AWS IoT Core

Verified 2026-11-02

Connect devices over MQTT with per-device identity, then route their data into AWS with rules — policies, topics, shadows, provisioning and limits.

Contents

Mental model 2

Things, certificates and policies 2

Endpoints and connecting 5

Topic design 7

Rules engine SQL and actions 8

Device shadows 9

Fleet provisioning 10

Defender, Greengrass and Basic Ingest 12

Limits and pricing 13

Further reading 14

Mental model

AWS IoT Core is a managed MQTT broker where every device has its own identity: an X.509 certificate with an IoT policy that says exactly which client ID it may use and which topics it may touch. Data leaves the broker through the **rules engine**, whose SQL statement selects messages by topic and hands them to actions such as Lambda, DynamoDB or S3. Get the identity model and the topic tree right first; everything else hangs off them.

Things, certificates and policies

Object	Role
Thing	Registry entry for one device
Certificate	Device identity, used for mutual TLS
IoT policy	What the certificate may do
Principal	Certificate attached to a thing

A device connects with a certificate; the IoT policy attached to that certificate is evaluated on every connect, publish and subscribe. Attach the certificate to

a thing so policy variables like `${iot:Connection.Thing.ThingName}` resolve. The sample is a policy's `Statement` array: replace `*:*` with your Region and account, and add the `iot:Connection.Thing.IsAttached` condition so only a certificate attached to the matching thing can connect. IoT policies are not IAM policies, though the JSON looks the same.

```
aws iot create-thing --thing-name pump-17
aws iot create-keys-and-certificate \
  --set-as-active \
  --certificate-pem-outfile cert.pem \
  --private-key-outfile key.pem
aws iot attach-thing-principal \
  --thing-name pump-17 --principal "$CERT_ARN"
aws iot attach-policy \
  --policy-name device --target "$CERT_ARN"
```

```
[
  {
    "Effect": "Allow",
    "Action": "iot:Connect",
    "Resource":
    "arn:aws:iot:*:*:client/${iot:ClientId}"
  },
  {
    "Effect": "Allow",
    "Action": "iot:Publish",
    "Resource":
    "arn:aws:iot:*:*:topic/dt/${iot:ClientId}/*"
  }
]
```

> **Gotcha:** `iot:Subscribe` checks a `topicfilter/` ARN, while `iot:Publish` and `iot:Receive` check `topic/` ARNs. Grant subscribe on a `topic/...` ARN and the device connects fine, but its SUBACK carries a failure code that many clients never check — so it just receives nothing.

Endpoints and connecting

Protocol	Auth	Port
MQTT	X.509 certificate	8883
MQTT	X.509, ALPN `x-amzn-mqtt-ca`	443
MQTT over WebSocket	SigV4 or custom authorizer	443
HTTPS (publish only)	X.509 or SigV4	8443 / 443

Every account has its own data endpoint per Region; get it from `describe-endpoint`. Use the device SDKs or any

MQTT client with the Amazon root CA. AWS IoT Core supports QoS 0 and 1, not QoS 2, and retained messages; MQTT 5 persistent sessions last up to one hour by default, adjustable to seven days.

```
aws iot describe-endpoint \
  --endpoint-type iot:Data-ATS
mosquitto_pub -h "$ENDPOINT" -p 8883 \
  --cafile AmazonRootCA1.pem \
  --cert cert.pem --key key.pem \
  -i pump-17 -q 1 \
  -t dt/pump-17/telemetry -m '{"rpm":1450}'
```

> **Gotcha:** The client ID must be unique. A second connection with the same ID disconnects the first, so two processes sharing a certificate and ID kick each other off in a loop.

