QUICKSTART AND VERIFIER GUIDE · QODIQA-QS-1.0

From zero to a *verified* decision.

Run the sandbox, make a first governed request, then check the evidence offline with a verifier that trusts nothing the gateway tells it.

VERIFIED ON 2026-10-29 · WINDOWS 11 · GATEWAY 1.0.0 · VERIFY 1.0.0

8
STEPS, CLONE TO PROOF

4
OFFLINE VERIFIER TOOLS

1
TAMPERED FIELD, CAUGHT

QODIQA-SANDBOX · BASH

```
$ docker compose -f docker-compose.quickstart.yml -f ops.yml up -d
$ until curl -sf http://127.0.0.1:3000/health; do sleep 2; done
{"status":"ok","uptime":1.444858324}

$ curl -s -X POST localhost:3000/sign -H "Authorization: Bearer $SK" \
  -H 'content-type: application/json' -d @sign-request.json -o sign.json
$ curl -s -X POST localhost:3000/ai/chat -H 'content-type: application/json' \
  -H "x-qodiqa-artifact: $ARTIFACT" -d '{"prompt":"hello"}' -o chat.json
{ "decision": "EXECUTE", "reason": "ALL_CHECKS_PASSED",
  "response": "[sandbox] hello",

$ node $V/verify.external.js --offline --historical \
  --key-file qodiqa-keys.json --bundle session.json
{"valid":true,"reason":"PASS","mode":"HISTORICAL",...}

$ node $V/qodiqa-receipt-verify.js --receipt chat.json \
  --keys qodiqa-transparency-keys.json
{"valid":true,"signature":{"ok":true,"reason":null},
 "binding":{"ok":true,"mismatches":[]}}
```

ED25519 SIGNATURE · VERIFIED OFFLINE EXIT 0

QODIQA-QS-1.0 | V1.0 | 2026-10-29

FRONT MATTER · ABOUT THIS DOCUMENT

A guide you can *run*, then check.

It takes you from a clean machine to a governed request, and then to evidence that a third party can verify offline. Every command and every output below was run for real, against the public repositories, on the machine named on this page.

HOW TO READ IT

Each step shows the command and the output it produced. Outputs are trimmed where they are long and say so; nothing is invented, and the secret signing key is never printed. Where Windows differs from macOS and Linux there is one clear path for each.

FOUR STATUSES, NEVER MIXED

STATUS	MEANING
IMPLEMENTED	In the code and covered by tests or a documented run.
SPECIFIED	Defined in the standard; not a claim about the gateway.
PLANNED	Listed on the gateway roadmap. Not supported today.
OUT OF SCOPE	Stated as not provided in the scope and limitations page.

> **BOUNDARIES OF THIS GUIDE**
>
> The sandbox is for evaluation, not for deployment. This guide is technical documentation: not a certification, an audit opinion or legal advice. QODIQA supports evidence for regulatory frameworks; it does not make anyone compliant. No endorsement or partnership with any named organisation, product or standard is stated or implied.

VERIFIED ON

ITEM	VALUE
Date	2026-10-29
OS and shells	Windows 11 Pro 10.0.26200; Git Bash and Windows PowerShell 5.1
Containers	Docker 29.8.0, Compose v5.5.1
Runtimes	Node.js v24.19.0, Python 3.14.7
Gateway	qodiqa-gateway 1.0.0 at `d4bdebb`
Verifiers	qodiqa-verify 1.0.0 at `fe77427`

DOCUMENT CONTROL

ITEM	VALUE
Document ID	QODIQA-QS-1.0
Version and date	1.0 · 2026-10-29
Licence	CC BY 4.0
Contact	www.qodiqa.com · contact@qodiqa.com

LICENCES OF THE MATERIAL DESCRIBED

MATERIAL	LICENCE
qodiqa-gateway, qodiqa-verify	Apache License 2.0
Specification corpus (32 documents)	CC BY 4.0

CONTENTS

What is *inside.*

01 **Get running** 04

1.1 Prerequisites and Step 1 04

02 **First governed request** 05

2.1 Sign and send (steps 2, 3) 05

2.2 Two ways to be refused 06

2.3 Windows PowerShell 07

2.4 Request lifecycle 08

03 **Verify it offline** 09

3.1 Step 6, collect the evidence 09

3.2 Run the verifiers 10

3.3 Tamper and read the verdict 11

04 **Policy and SDKs** 12

4.1 Policy basics 12

4.2 Three policies to copy 13

4.3 SDKs 14

05 **Run it for real** 15

5.1 Troubleshooting 15

5.2 Production checklist 16

5.3 Next steps and known issues 17

SHORT ON TIME

Pages 4 to 6

Start the sandbox, get one EXECUTE and one BLOCK.

ON WINDOWS

Page 7

The same path in PowerShell 5.1 with `curl.exe`.

ONLY HERE TO VERIFY

Pages 9 to 11

Collect evidence, run the four verifiers, tamper with one field.

1.1 PREREQUISITES AND STEP 1

Up in five *minutes.*

The sandbox brings its own stand-in model, Redis and PostgreSQL: no provider account, no API key, no `.env` file.

WHAT YOU NEED

TOOL	TESTED WITH
Docker with Compose v2	Docker 29.8.0, Compose v5.5.1
Node.js 22 or later	v24.19.0
git and curl	git 2.55.0, curl 8.21.0
Free port	3000, loopback only

Shells. macOS, Linux and Git Bash use this page; Windows PowerShell 5.1 uses page 7. `jq` is not needed.

