AWS Emulator

Version 1.0 · verified 2026-11-12

A local, in-memory stand-in for the AWS APIs — install it, point the CLI and SDKs at it, wire it into tests and CI, and know where it differs from AWS.

Video walkthrough — scan the code to watch.

Contents

Mental model 2

Pick a run mode 2

Install and start 4

Point the CLI and SDKs at it 7

Trust its certificate 9

Host-name traps 11

Add it to an existing project 12

CI and parallel runs 15

Embedded mode for Rust 17

Fidelity, versions and limits 19

Further reading 21

Mental model

AWS Emulator is a local program that serves the AWS APIs on one HTTPS endpoint, `https://127.0.0.1:9297`. Your code does not change: you point the AWS CLI or SDK at that endpoint with environment variables, and requests that would have gone to AWS go to the emulator instead. Each run is one account (`000000000000`), keeps every resource in memory, and loses all of it when it stops. It is generated from the same API models as the SDKs, so an implemented operation returns AWS's results and errors, an unimplemented one returns `501 NotImplemented`, and IAM policies are not enforced as on AWS.

Pick a run mode

Mode	Use when
Linux package	Linux x86_64 or aarch64, glibc ≥ 2.17
Docker	macOS and Windows, or Linux by choice

Mode	Use when
Embedded	Rust tests with the AWS SDK for Rust
Agent skill	A coding agent installs and wires it

The emulator runs natively only on Linux. AWS publishes no container image: on macOS and Windows you build one from a short Dockerfile that installs the Linux package. You need no AWS account, and there is no fee.

The skill route needs AWS CLI 2.35.0 or later. It installs, starts and connects the emulator for you, and checks whether your project's calls are covered.

```
aws agent-toolkit add-skill \
  --skill-name aws-emulator --region us-east-1
# then ask your agent:
# "Install AWS Emulator and point my repo at it"
```

Install and start

Option	Default	Purpose
`-p`, `--port`	`9297`	`0` lets the OS pick
`-b`, `--bind`	`127.0.0.1`	`0.0.0.0` only in a container
`--idle-timeout`	none	Stop after N idle seconds
`-a`, `--auth`	`on`	Require SigV4 signatures
`-t`, `--telemetry`	`on`	`off` opts this run out

On Linux, install the RPM or Debian package from `downloads.aws.com` and run `aws-emulator`. It stays in the foreground; `Ctrl+C` or `SIGTERM` stops it cleanly and deletes its files and all resources.

```
DL=https://downloads.aws.com/aws-emulator
U=$DL/packages/aws-emulator/LTS/latest
# RPM, x86_64 (or aws-emulator-aarch64.rpm)
sudo dnf install -y "$U/aws-emulator-x86_64.rpm"
```

```
# Debian, amd64 (or aws-emulator_arm64.deb)
curl -fsSLo emu.deb "$U/aws-emulator_amd64.deb"
sudo apt-get install -y ./emu.deb
aws-emulator -v  # version, release date, commit
aws-emulator     # serves https://127.0.0.1:9297
```

For Docker, build the image from the official Dockerfile (pass `--build-arg ARCH=aarch64` on Apple silicon). Then run it with the port in all three places, on `127.0.0.1` only, with the files directory mounted.

```
mkdir -p ~/.aws/emulator  # else owned by root
docker build -t aws-emulator .
P=9297
docker run --rm --memory=2g --memory-swap=2g \
  -p "127.0.0.1:$P:$P" -e HOME="$HOME" \
  -v "$HOME/.aws/emulator:$HOME/.aws/emulator" \
  aws-emulator -p "$P"
```

```
ARG REPO=public.ecr.aws/amazonlinux
FROM ${REPO}/amazonlinux:2023-minimal
ARG ARCH=x86_64
ARG DL=https://downloads.aws.com/aws-emulator
ARG U=${DL}/packages/aws-emulator/LTS/latest
ADD ${U}/aws-emulator-${ARCH}.rpm /tmp/emu.rpm
RUN rpm -i /tmp/emu.rpm && rm /tmp/emu.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"]
```

> **Warning:** `-p 9297:9297` publishes the emulator to your whole network. Always write `-p 127.0.0.1:9297:9297`. On Linux, also add `--user "$(id -u):$(id -g)"` so the files it writes belong to you.