Topic design

Rule	Value
Max forward slashes	7
Max topic size	256 bytes
Reserved prefix	`$aws/...` — AWS only
Scope	One account and Region

Put the device ID in the topic and keep the hierarchy shallow: `dt/<device>/<kind>` for telemetry, `cmd/<device>/<kind>` for commands, so a policy can pin each device to its own branch with `${iot:ClientId}`. Topics and filters are case-sensitive, and names must not contain personal data. The same name in another Region or account is a different topic.

```
dt/pump-17/telemetry    device → cloud
cmd/pump-17/restart     cloud → device
dt/+/telemetry          rule: every device
$aws/things/pump-17/shadow/update
```

Rules engine SQL and actions

Piece	Example
`FROM`	`'dt/+/telemetry'`
`topic(n)`	1-based topic segment
`WHERE`	`rpm > 3000`
Actions	Lambda, DynamoDB, S3, SQS, Kinesis, HTTP
`errorAction`	Where failed action payloads go

A rule subscribes to a topic filter, transforms each message with SQL, and runs every action in its list. Set `awsIotSqlVersion` explicitly — the documentation's examples use `2016-03-23`. Each action assumes an IAM role you give it.

```
SELECT rpm, topic(2) AS device,
       timestamp() AS ts
FROM 'dt/+/telemetry'
WHERE rpm > 3000
```

> **Gotcha:** A `PUBACK` means the rules engine received the message, not that the action succeeded. Add an `errorAction` (an S3 bucket or SQS queue) so payloads whose action failed are kept instead of lost.

Device shadows

Section	Written by
`desired`	Cloud or app
`reported`	Device
`delta`	Computed: desired ≠ reported
`version`	Optimistic locking

A shadow is a JSON document per thing — one classic and any number of named shadows — that stores state while the device is offline. The device subscribes to `.../shadow/update/delta`, applies the change, then reports it. Each shadow document is limited to 8 KB.

```
{
  "state": {
    "desired": { "mode": "eco" },
    "reported": { "mode": "boost" }
  },
  "version": 17
}
```

> **Tip:** Send `version` with updates from the cloud. A stale writer then gets a rejection instead of overwriting a newer state.

Fleet provisioning

Method	When
Provisioning by claim	Shared claim cert flashed at the factory
By trusted user	App fetches a 5-minute claim cert

Method	When
Pre-provisioning hook	Lambda approves each device

Instead of a certificate per device at manufacture, the device connects with a restricted claim certificate, calls `CreateKeysAndCertificate` or `CreateCertificateFromCsr` over MQTT, then `RegisterThing` with a provisioning template. Both are reserved topics with `accepted` and `rejected` replies: `$aws/certificates/create/json` and `$aws/provisioning-templates/<name>/provision/json`. The device must reconnect with its new certificate afterwards.

```
aws iot create-provisioning-template \
  --template-name pumps \
  --provisioning-role-arn "$ROLE_ARN" \
  --template-body file://template.json \
  --enabled
```

> **Warning:** A claim certificate is shared by every device in a batch. Scope its policy to the provisioning topics only, and use a pre-provisioning hook to reject serial numbers you did not build.

Defender, Greengrass and Basic Ingest

Feature	Use
Device Defender	Audit configs, detect anomalous behavior
Greengrass V2	Run components at the edge
Basic Ingest	`$aws/rules/<rule>/...`, no messaging charge

Device Defender audits your IoT configuration and flags devices whose behavior deviates from what you expect. Greengrass runs local processing on a gateway. Basic Ingest publishes straight to a rule, skipping the broker and its messaging cost — but nothing can subscribe to those messages.

```
mosquitto_pub -h "$ENDPOINT" -p 8883 \
  --cafile AmazonRootCA1.pem \
  --cert cert.pem --key key.pem -i pump-17 \
  -t '$aws/rules/pumpTelemetry/dt/pump-17' \
  -m '{"rpm":1450}'
```

Limits and pricing

Quota	Default
Message size	128 KB
Publishes per second per connection	100, not adjustable
Throughput per connection	512 KB/s
Subscriptions per connection	50, adjustable
Connects per second per client ID	1

Publishes over the per-connection rate are discarded, so batch readings into one message rather than sending one per sensor. Pricing is metered separately for connectivity, messaging, shadow operations and rules

triggered plus actions executed. Many account-level quotas are lower in some Regions — check the quotas page for yours.

```
aws service-quotas list-service-quotas \
  --service-code iotcore \
  --query 'Quotas[].[QuotaName,Value]' \
  --output table
```

Further reading

- AWS IoT Core Developer Guide
- AWS IoT Core endpoints and quotas
- Designing MQTT topics for AWS IoT Core
- AWS IoT Core pricing