STEP 1 · CLONE, START, COLLECT YOUR TWO KEYS

MACOS · LINUX · GIT BASH

```
mkdir qodiqa-quickstart && cd qodiqa-quickstart
git clone https://github.com/qodiqa/qodiqa-gateway.git
git clone https://github.com/qodiqa/qodiqa-verify.git
cd qodiqa-gateway
```

ONE EXTRA FILE

The stock Compose file gives no operator key, which the evidence endpoints need. `ops.yml` keeps one across a restart, signs a checkpoint every 2 entries (default 1,000) and mounts the policy file for page 12. This guide's addition, not the repository's.

OPS.YML · CREATE IT IN QODIQA-GATEWAY

```
services:
  gateway:
    environment:
      QODIQA_RBAC_STORE_PATH: /app/data/rbac-users.json
      QODIQA_AUDIT_CHECKPOINT_EVERY: "2"
    volumes:
      - ./scripts/dev/policies.sandbox.json:/app/scripts/dev/policies.sandbox.json
```

START THE SANDBOX, WAIT FOR HEALTH

```
dc() { docker compose -f docker-compose.quickstart.yml -f ops.yml "$@"; }
dc up -d
until curl -sf http://127.0.0.1:3000/health; do sleep 2; done
```

EXPECTED OUTPUT

```
{"status":"ok","uptime":1.444858324}
```

OPERATOR KEY, RESTART, SIGNING KEY

```
export OK=$(dc exec -T gateway node scripts/rbac-bootstrap-key.js 2>/dev/null \
  | sed -n 's/.*"apiKey": "\(ok_[0-9a-f]*\)".*/\1/p')
dc restart gateway
until curl -sf http://127.0.0.1:3000/health >/dev/null; do sleep 2; done
export SK=$(dc logs --no-log-prefix gateway | sed -n 's/^Authorization: Bearer //p' | tail -1)
```

2.1 STEPS 2 AND 3 · SIGN, THEN EXECUTE

Ask first, *then* send.

Run both steps within a minute: an artifact lives at most 60 seconds. The body you send must be exactly the body you asked the gateway to sign.

STEP 2 · SIGN THE REQUEST YOU INTEND TO MAKE

MACOS · LINUX · GIT BASH

```
cd .. && mkdir work && cd work
cat > sign-request.json <<'EOF'
{ "request": { "method": "POST", "path": "/ai/chat",
               "body": { "prompt": "hello" } },
  "payload": { "userId": "me" }, "ttl": 60000 }
EOF
curl -s -X POST localhost:3000/sign -H "Authorization: Bearer $SK" \
  -H 'content-type: application/json' -d @sign-request.json -o sign.json
```

SIGN.JSON · EXCERPT

```
{ "artifactId": "525f88aa-3082-43a3-8ff9-35d6fd1c298f",
  "artifact": { "expiry": 1793276404404,
    "payload": { "userId": "me", "provider": "local", "tenantId": "dev-local",
      "requestHash": "2287207e...ace08b", "policyHash": "43526542...2c0f01",
      "callerContext": { "callerName": "dev-local-dev", ... } },
    "publicKeyId": "dev-local:key_v1",
    "signature": "LAodufRk...hHAA==", "timestamp": 1793276344404 },
  "meta": { "effectiveTtlMs": 60000, "ttlClamped": false, ... } }
```

STEP 3 · SEND IT, WITH THE ARTIFACT IN A HEADER

MACOS · LINUX · GIT BASH

```
ARTIFACT=$(node -e "process.stdout.write(Buffer.from(JSON.stringify(
  require('./sign.json').artifact)).toString('base64'))")
curl -s -X POST localhost:3000/ai/chat -H 'content-type: application/json' \
  -H "x-qodiqa-artifact: $ARTIFACT" -d '{"prompt":"hello"}' -o chat.json
node -e "console.log(JSON.stringify(require('./chat.json'),null,1))"
```

CHAT.JSON · EXCERPT — EXIT EXECUTE

```
{ "decision": "EXECUTE", "reason": "ALL_CHECKS_PASSED",
  "response": "[sandbox] hello",
  "artifactId": "525f88aa-3082-43a3-8ff9-35d6fd1c298f",
  "transparency": { "transparencyId": "bd0ee363-4fe6-434d-820e-b30fb115206c",
    "persisted": true, "signerKeyId": "tr-dev-local:key_v1", ... },
  "aiDisclosure": { "aiGenerated": true, "model": "sandbox-echo-1", ... },
  "receipt": { "v": 1, "decision": "ALLOW", "keyId": "tr-dev-local:key_v1",
    "entryHash": "8de27eeb...", "leafHash": "b04fd26e...",
    "signature": "AJA/iFhM...gCQ==", ... } }
```

WHAT JUST HAPPENED

The gateway verified the signature, recomputed the request hash from method, path and body, consumed the one-time nonce, checked consent and the sandbox rules, called the stand-in model, and appended the decision to the audit chain and the transparency log. The signed `receipt` proves the gateway recorded it.

2.2 STEPS 4 AND 5 · TWO WAYS TO BE REFUSED

A *refusal* is a decision too.

Every BLOCK is an HTTP 200 with a reason, never a silent failure. A reused artifact and a rule hit look different.

STEP 4 · REPLAY THE SAME ARTIFACT

MACOS · LINUX · GIT BASH

```
curl -s -X POST localhost:3000/ai/chat -H 'content-type: application/json' \
  -H "x-qodiqa-artifact: $ARTIFACT" -d '{"prompt":"hello"}'
```

EXPECTED OUTPUT — EXIT BLOCK

```
{ "decision": "BLOCK", "reason": "REPLAY_DETECTED",
  "details": { "requestId": "525f88aa-3082-43a3-8ff9-35d6fd1c298f" },
  "requestId": "976d1ea9-1d54-4b43-9777-0caf9f3593e2" }
```

The artifact carries a one-time code (a nonce) recorded atomically in Redis when step 3 used it. A second use is refused; sign again.