Point the CLI and SDKs at it

File in `~/.aws/emulator/<port>/`	Holds
`emulator_credentials`	This run's access keys
`config`	Region, endpoint, CA bundle
`ca.pem`	This run's TLS certificate
`truststore.p12`	Same certificate, for Java

Each start writes new keys and a new certificate, readable only by you, which work with this run only. Requests must be signed with them (or with keys you create in the emulator's own IAM or STS). Put the variables in a file you `source` from the same script that runs your tests.

```
# emulator-env.sh — usage: source it [port]
P=${1:-9297}
D="$HOME/.aws/emulator/$P"
C="$D/emulator_credentials"
```

```
export AWS_ENDPOINT_URL="https://127.0.0.1:$P"
export AWS_SHARED_CREDENTIALS_FILE="$C"
export AWS_CONFIG_FILE="$D/config"
export AWS_CA_BUNDLE="$D/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
unset AWS_ACCESS_KEY_ID AWS_SECRET_ACCESS_KEY
unset AWS_SESSION_TOKEN AWS_PROFILE
```

Confirm where requests go before you trust a green test run:

```
aws sts get-caller-identity
# Account: 000000000000
# Arn: arn:aws:iam::000000000000:user/emulator
```

> **Gotcha:** environment credentials beat the credentials file. Leave `AWS_ACCESS_KEY_ID` set and the SDK signs with your real keys, so the emulator answers `AccessDenied` ("Sign the request with the emulator credentials"). Leave `AWS_ENDPOINT_URL` unset and your code calls real AWS instead.

Trust its certificate

Client	Setting
CLI v2, Boto3, Go v2	`AWS_CA_BUNDLE`
JavaScript v3 (Node.js)	`NODE_EXTRA_CA_CERTS`
Java 2.x	`JAVA_TOOL_OPTIONS` + `truststore.p12`
Rust, .NET	`SSL_CERT_FILE`

The emulator serves HTTPS only, with a certificate it creates on each start. Point the setting at the **path** of `ca.pem` and it keeps working across restarts; if you copy or import the certificate instead, redo it after every

start. On Windows, some SDKs need `ca.pem` imported into the certificate store.

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

> **Gotcha:** every setting except `NODE_EXTRA_CA_CERTS` *replaces* the default trust store, so the same process can no longer reach real AWS or any other HTTPS host (Java fails with `PKIX path building failed`). To reach both, build a bundle of your system certificates plus `ca.pem`. `JAVA_TOOL_OPTIONS` also reaches Gradle and Maven — scope it with `jvmArgs` or Surefire's `argLine`.

Host-name traps

Request	What breaks	Fix
`sync-`, `data-`, `stream-` ops	SDK prefixes host	Disable host prefix
S3 via `localhost`	Bucket moves to host	Use `127.0.0.1`
S3 Control	Account ID in host	Not reachable

The emulator serves every service on one host name, so any request your SDK sends to a different host never arrives. Step Functions `StartSyncExecution`, Cloud Map `DiscoverInstances` and CloudWatch Logs `StartLiveTail` get prefixes like `sync-127.0.0.1`. The env var covers the CLI and Boto3 (or `Config(inject_host_prefix=False)`); JavaScript and Java need a client option.

```
import { SFNClient } from "@aws-sdk/client-sfn";

// Endpoint and keys come from emulator-env.sh.
```

```
const sfn = new SFNClient({
  // the JS SDK ignores the env var
  disableHostPrefix: true,
});
// Java 2.x: set the advanced client option
// DISABLE_HOST_PREFIX_INJECTION to true in the
// client override configuration.
```

> **Warning:** set `AWS_DISABLE_HOST_PREFIX_INJECTION` only in the emulator shell. Against real AWS the prefixed operations then fail — Cloud Map `DiscoverInstances` returns `UnknownOperationException`.

Add it to an existing project

Existing setup	Change
Endpoint hard-coded in code	Remove it; let env vars apply

Existing setup	Change
`http://localhost:4566` style	`https://127.0.0.1:<port>`
Dummy `test` / `test` keys	Emulator keys, or `--auth off`
Retry or S3 settings in config	Copy emulator `config`, add them
Hand-written SDK mocks	Swap the ones that miss AWS errors

Start with the tests that are slow, flaky or costly against AWS and keep what already works. An explicit `endpoint` in client code wins over `AWS_ENDPOINT_URL`, so the cleanest integration is code with no endpoint at all. `AWS_CONFIG_FILE` swaps out your own config, and the emulator rewrites its copy on each start, so put extra settings in a copy and point the variable at that.

```
#!/usr/bin/env bash
# scripts/test-integration.sh
```

```
set -euo pipefail
aws-emulator --idle-timeout 900 > emu.log &
EMU=$!
trap 'kill "$EMU"' EXIT  # SIGTERM: clean stop
H=https://127.0.0.1:9297/__admin/health
CA=$HOME/.aws/emulator/9297/ca.pem
for _ in $(seq 30); do
  curl -sf --cacert "$CA" "$H" && break
  sleep 1
done
source ./emulator-env.sh 9297
npm run test:integration
```

> **Tip:** there is no reset route. For a clean slate between suites, restart the emulator — it starts in seconds — rather than writing teardown code.

