improving fidelity, so check the scores with each release. To see which gaps affect your project, see the section called "Checking your project".

To reproduce the scores, or to check a newer build for the services that you use, you can run the fidelity tests yourself with `fidelity_tests_runner`, the Python runner in the aws-emulator-evaluation-suite repository. It runs the tests only against an emulator on your machine. For instructions, see the repository's README file.

Telling us about a difference

If the emulator acts differently from AWS, open an issue in the aws-emulator repository on GitHub. Include the following:

- Steps to reproduce, including the request that you sent
- The response that you got from the emulator
- What you expected

If you use the `aws-emulator` skill, you can also ask your AI coding agent to open the issue on GitHub for you. For more information, see the section called "Checking your project".

Versions and compatibility in AWS Emulator

We aim to keep breaking changes rare, so that your tests keep working when you upgrade. This topic explains version numbers, what you can rely on, and how fidelity fixes can change your test results.

Topics

- Version numbers
- Finding and installing a specific version
- Security fixes
- What stays compatible
- How fidelity changes affect your tests

Version numbers

AWS Emulator versions use the format `MAJOR.MINOR.PATCH`:

- **MAJOR** – Changes that can break how you start, configure, or connect to the emulator. Examples are a removed or renamed option, environment variable, or control route.
- **MINOR** – New features that don't break how you start, configure, or connect to the emulator, such as new options or new control routes.
- **PATCH** – Bug fixes and fidelity fixes.

To see your version, run the following command.

```
aws-emulator -v
```

The command prints three lines:

- `Version` – The version of the emulator
- `Release date` – The date of the release
- `Commit` – The GitHub commit for the build

Finding and installing a specific version

Packages are published under the following address. The latest packages are in the `latest/` folder.

```
https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/
```

For the list of published versions, see the following file.

```
https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/versions.json
```

To install a specific version, put the version in the package file name. Replace `<version>` with a version from the list:

- **RPM package (x86_64)** – `https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/aws-emulator-<version>-1-x86_64.rpm`
- **RPM package (aarch64)** – `https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/aws-emulator-<version>-1-aarch64.rpm`
- **Debian package (amd64)** – `https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/aws-emulator_<version>-1_amd64.deb`
- **Debian package (arm64)** – `https://downloads.aws.com/aws-emulator/packages/aws-emulator/LTS/aws-emulator_<version>-1_arm64.deb`

Install the package the same way as the latest release, as shown in the section called “To install AWS Emulator”.

We recommend that you use the latest version. If a new release changes your test results, see the section called “How fidelity changes affect your tests”.

Security fixes

We publish security fixes only in the latest release. Older releases don’t get fixes. To get them, upgrade to the latest release.

A security fix might need a breaking change, even within a major version. For example, an operation that works without credentials might start to need them.

What stays compatible

Within a major version, these stay the same:

- The `aws-emulator` command, its options, and their default values
- The environment variables in this guide
- The emulator control routes, such as the health check
- The package URLs on `downloads.aws.com`
- The startup line of the log and its fields, which give the address and port that the emulator serves on

Don't rely on the following. They can change in any release:

- The fields in log output, other than the startup line. The emulator writes its log as JSON lines, but the rest of the log isn't a stable interface.
- The content of the `-h` help text
- Files in the emulator's local directories, other than the ones in this guide
- The size and format of the credentials that the emulator generates
- What's in the installation package, other than the `aws-emulator` command and the license and notice files

How fidelity changes affect your tests

We test each release against the fidelity tests in the aws-emulator-evaluation-suite repository on GitHub. We publish the scores for each service with each release, with the version of the fidelity tests that we used. For more information, see the section called "Reading the fidelity scores".

- **Behavior that a passing fidelity test covers** – Fidelity tests that pass don't regress.
- **Behavior that no fidelity test covers** – A release can fix a response so it matches AWS. If your tests relied on the earlier response, they can start to fail.