STEP 5 · TRIP A RULE, WITH GOVERN.MJS (RIGHT)

MACOS · LINUX · GIT BASH · POWERSHELL

```
node govern.mjs "sandbox-block-me please"
node govern.mjs "hello again"
```

EXPECTED OUTPUT

```
BLOCK CONTENT_RULE_BLOCKED  [no receipt]
EXECUTE ALL_CHECKS_PASSED [sandbox] hello again [receipt ALLOW]
```

The sandbox's demo rule `SANDBOX-DEMO-CONTENT-RULE` refuses the phrase `sandbox-block-me`.

GOVERN.MJS · SIGN AND SEND IN ONE GO (NODE 22+, ANY OS)

CREATE GOVERN.MJS IN THE WORK FOLDER

```
// govern.mjs: sign one request, send it with the artifact, keep the answer
import { writeFile } from 'node:fs/promises';
const [prompt, userId = 'me'] = process.argv.slice(2);
const base = 'http://localhost:3000', body = { prompt };
const post = (path, headers, payload) => fetch(base + path, {
  method: 'POST', headers: { 'content-type': 'application/json', ...headers },
  body: JSON.stringify(payload) }).then((r) => r.json());

const signed = await post('/sign', { authorization: `Bearer ${process.env.SK}` }, {
  request: { method: 'POST', path: '/ai/chat', body },
  payload: { userId }, ttl: 60000 });
const artifact = Buffer.from(JSON.stringify(signed.artifact)).toString('base64');
const out = await post('/ai/chat', { 'x-qodiqa-artifact': artifact }, body);
await writeFile('last.json', JSON.stringify(out));
console.log(out.decision, out.reason, out.response ?? '',
  out.receipt ? `[receipt ${out.receipt.decision}]` : '[no receipt]');
```

It does steps 2 and 3 for any prompt: `/sign`, then `/ai/chat` with the artifact. In PowerShell create it with a here-string (`@' … '@ | Set-Content govern.mjs -Encoding ascii`).

2.3 WINDOWS POWERSHELL 5.1 · STEPS 1 TO 3

The same path, in *PowerShell*.

Use `curl.exe`, not the `curl` alias, and keep JSON in files: 5.1 strips the quotes from an inline `-d '{...}'`. Steps 4 and 5 are page 6 unchanged.

STEP 1 · CLONE, CREATE OPS.YML, START

WINDOWS POWERSHELL 5.1 (NO && OPERATOR IN 5.1)

```
mkdir qodiqa-quickstart; cd qodiqa-quickstart
git clone https://github.com/qodiqa/qodiqa-gateway.git
git clone https://github.com/qodiqa/qodiqa-verify.git
cd qodiqa-gateway
@'
services:
  gateway:
    environment:
      QODIQA_RBAC_STORE_PATH: /app/data/rbac-users.json
      QODIQA_AUDIT_CHECKPOINT_EVERY: "2"
    volumes:
      - ./scripts/dev/policies.sandbox.json:/app/scripts/dev/policies.sandbox.json
'@ | Set-Content ops.yml -Encoding ascii
function dc { docker compose -f docker-compose.quickstart.yml -f ops.yml @args }
dc up -d
do { Start-Sleep 2; $h = curl.exe -sf http://127.0.0.1:3000/health } until ($h); $h
```

OPERATOR KEY, RESTART, SIGNING KEY

```
$env:OK = (dc exec -T gateway node scripts/rbac-bootstrap-key.js 2>$null |
  Select-String '"apiKey"') -replace '.*"(ok_[0-9a-f]+)".*', '$1'
dc restart gateway
do { Start-Sleep 2; $h = curl.exe -sf http://127.0.0.1:3000/health } until ($h)
$env:SK = ((dc logs --no-log-prefix gateway) -match '^Auth')[-1].Split(' ')[-1]
```

STEPS 2 AND 3 · SIGN, THEN SEND

WINDOWS POWERSHELL 5.1

```
cd ..; mkdir work; cd work
@'
{ "request": { "method": "POST", "path": "/ai/chat",
               "body": { "prompt": "hello" } },
  "payload": { "userId": "me" }, "ttl": 60000 }
'@ | Set-Content sign-request.json -Encoding ascii
'{"prompt":"hello"}' | Set-Content body.json -Encoding ascii -NoNewline
curl.exe -s -X POST localhost:3000/sign -H "Authorization: Bearer $env:SK" `
  -H "content-type: application/json" -d "@sign-request.json" -o sign.json
$env:ARTIFACT = node -e "process.stdout.write(Buffer.from(JSON.stringify(
  require('./sign.json').artifact)).toString('base64'))"
curl.exe -s -X POST localhost:3000/ai/chat -H "content-type: application/json" `
  -H "x-qodiqa-artifact: $env:ARTIFACT" --data-binary "@body.json" -o chat.json
node -e "const c=require('./chat.json');
console.log(c.decision, c.reason, c.response)"
```