CI and parallel runs

Need	Do
Several suites at once	One emulator per suite, own port
Free port, native	`-p 0`, read `endpoint_url`
Several Regions	One emulator per Region
Reproducible results	Pin a version from `versions.json`
Telemetry off	`-t off` or `EMULATOR_TELEMETRY=off`

Each emulator has its own port, keys, files directory and state, so parallel jobs never see each other's resources. With `-p 0`, read the port from the JSON startup line on stdout, matching it by its message rather than its position. Don't use `-p 0` in Docker: Docker can't forward a port chosen after start.

```
aws-emulator -p 0 -t off > emu.log &
M="emulator server information"
```

```
Q='fromjson? | .fields
  | select(.message == $m) | .endpoint_url'
until URL=$(jq -rRe --arg m "$M" "$Q" emu.log)
do sleep 0.2; done
P=${URL##*:}          # e.g. 41873
source ./emulator-env.sh "$P"
```

On a Linux runner, install a pinned package and start it in the background; later steps can use it.

```
- name: Start AWS Emulator
  run: |
    U=https://downloads.aws.com/aws-emulator
    U=$U/packages/aws-emulator/LTS
    curl -fsSLo e.deb \
      "$U/aws-emulator_${V}-1_amd64.deb"
    sudo apt-get install -y ./e.deb
    aws-emulator -t off --idle-timeout 1800 \
      > emu.log 2>&1 &
```

```
env:
  V: "1.0.337" # from LTS/versions.json
```

> **Gotcha:** in Docker, mapping a different host port to the same container port (`-p 127.0.0.1:9300:9297`) makes every container write to `~/.aws/emulator/9297/` , and they overwrite and delete each other's credentials. Change the port in all three places.

Embedded mode for Rust

Item	Value
Crate	`aws-emulator-embedded` , from GitHub
Dependency section	`[dev-dependencies]`
Version pin	`rev = "<release commit>"`
Auth default	Off (`Authentication::On` to sign)

Embedded mode runs a separate emulator inside each test, with no process, port, certificate or env vars: you hand its HTTP client to the SDK. Clients built from one emulator share its state; two emulators never do. Pin a commit, not a tag, because a commit can't move.

```
use aws_config::{BehaviorVersion, Region};
use aws_emulator_embedded::AwsEmulatorEmbedded;
#[tokio::test]
async fn creates_bucket() {
  let emu = AwsEmulatorEmbedded::default();
  let cfg = aws_config::defaults(
      BehaviorVersion::latest())
    .http_client(emu.http_client())
    .region(Region::new("us-east-1"))
    .credentials_provider(emu
      .aws_credentials_provider()).load().await;
  let s3 = aws_sdk_s3::Client::new(&cfg);
  s3.create_bucket().bucket("demo-bucket")
```

```
    .send().await.unwrap();
}
```

> **Gotcha:** the first `cargo test` compiles the emulator's service crates into your test binary — budget a long cold build and a large `target/`. Share one `CARGO_TARGET_DIR`, cache `target/` in CI, and keep the `rev` pin stable.

Fidelity, versions and limits

Area	Behavior
Unimplemented operation	HTTP 501 `NotImplemented`
Control plane (create, list)	Closest to AWS
Data plane, CloudFormation, CDK, Terraform	Varies by service
IAM, bucket and KMS key policies	Not enforced as on AWS

Area	Behavior
Regions and accounts	Not isolated in one emulator
Version scheme	`MAJOR.MINOR.PATCH`; fixes in latest only

Fidelity is measured with published tests that run a workflow on AWS and assert its result, state and exact errors. Each release lists a per-service score (tests passed out of total); compare scores only across the same test-suite version. Before you rely on a workflow, look up its operations in `manifest.jsonl` in the `aws-emulator-evaluation-suite` repository, and run uncovered operations against both the emulator and AWS.

A patch release can fix a response to match AWS and break a test that relied on the old one. Write assertions for what AWS does, pin a version if upgrades disturb CI, and report differences as GitHub issues with the

request, the response and AWS's response. Never put real data or real credentials in it, and run only code you trust: function code it runs is not isolated from your machine.

> **Warning:** a passing test proves nothing about security. The emulator never denies a request signed with its own key, and it ignores S3 bucket policies and KMS key policies. Test permissions, and the whole app, on real AWS before production.

Further reading

- AWS Emulator User Guide
- Pointing your AWS SDK at AWS Emulator
- Command reference
- Fidelity in AWS Emulator
- Releases and fidelity scores
- aws-emulator-evaluation-suite