The fidelity tests don't cover every behavior yet. We're continuously adding new fidelity tests based on customer feedback and fidelity issues that we discover. To learn how to check which behavior they cover, see the section called "Checking your project". To limit the effect of these changes:

- If new releases change your test results, install a specific version, as described in the section called “Finding and installing a specific version”. Then a new release changes your results only when you choose to upgrade. Test a new version before you upgrade.
- Write tests that expect what AWS does, not what the emulator does today. As the emulator’s fidelity improves, it moves closer to AWS. For the APIs that matter to your app, also run your tests against AWS. For more information, see *Fidelity*.

If a patch release changes your test results, either the emulator now matches AWS more closely, or it’s a bug that you can report. Read the release notes on the AWS Emulator releases page on GitHub before you move to a new minor or major version.

If a release makes the emulator act differently from AWS, tell us. For more information, see the section called “Telling us about a difference”.

Security in AWS Emulator

AWS Emulator is a tool for building and testing on your own machine. It doesn't run in the AWS Cloud, so security works differently from security for an AWS service. This topic tells you what not to test, how the emulator helps protect your environment, and what you must do.

Topics

- Don't use AWS Emulator to test security
- Shared responsibility
- Reporting a security issue

Don't use AWS Emulator to test security

> **Important**
>
> Don't use AWS Emulator to test whether your application is secure. Passing tests in the emulator don't show that your permissions are correct or safe.

Don't use the emulator to test the following:

- IAM policies and permissions boundaries
- Resource-based policies, such as bucket policies
- AWS KMS key policies
- Access across AWS accounts or across AWS Regions
- Any other security control

Some services in the emulator evaluate IAM policies for IAM users and roles that you create in the emulator, but not the same way as AWS. The emulator never denies a request signed with its own key. It doesn't check AWS KMS key policies or Amazon S3 bucket policies. It also doesn't support calls across accounts or Regions. So a request that AWS denies might succeed in the emulator. A request that AWS allows might fail.

The emulator helps you catch many issues early. Still, test your security settings and your whole app on real AWS before you go to production.

Shared responsibility

We share the work of security with you. We build the emulator to run safely on your machine. You decide where it runs, what data you put in it, and how you test before production.

What we do

We built the emulator to help protect your environment in the following ways:

- **By default, it listens only on your own machine.** The emulator listens on `127.0.0.1:9297`. Some resources that you create, such as databases, get an endpoint of their own on another port of `127.0.0.1`. So only your own machine can reach the emulator. The Docker command in this guide also opens its ports only to your own machine.
- **It makes its own credentials.** When it starts, the emulator makes a new access key ID and secret access key. It writes them to `~/.aws/emulator/<port>/emulator_credentials`, a file that only your user can read.
- **Its keys work only with that run of the emulator.** By default, when the emulator runs as a local process, API requests must be signed with the keys from this run, except operations that AWS serves without a signature. The emulator also accepts keys that you create through its own IAM or STS. It rejects unsigned requests and requests signed with any other keys. Each time the emulator starts, it makes new keys, so keys from an earlier run don't work.

  > **Note**
  >
  > Some services accept requests that aren't signed with standard SigV4. The emulator doesn't handle these requests the same way as AWS. If you use these services, make sure that the emulator is private, as described in the section called "What you do".

- **It cleans up after itself.** When the emulator stops cleanly, it deletes the files that it wrote for its port, including the credentials file. It keeps emulated resources in memory, so they're gone when it stops.
- **It keeps your data local.** The emulator doesn't send your requests, responses, or resource data to AWS. Telemetry never includes what's in your requests or responses, your code, your credentials, or personal data. For more information, see *Telemetry*.

What you do

You must do the following:

- **Keep the emulator private.** Run it on your own machine, and don't run it on a machine that other people can sign in to: other users and programs on the same machine can reach the emulator's ports. Ports for resources with their own endpoint don't check request signatures, so only the resource's own sign-in, such as a password, protects them. Don't open the emulator's ports to the internet. In Docker, publish them on `127.0.0.1`, as shown in the section called "Docker".
- **Keep authentication on.** If you start the emulator with `--auth off`, it serves any request, signed or not. The option exists for tools that can't sign requests and for a simpler local setup. Use it at your own risk, and only on a machine that nobody else can reach.
- **Keep real credentials out.** Don't use real AWS credentials with the emulator. Before you run your tests, unset any variables that hold real credentials. For more information, see the section called "Point your SDK at the emulator".
- **Keep real data out.** Use only test data. Don't put real customer data or production data in the emulator.
- **Run only code that you trust.** When the emulator runs code for you, such as function code, it doesn't isolate that code from your machine. The code runs with your user's permissions and can read and change your files. Make sure that the code that you pass to the emulator doesn't do anything that you don't intend.
- **Make sure your tests reach the emulator.** If `AWS_ENDPOINT_URL` isn't set, your code calls AWS with whatever credentials it finds. Set the variables in the same script that runs your tests.
- **Keep the emulator up to date.** We publish security fixes only in the latest release. To get them, upgrade to the latest release. For more information, see the section called "Security fixes".
- **Test on real AWS.** Before production, test your app and its security settings on real AWS.

Reporting a security issue

If you find a security issue in AWS Emulator, don't open a public issue on GitHub. Instead, report it to us in one of the following ways:

- Submit it to the AWS Vulnerability Disclosure Program on HackerOne.
- Send an email to aws-security@amazon.com.

For more information, see the AWS vulnerability reporting page.

Telemetry in AWS Emulator

AWS Emulator collects usage data and reports it to AWS. This data never includes what's in your requests or responses, your code, your credentials, or personal data. It also can't be used to identify you or tie activity to your account or organization. We look at it only in aggregate, to find gaps in the emulator and spot issues that affect many users. Telemetry is on by default. To learn what data is collected and how to turn telemetry off, see the section called "What is collected" and the section called "Turning telemetry off".

The emulator sends telemetry over Transport Layer Security (TLS) to an endpoint that AWS owns. We don't share it with third parties, and we keep it for up to 90 days.

What is collected

The emulator sends the following kinds of data. Every value is a random ID, a number, or a value from a fixed list. The emulator never sends free text.

- **Installation and run IDs** – A random ID for the installation and a random ID for each run. The emulator makes them up. They don't come from your account, hardware, or network. We use them only to count installations and to group the events of one run.
- **Runtime environment** – Your operating system and CPU type; the AWS SDK or tool that sent the request, its version, and the version of its programming language; the AWS Region that requests use; how you installed the emulator; which optional features are on; whether the emulator runs on its own or embedded, and in a container or in CI; and which AI coding assistant drives the emulator, if any.
- **Product version** – The emulator version, the version last used by this installation, and whether this is the first run.
- **Usage and outcomes** – Counts for each AWS operation, including how many times it was called and how each call ended: success, client error, server error, or not implemented yet (HTTP 501). Errors are reported as a category, such as `ValidationError` or `Throttling`, never as message text. Response time is sent as the minimum, maximum, and total for each operation.
- **Performance** – Peak memory use (as a range), throughput, peak concurrency, startup outcome, and startup time.
- **Crash type** – If the emulator process crashes, the kind of crash, such as a panic.
- **Project ID** – An ID for the build of the emulator that you run. Everyone on the same release has the same value, so it doesn't identify you.

The emulator checks every value against a fixed list before it sends it:

- Regions must match a known AWS Region.
- Operations must exist in the emulated services.
- Errors, SDK names, and versions are sent as a category. If the emulator doesn't know an error, it sends `Other`. If it doesn't know any other value, it sends `unknown`.

The receiving service checks every value again. It stores a value outside these lists as `unknown`, or it drops the record.

The emulator doesn't collect any of the following:

- What's in your requests or responses, including parameters and payloads
- Resource names, Amazon Resource Names (ARNs), keys, or file paths
- Values from environment variables or config files
- Credentials or your code
- The raw User-Agent string, error messages, or stack traces
- Account IDs or personal data

We use your IP address only at the network edge, to protect the endpoint from abuse. It isn't in the telemetry records. We keep it for a short time in edge access logs. We can't link it to any telemetry ID.

How telemetry is sent

AWS Emulator adds up activity in memory. It sends a batch every five minutes, and once more when it stops. If the endpoint can't be reached, the emulator queues batches locally and retries later. It discards queued data that is older than 14 days.