EXPECTED OUTPUT

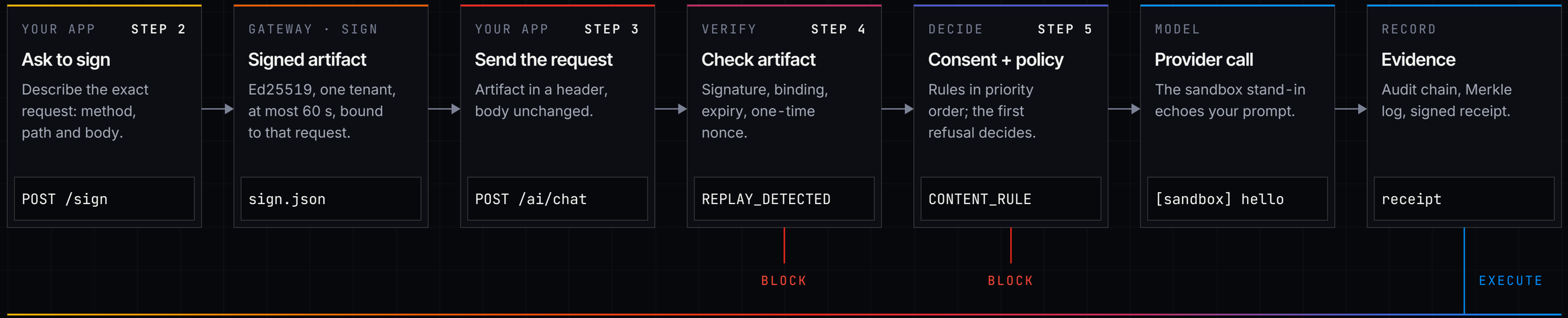
```
EXECUTE ALL_CHECKS_PASSED [sandbox] hello
```

Replay (step 4): repeat the second `curl.exe` line, without `-o`. Step 5: create `govern.mjs` with a here-string as above, then `node govern.mjs "sandbox-block-me please"`. `$env:SK` is visible to node as `process.env.SK`.

2.4 THE WHOLE PATH AT A GLANCE

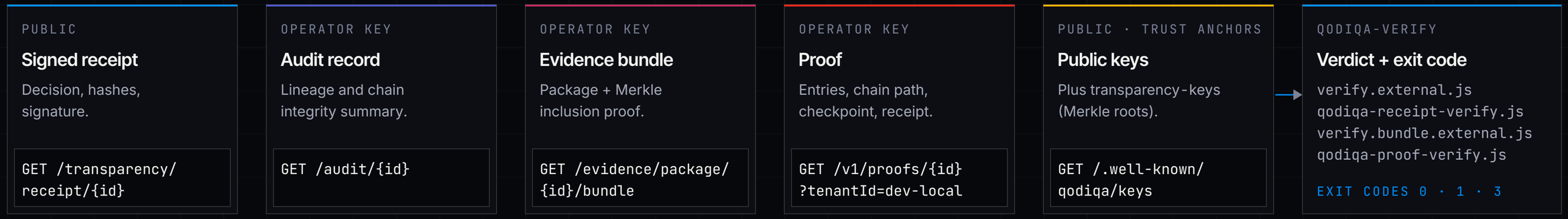
Where each *command* sits.

One request, from signing to offline verification. The step numbers are the steps of the walkthrough on pages 5 to 11.



STEP 6 · FETCH THE EVIDENCE

STEP 7 · CHECK IT OFFLINE



3.1 STEP 6 · COLLECT THE EVIDENCE

Evidence you do *not* have to trust.

The verifiers in `qodiqa-verify` import nothing from the gateway and make no network call: evidence plus public keys in, verdict out.

WHAT EACH CHECK PROVES

CHECK	FAILS WHEN
Signature (Ed25519)	the artifact, receipt or seal was altered
Request binding	method, path or body differ from what was signed
Hash chain	an audit entry no longer hashes to its own `hash`
Merkle inclusion	the entry is not under the signed root
Checkpoint	the signed head does not match the chain path

> **TRUST THE KEYS, NOT THE BUNDLE**
>
> Keys fetched from the gateway you check only prove self-consistency; in an audit take them over a channel you trust. Valid does not mean the operator behaved well or the model was right.

READ THE AUDIT RECORD

AUDIT.JSON: LINEAGE AND CHAIN INTEGRITY

```
node -p "const d=require('./audit.json').data;
[d.lineageState, d.chainIntegrity.valid].join(' ')"
# prints: COMPLETE true
```

STEP 6 · FETCH KEYS, RECEIPT, AUDIT RECORD, BUNDLE AND PROOF

MACOS · LINUX · GIT BASH · IN THE WORK FOLDER

```
ID=$(node -p "require('./sign.json').artifactId")   # = payload.requestId
for k in keys transparency-keys; do
  curl -s localhost:3000/.well-known/qodiqa/$k -o qodiqa-$k.json; done
curl -s localhost:3000/audit/$ID -H "Authorization: Bearer $OK" -o audit.json
sleep 70   # a signed Merkle root is written every 60 seconds
curl -s localhost:3000/evidence/package/$ID/bundle \
  -H "Authorization: Bearer $OK" -o bundle.json
curl -s "localhost:3000/v1/proofs/$ID?tenantId=dev-local&fullChain=true" \
  -H "Authorization: Bearer $OK" -o proof.json
```

WINDOWS POWERSHELL 5.1

```
$id = node -p "require('./sign.json').artifactId"   # = payload.requestId
foreach ($k in 'keys','transparency-keys') {
  curl.exe -s localhost:3000/.well-known/qodiqa/$k -o qodiqa-$k.json }
curl.exe -s localhost:3000/audit/$id -H "Authorization: Bearer $env:OK" -o audit.json
Start-Sleep 70   # a signed Merkle root is written every 60 seconds
curl.exe -s localhost:3000/evidence/package/$id/bundle `
  -H "Authorization: Bearer $env:OK" -o bundle.json
curl.exe -s "localhost:3000/v1/proofs/${id}?tenantId=dev-local&fullChain=true" `
  -H "Authorization: Bearer $env:OK" -o proof.json
```

Why the wait. The bundle holds a Merkle inclusion proof only once a signed root covers the entry; before that `transparency.available` is `false`. **Why `fullChain`.** Without it the gateway withholds the chain path whenever it holds other requests' entries, and the verdict adds `CHECKPOINT_ABSENT`; it needs the admin key.

3.2 STEP 7 · RUN THE VERIFIERS

Four tools, *one* verdict each.

No server, no database and no gateway code is involved. Each tool prints a verdict and sets an exit code you can script against.

STEP 7 · CHECK THE ARTIFACT, THE RECEIPT, THE BUNDLE AND THE PROOF

MACOS · LINUX · GIT BASH · IN THE WORK FOLDER

```
V=../qodiqa-verify/bin
node -e "const fs=require('fs'),a=require('./sign.json').artifact;
fs.writeFileSync('artifact.json',JSON.stringify(a));
fs.writeFileSync('session.json',JSON.stringify({artifact:a,
  request:{method:'POST',path:'/ai/chat',body:{prompt:'hello'}}}))"
node $V/verify.external.js --offline --historical \
  --key-file qodiqa-keys.json --bundle session.json
node $V/qodiqa-receipt-verify.js --receipt chat.json \
  --keys qodiqa-transparency-keys.json
node $V/verify.bundle.external.js --keys qodiqa-keys.json \
  --transparency-keys qodiqa-transparency-keys.json bundle.json
node $V/qodiqa-proof-verify.js --proof proof.json --artifact artifact.json
```

PowerShell: `$v = '..\qodiqa-verify\bin'`, then `node $v\verify.external.js ...`, same arguments, backtick for line breaks.

SECOND IMPLEMENTATION, RECORDED PROOF

PYTHON, STANDARD LIBRARY ONLY

```
python ../qodiqa-verify/python/qodiqa_proof_verify.py \
  --proof ../qodiqa-verify/examples/proof/proof.example.json
```

VERIFY.EXTERNAL.JS · THE ARTIFACT — EXIT 0


```
{"valid":true,"reason":"PASS","mode":"HISTORICAL","callerContext":{...},...}
```


QODIQA-RECEIPT-VERIFY.JS · THE RECEIPT — EXIT 0


```
{"valid":true,"signature":{"ok":true,"reason":null},
 "binding":{"ok":true,"mismatches":[]}}
```


VERIFY.BUNDLE.EXTERNAL.JS · THE EVIDENCE BUNDLE — EXIT 0


```
{ "bundleHashValid": true, "bundleSignatureValid": true,
  "packageValid": true, "transparencyProofValid": true,
  "trust": { "anchor": "keys", "keyId": "dev-local:key_v1", ... },
  "valid": true, "reasons": [] }
```


QODIQA-PROOF-VERIFY.JS · THE PROOF — EXIT 3


```
{"verdict":"VALID_PARTIAL","failed":[],"partial":["ANCHORS_ABSENT"]}
```


WHY EXIT 3 IS THE GOOD NEWS HERE

Every check that could run passed. Missing: an outside witness (ANCHORS_ABSENT); VALID needs --anchors-repo.

PYTHON · RECORDED PROOF — EXIT 3


```
{"verdict": "VALID_PARTIAL", "failed": [], "partial": [...]}
```

3.3 STEP 8 · TAMPER WITH ONE FIELD

Change one field. *Watch* it fail.

One edit, ALLOW to BLOCK, breaks every seal that covers it, and the verifier exits non-zero.

BEFORE, AFTER, AND THE ONE-FIELD DIFF

MACOS · LINUX · GIT BASH

```
sed 's/"decision":"ALLOW"/"decision":"BLOCK"/' bundle.json \
  > bundle-tampered.json
diff <(tr ',' '\n' < bundle.json) \
     <(tr ',' '\n' < bundle-tampered.json)
```

DIFF · ONE LINE DIFFERS (ITS NUMBER VARIES)

```
13c13
< "decision":"ALLOW"
---
> "decision":"BLOCK"
```

WINDOWS POWERSHELL 5.1

```
$s = [IO.File]::ReadAllText("$PWD\bundle.json")
$re = [regex]::new('"decision":"ALLOW"')
[IO.File]::WriteAllText("$PWD\bundle-tampered.json",
  $re.Replace($s, '"decision":"BLOCK"', 1))
```

VERIFY THE TAMPERED COPY

```
node $V/verify.bundle.external.js --keys qodiqa-keys.json \
  --transparency-keys qodiqa-transparency-keys.json \
  bundle-tampered.json
```

BEFORE · BUNDLE.JSON — EXIT 0


```
{ "bundleHashValid": true, "bundleSignatureValid": true,
  "packageValid": true, "transparencyProofValid": true,
  "valid": true, "reasons": [] }
```


AFTER · BUNDLE-TAMPERED.JSON — EXIT 1


```
{ "bundleHashValid": false, "bundleSignatureValid": false,
  "packageValid": false, "transparencyProofValid": true,
  "valid": false,
  "reasons": [
    "BUNDLE_HASH_MISMATCH: expected 017642e9...120c, computed f7b0b877...3ec2",
    "BUNDLE_SIGNATURE_INVALID", "EVIDENCE_PACKAGE_INVALID",
    "  package: PACKAGE_HASH_MISMATCH: expected 08c76a25...010d, computed 72955938...fe6b",
    "  package: SIGNATURE_INVALID" ] }
```


EDIT	VERDICT	EXIT
Artifact: send goodbye instead of hello	REQUEST_MISMATCH	1
Receipt: ALLOW to BLOCK	BAD_SIGNATURE	1
Proof: first ALLOW to BLOCK	INVALID, ENTRY_HASH:0	1

READING THE EXIT CODES

0 valid · 1 invalid · 2 usage · 3 valid but partial. Use the Node verifier for live proofs (Python: page 17).