When you start the emulator, it prints one line that says whether telemetry is on and how to change it. The emulator sends nothing before it prints this notice. When the emulator is embedded, it writes the notice to the emulator log instead of the console.

Turning telemetry off

You can turn telemetry off in four ways: an option, an environment variable, the emulator settings file, or the embedded API. Any one of them is enough. For details, see the following sections.

When telemetry is off, the emulator sends nothing. It also deletes the installation ID and any queued data, and it doesn't make a new ID.

The emulator keeps the installation ID and the queue in `~/.aws/emulator/telemetry/` on Linux and macOS, and in `%USERPROFILE%\.aws\emulator\telemetry\` on Windows.

Use a command-line option

Start the emulator with the `--telemetry` option (short form `-t`) set to `off`. This setting applies only to that emulator process.

```
aws-emulator --telemetry off
```

In a container, add `--telemetry off` after the image name in the `docker run` command.

Use an environment variable

Set `EMULATOR_TELEMETRY` to `off` before you start the emulator. This setting applies to every emulator that you start from that shell.

`EMULATOR_TELEMETRY` accepts on, `true`, or `1` to turn telemetry on, and `off`, `false`, or `0` to turn it off. Values aren't case sensitive. Any other value turns telemetry off, and the startup notice shows a warning.

Bash

```
export EMULATOR_TELEMETRY=off
aws-emulator
```

Docker – Pass the variable to the container with the `-e` option. On Linux, also add `--user "$(id -u):$(id -g)"` after `docker run`, as described in the section called "Docker".

```
docker run --rm --name aws-emulator -e EMULATOR_TELEMETRY=off \
  --memory=2g --memory-swap=2g \
```

```
  -p 127.0.0.1:9297:9297 \
  -v "$HOME/.aws/emulator:/home/emulator/.aws/emulator" \
  aws-emulator
```

Docker in Windows PowerShell

```
docker run --rm --name aws-emulator -e EMULATOR_TELEMETRY=off `
  --memory=2g --memory-swap=2g `
  -p 127.0.0.1:9297:9297 `
  -v "$env:USERPROFILE\.aws\emulator:/home/emulator/.aws/emulator" `
  aws-emulator
```

Setting the variable in your own shell doesn't turn telemetry off inside a container. For more information about the Docker command, see the section called "Docker".

Use a configuration file

To turn telemetry off for every run on a machine, add these lines to the emulator settings file, `config.toml`. The file is `~/.aws/emulator/config.toml` on Linux and macOS, and `%USERPROFILE%\.aws\emulator\config.toml` on Windows. To use a different file, set the `EMULATOR_CONFIG` environment variable to its path.

```
[telemetry]
enabled = false
```

Use the embedded configuration API

In embedded mode, turn off telemetry in the options that you pass when you create the emulator.

```
use aws_emulator_embedded::{AwsEmulatorEmbedded, EmbeddedOptions, Telemetry};

fn main() {
    let emulator =
 AwsEmulatorEmbedded::new(EmbeddedOptions::default().telemetry(Telemetry::Off));
    let _http_client = emulator.http_client();
}
```

You might build a tool that includes the emulator, such as a test framework that other people run. If you do, tell your users about telemetry and give them a way to turn it off. You can also ship your tool with telemetry off.

How the settings combine

An off setting always wins. Telemetry is off if any of these is true:

- You start the emulator with `--telemetry off`.
- `EMULATOR_TELEMETRY` is set to a value that turns telemetry off.
- The emulator settings file sets `enabled = false`.
- The tool that embeds the emulator turns it off.

A tool that embeds the emulator can’t turn telemetry on if you turned it off.

To turn telemetry back on, remove every setting that turns it off, and then restart the emulator.

Related resources

To learn more, see these links:

- AWS Emulator on GitHub – Get the source code.
- Product page on the AWS website – Learn what the emulator does. Read answers to common questions.
- Releases on GitHub – See each release and its fidelity scores.
- Issues on GitHub – Report a bug, or a case where the emulator doesn't act like AWS.
- aws-emulator-evaluation-suite on GitHub – Get the fidelity tests and run them yourself.
- Agent Toolkit for AWS User Guide – Install the `aws-emulator` skill for AI coding agents.