4.1 POLICY BASICS

Rules *decide,* in order.

Rules run in `priority` order, the first refusal decides, and the set's hash is bound into every artifact you sign.

A RULE, AS THE SANDBOX SHIPS IT

SCRIPTS/DEV/POLICIES.SANDBOX.JSON · ONE OF THREE RULES

```
{ "id": "SANDBOX-DEMO-CONTENT-RULE", "type": "CONTENT_RULE",
  "priority": 20, "enabled": true,
  "config": { "contentFormat": 1, "direction": "input",
    "action": "BLOCK", "threshold": 1,
    "items": [ { "id": "demo-phrase", "kind": "phrase",
                 "value": "sandbox-block-me" } ] } }
```

FIELD	MEANING
type	REQUIRE_CONSENT, BLOCK_IP, TIME_WINDOW, RATE_LIMIT, CONTENT_RULE, DLP, DATA_RESIDENCY, ABAC, BUDGET, ROUTING
priority	lower runs first; the first refusal decides
config	type-specific; validated at load, so a typo stops start-up

SANDBOX ONLY

A file is how the sandbox changes rules. In a deployment the enforced set lives in the PostgreSQL control plane and changes pass draft, review, approval and publication by different operators.

CHANGE THE RULES, RESTART, CHECK THEY LOADED

MACOS · LINUX · GIT BASH · BACK IN QODIQA-GATEWAY

```
cd ../qodiqa-gateway
# edit scripts/dev/policies.sandbox.json: add rules to "policies"
dc restart gateway
until curl -sf http://127.0.0.1:3000/health >/dev/null; do sleep 2; done
dc logs --no-log-prefix gateway | grep POLICIES_LOADED | tail -1
export SK=$(dc logs --no-log-prefix gateway \
  | sed -n 's/^Authorization: Bearer //p' | tail -1)
cd ../work
```

POLICIES_LOADED · AFTER PAGE 13'S THREE RULES


```
{"eventType":"POLICIES_LOADED","count":6,"ids":["SANDBOX-REQUIRE-CONSENT",
 "SANDBOX-BLOCK-IP","SANDBOX-DEMO-CONTENT-RULE","ACME-CODENAMES",
 "ABAC_SHORT_PROMPTS","DLP-NO-CARDS"],"role":"import-source"}
```


BEFORE YOU WRITE RULES

- **A restart gives new signing and public keys** and clears the sandbox's in-memory audit: collect evidence (page 9) first. PowerShell: `dc restart gateway`, then the `$env:SK` lines of page 7.
- **A missing attribute makes a comparison false.** A DENY rule does not fire on what it cannot see; a REQUIRE rule refuses on it. Write allow-lists as REQUIRE.
- **`purpose` and `dataRegion` are declared by the caller:** policy routing, not a security boundary.

4.2 THREE POLICIES TO COPY

Paste, restart, *prove* it.

Add the three objects to the `policies` array, restart as on page 12, and send each prompt with `node govern.mjs "..."`.

1 · BLOCK PROJECT CODENAMES AND TICKET NUMBERS (CONTENT RULE)

```
{ "id": "ACME-CODENAMES", "type": "CONTENT_RULE", "priority": 21,
  "enabled": true,
  "config": { "contentFormat": 1, "direction": "input",
    "action": "BLOCK", "threshold": 1, "items": [
      { "id": "codename", "kind": "phrase", "value": "project nightingale" },
      { "id": "ticket", "kind": "regex", "value": "nx-[0-9]{4}" } ] } }
```

2 · REFUSE CARD NUMBERS AND CREDENTIALS (DATA LOSS PREVENTION)

```
{ "id": "DLP-NO-CARDS", "type": "DLP", "priority": 50, "enabled": true,
  "config": { "dlpFormat": 1, "direction": "input", "action": "BLOCK",
    "rules": ["FINANCIAL", "CREDENTIAL"], "minConfidence": 0.6 } }
```

3 · REFUSE LONG PROMPTS (ATTRIBUTE RULE)

```
{ "id": "ABAC_SHORT_PROMPTS", "type": "ABAC", "priority": 31,
  "enabled": true,
  "config": { "abacFormat": 1, "effect": "DENY",
    "reason": "PROMPT_TOO_LONG",
    "when": { "attr": "request.promptChars", "op": "gt", "value": 200 } } }
```

Mind the commas and keep the file valid JSON: an invalid rule stops the gateway at start-up.

WHAT EACH ONE DID, AS TESTED

RULE	PROMPT	RESULT
1	Summarize project nightingale for me	BLOCK CONTENT_RULE_BLOCKED
1	Please fix NX-1234 today	BLOCK CONTENT_RULE_BLOCKED
1	Summarize the quarterly report	EXECUTE
2	My card number is 4111 1111 1111 1111	BLOCK DLP_BLOCKED, rule CARD
2	What is two plus two	EXECUTE
3	300 characters (`word` times 60)	BLOCK ABAC_PROMPT_TOO_LONG, with receipt
3	Short prompt	EXECUTE

KEYS CHANGE AT EVERY RESTART: FETCH THEM AGAIN

```
curl -s localhost:3000/.well-known/qodiqa/transparency-keys \
  -o qodiqa-transparency-keys.json
node $V/qodiqa-receipt-verify.js --receipt last.json \
  --keys qodiqa-transparency-keys.json --decision BLOCK
```

EXPECTED OUTPUT — EXIT 0

```
{"valid":true,"signature":{"ok":true,"reason":null},
 "binding":{"ok":true,"mismatches":[]}}
```

4.3 SDKS

The same flow, in *code*.

Sign, send, read the decision. None of the SDKs is published to a package registry: install from the repository.

JAVASCRIPT · GOVERNED.MJS · RAN: EXECUTE ALL_CHECKS_PASSED

```javascript
import { QodiqaClient } from '@qodiqa/sdk';
const client = new QodiqaClient({ baseUrl: 'http://localhost:3000',
                                  signToken: process.env.SK });
const body = { prompt: 'hello' };
const signed = await client.sign.sign({
  request: { method: 'POST', path: '/ai/chat', body },
  payload: { userId: 'me' }, ttl: 60_000 });
const artifact = Buffer.from(JSON.stringify(signed.artifact)).toString('base64');
const chat = await client.chat.create(artifact, body);
console.log(chat.decision, chat.reason);
```

INSTALL FROM THE REPOSITORY, THEN RUN

```
(cd ../qodiqa-gateway && npm run sdk:build)
npm init -y && npm install ../qodiqa-gateway/sdk/javascript
node governed.mjs
```

TypeScript: the same code in an `async` function, compiled with `tsc --noEmit --strict` against the SDK's declarations without errors.

Go and Java have no sign/chat helpers: they call operations by name (`signRequest`) and implement protocol V1 only. See `sdk/go/README.md` and `sdk/java/README.md`.

PYTHON · GOVERNED.PY · RAN: EXECUTE ALL_CHECKS_PASSED

```python
import base64, json, os
from qodiqa import QodiqaClient

client = QodiqaClient(base_url='http://localhost:3000',
                      sign_token=os.environ['SK'])
body = {'prompt': 'hello'}
signed = client.sign.sign({
    'request': {'method': 'POST', 'path': '/ai/chat', 'body': body},
    'payload': {'userId': 'me'}, 'ttl': 60000})
artifact = base64.b64encode(json.dumps(signed['artifact']).encode()).decode()
chat = client.chat.create(artifact, body)
print(chat['decision'], chat['reason'])
```

INSTALL FROM THE REPOSITORY, THEN RUN

```
python -m venv venv    # macOS/Linux: venv/bin/...  Windows: venv/Scripts/...
venv/Scripts/pip install ../qodiqa-gateway/sdk/python
venv/Scripts/python governed.py
```

GO AND JAVA	EXERCISED WITH	RESULT
Go 1.21+	`sdk/go/cmd/smoke`, `golang:1.22` container	8 of 8 pass
Java 17+	Smoke in `sdk/java`, `maven:3.9` container	8 of 8 pass

5.1 TROUBLESHOOTING

Problems we *actually* hit.

Every row was met while writing this guide, or is a cause the repository documents. A BLOCK is an HTTP 200 with a `reason`.

YOU SEE	CAUSE AND FIX
`all predefined address pools have been fully subnetted` on `dc up`	Docker has run out of default networks (this machine had 60+). Prune networks you no longer need, or give the project a free subnet: add `networks: { default: { ipam: { config: [{ subnet: 10.77.5.0/24 }] } } }` to `ops.yml`.
Gateway exits: `FATAL: Redis is not reachable at redis://redis:6379`	Seen once, on a Docker host starved of CPU. Run `dc up -d` again; give Docker more CPU and memory if it repeats.
`curl: (7)` right after `up`	First boot applies 96 migrations to a fresh database. The `until` loop in step 1 waits for `/health`.
`BLOCK UNAUTHORIZED` from `/sign`	Wrong or stale signing key. After a restart the log holds several keys: take the last (`tail -1`).

THE NPM PATH, INSTEAD OF COMPOSE

MACOS · LINUX · GIT BASH · IN A FRESH CLONE OF QODIQA-GATEWAY

```
npm ci
QODIQA_SEM_EMBEDDING_ENABLED=false QODIQA_SEM_FT3_ENABLED=false \
QODIQA_SEM_FT2_ENABLED=false \
  npm run dev:sandbox -- --port 3001 --redis-url redis://127.0.0.1:56379
```

It starts disposable Redis and PostgreSQL containers and prints the same signing key; use port 3001 in the commands. With the default `--redis-url` it reuses any Redis already on 6379, so always pass an unused port as above.

YOU SEE	CAUSE AND FIX
`BLOCK MISSING_ARTIFACT`	No `x-qodiqa-artifact` header on `/ai/chat`.
`BLOCK REQUEST_MISMATCH`	The body you sent is not the body you signed. Send exactly what you signed.
`BLOCK EXPIRED` or `BLOCK REPLAY_DETECTED`	Artifacts live at most 60 s and work once. Sign again.
`MALFORMED_JSON` from PowerShell	5.1 strips quotes from an inline `-d '{...}'`. Put the JSON in a file and use `-d "@file"`.
`401 UNAUTHORIZED` on `/audit`, `/evidence`, `/v1/proofs`	This boot does not know your operator key. It is read once at boot: use `ops.yml`, issue the key, then `dc restart gateway`.
`403 MFA_ENROLLMENT_REQUIRED` on `roots/generate`	Manual roots need a TOTP step-up. Do not generate one: wait up to 60 s for the scheduled root.
`transparency.available: false` in the bundle	No signed root covers the entry yet. Re-fetch after a minute.
`CHECKPOINT_ABSENT` in the proof verdict	No signed checkpoint covers the request yet (default: every 1,000 entries or 15 min; `ops.yml` sets 2), or the chain path is withheld: use `&fullChain=true` with the admin key.
`EXPIRED` from `verify.external.js`	Add `--historical` to verify as of the signing time.
`npm run dev:sandbox` ends with `INFERENCE_POOL_STARTUP_FAILED`	Content-screening models are not in the repository. Set the three `QODIQA_SEM_*` variables shown on the left, as Compose does.

5.2 PRODUCTION CHECKLIST

What is built, what *you* must add.

The sandbox is not a deployment. From the repository's production checklist, limitations and roadmap, one status per row.

AREA	STATUS	WHAT TO DO, OR WHERE THE LIMIT IS
Boot validation and secrets	IMPLEMENTED	Production refuses to start on any violation (`BOOT_VALIDATION_FATAL`). Set `NODE_ENV=production` and two different random 64-hex secrets (`QODIQA_KEY_ENCRYPTION_KEY`, `QODIQA_EXECUTION_INVARIANT_SECRET`) from a secret manager.
PostgreSQL and Redis	IMPLEMENTED	Both required. TLS to both, Redis AOF (`appendfsync` everysec or always), `STRICT_REPLAY_PROTECTION=true`. The production profile needs your own validation; power-loss durability is not demonstrated on hardware.
Signing identities, RBAC, operator MFA	IMPLEMENTED	One signing identity per caller, role-based operator keys, TOTP step-up, four-eyes policy governance. SSO (OIDC, SAML, LDAP) validated against test infrastructure only.
Local provider (Ollama)	IMPLEMENTED	Exercised end to end. Adapters for OpenAI, Anthropic, Gemini, Mistral, Azure OpenAI and Bedrock are experimental: simulated transport only.
External anchoring and witnesses	IMPLEMENTED	Opt-in and asynchronous. Without an independent witness the evidence is operator-attested. Whether witnesses are independent is a deployment fact.
Backup, restore, disaster recovery	IMPLEMENTED	Documented tooling and runbooks under `docs/operations/`. Test your restore.
Content screening by meaning	OUT OF SCOPE	Models and reference data are not distributed. Start with `QODIQA_SEM_EMBEDDING_OFF_ACK=true` (production) and treat rule-based screening as one layer.
Closing direct access to providers	OUT OF SCOPE	QODIQA enforces at the gateway; traffic that bypasses it is not intercepted. Close it at the network.
Clustering, failover, throughput figures	OUT OF SCOPE	One instance is the supported deployment. No capacity figure is claimed: size by load testing.
Certification of an implementation	SPECIFIED	Defined in the standard (QODIQA-CERT-2026-001). The gateway is not a certification and makes no one compliant.
Retention policy type; revocation de-duplication; anomaly baselines	PLANNED	On the roadmap. Not supported today.

5.3 WHERE TO GO NEXT, AND WHAT WE FOUND

Next *steps,* honestly.

Read the standard, then the technical overview and the compliance guide. Three behaviours found while testing are listed, not worked around in silence.

WHERE TO GO NEXT

READ	FOR
QODIQA Standard STD-1.0	the normative requirements behind every step here
Technical Overview	architecture, trust boundaries, the evidence chain
Compliance and Self-Assessment Guide	mapping evidence to frameworks; not legal advice

THE REPOSITORIES

REPOSITORY	CONTAINS
`qodiqa/qodiqa-gateway`	gateway, SDKs, `docs/guides/quickstart.md`
`qodiqa/qodiqa-verify`	offline verifiers, recorded sessions, vectors
`qodiqa/qodiqa-anchors`	external anchoring of checkpoints
`qodiqa/qodiqa`	the 32-document specification corpus

Contact: www.qodiqa.com · contact@qodiqa.com. Vulnerabilities go through each repository's `SECURITY.md`.

KNOWN ISSUES FOUND WHILE WRITING THIS GUIDE

ID	WHAT HAPPENS	REPRO IN SHORT
Q-01	The Python proof verifier answers `INVALID` (`TRANSPARENCY_INVALID`) on a proof downloaded from a live sandbox, while the Node verifier answers `VALID_PARTIAL`. The live proof carries the transparency leaf in database spelling (`entry_hash`); the recorded examples use `entryHash`. Camel-casing the keys makes Python agree.	Fetch `/v1/proofs/{id}`, run both `qodiqa-proof-verify` tools.
Q-02	`npm run dev:sandbox` on a clean clone dies with `FATAL INFERENCE_POOL_STARTUP_FAILED`; the README says the sandbox sets the screening flags for you, which only the Compose file does.	`npm ci && npm run dev:sandbox`; works with the three `QODIQA_SEM_*` variables.
Q-03	Observation: a BLOCK from a content rule, DLP or a missing artifact adds no audit-chain entry and no receipt; policy-engine blocks (ABAC, rate limit) and `REQUEST_MISMATCH` do. The README says decisions are appended to the chain.	Compare `GET /audit/chain/integrity` counts around each BLOCK.

Also noted: the stock Compose sandbox gives no way to obtain an operator key, so `ops.yml` is this guide's workaround; and `npm run dev:sandbox` with the default `--redis-url` will reuse any Redis already on 6379. Each is reproducible with the commands in this guide.

QODIQA

Check it yourself.

WWW.QODIQA.COM