DOTCHECK

Pro API

Score media. Issue a report. Hand anyone a Verify link.

VERSION	Pro API v2026.10
DATE	3 September 2026
PAGE	dotcheck.ai/api

Integrator handbook for holders of a Pro API key (`dc_…`). Read it as a procedure: what to do, what Express returns, what you must store next, and what not to invent. Scores are estimates from the same engines as Check — not proof of authenticity. This edition is Docs v2026.10 lockstep. Measured results live on the Model Card and Technical Report, not here.

Petr Jaroch (trading as DotCheck), IČO 03330389, Jenikovice 2, 535 01 Prelouc, Czech Republic

dotcheck.ai/api

Contents

1. How to use this guide

This document is the deep English download for the Pro HTTP API. The short live storefront stays at dotcheck.ai/api. If a curl sample and this PDF disagree on a moving detail, trust `/api` for the host and the examples repo for runnable code — then re-check the route chapter here for **shape** honesty (array vs object, hashes, Confirm refusal).

1.1. Who it is for

You already have (or are about to buy) **Pro**, you can call HTTPS APIs, and you want to score media from your own backend — optionally freeze a report other people can open on DotCheck Verify.

1.2. Who it is not for

- People who only need the Chrome extension, Check in a browser, or the Android app (use those products).
- People hunting held-out accuracy tables (use Docs).
- People who need vendor Confirm (Sightengine / Winston) over HTTP — that path is **first-party only**.

1.3. How to read the chapters

1. **Quickstart** — do the happy path once end-to-end so the rest of the book has a picture in your head.
2. **Concepts** — why the API is shaped the way it is (especially hashes and array responses). Skip this and you will invent wrappers that do not exist.
3. **Authentication** — keys, headers, and what `401` vs `403` means in practice.
4. **Score media** — one modality at a time: what to send, what comes back, what to store, common mistakes.
5. **Reports and verify** — how issuing differs from scoring, and how a stranger checks a report.
6. **Usage and limits** — how not to surprise yourself with `429`.
7. **Errors** — symptom → code → what to change.
8. **Recipes** — copyable pipelines.

> THE THREE STEPS (MEMORIZE THESE)
>
> **1. Score** — call `/analyze-…`. You receive a model estimate (`probability`) and a content identity (`clean_hash` / `sourceSha256`).
> **2. Issue** (optional) — call `POST /report` with that **server** hash and a `kind`. Express freezes the score **it already holds**. You receive a `reportId`.
> **3. Verify** — hand anyone `https://dotcheck.ai/v/{reportId}`. They do not need your API key.

2. Quickstart

Goal: in one sitting, create a key, score one image, optionally issue a report, and open a Verify link. Do not parallelize yet. Do not batch. Confirm the **shapes** with your eyes.

2.1. Step 1 — Be on Pro

Subscribe on dotcheck.ai plans. The API key middleware requires Pro (`403 pro_required` otherwise). Premium can issue reports from Check, but **minting and using** `dc_…` keys is a Pro product.

2.2. Step 2 — Sign in on Check and create a key

1. Open Check → Account and sign in with Google (same account as billing).
2. Create an API key. The plaintext value looks like `dc_live_…` and is shown **once**. Copy it into a secret store (environment variable, vault, CI secret). If you lose it, revoke and create another — you cannot re-display the old plaintext.
3. You may keep up to **five** active keys. Use separate keys for staging vs production if that helps rotation.
4. Keys **never** appear on the public `/api` page. Key create/list/delete over HTTP with the Bearer key itself is refused (`403 firebase_auth_required`) — management stays on Check with Google.

2.3. Step 3 — Score one image

Set `BASE` to the Base URL in Section 4. Send **one** HTTPS image URL:

```
curl -sS -X POST "$BASE/analyze-image" \
  -H "Authorization: Bearer $DOTCHECK_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"image_urls":["https://example.com/photo.jpg"]}'
```

What you must notice in the response:

1. The top-level JSON is an **array**, not `{ "results": [ … ] }`.
2. Element `0` has `probability` (0–1), `clean_hash` (long hex), and usually `engine` / `remaining`.
3. Save `clean_hash` if you might issue later. Saving only the percent is not enough for `POST /report`.

Typical success (abbreviated):

```
[
  {
    "probability": 0.42,
    "clean_hash": "495ad70c4a1e2f3b…",
    "engine": "inhouse@14",
    "remaining": { "left": 29999, "limit": 30000, "scope": "week", "surface": "api" }
  }
]
```

If you parse this as an object and look for `.probability` at the root, you will get `undefined` / null and think the API is broken. It is not — unwrap `[0]` (or loop).

2.4. Step 4 — Optionally issue a report

Only after a successful score. Pass the **same** `clean_hash` Express returned, plus `kind: "image"`. Do **not** send your own percentage field hoping Express will trust it — it will not.

```
curl -sS -X POST "$BASE/report" \
  -H "Authorization: Bearer $DOTCHECK_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"clean_hash":"495ad70c4a1e2f3b…","kind":"image"}'
```

Success is HTTP `201` with `{ "ok": true, "report": { "reportId": "DC-…", "score": 42, … } }`.

2.5. Step 5 — Hand over Verify

The issue JSON does not include a `verifyUrl` field. Construct it from `reportId`:

```
https://dotcheck.ai/v/DC-XXXX-XXXX
```

Open it in a private window. You should see the issued record. The recipient needs no API key.

2.6. Runnable samples

Python and Node scripts that score, optionally issue, and append a `results.jsonl` line: github.com/AIPhoenician/DotCheck/…/examples/pro-api

```
export DOTCHECK_API_KEY=dc_live_…
python score_issue_verify.py --url https://example.com/photo.jpg --issue
```

> **QUICKSTART CHECKLIST**
>
> □ Pro active · □ Key stored as secret · □ Analyze returned a **JSON array** · □ You copied `clean_hash` from element 0 · □ Issue used that hash (not a client %) · □ Verify URL opens without your key · □ Confirm was **not** attempted via the key

3. Concepts

3.1. Same service as Check

The Pro HTTP API is a thin client of the same Express service that powers Check, the Chrome extension, and the Android app. Your key scores media on our in-house inference path and can freeze a verifiable report. There is no second "API-only" model stack and no silent vendor Confirm through the key.

The engines are the same as Docs: Vermeer (Niepce + Janssen), Valla (Rask + Comenius + named tinies; Ma Jianzhong for Chinese), Muybridge (Plateau + Blau), and Helmholtz. Measured holdouts live on the Model Card and Technical Report — not in this guide.

When Check shows a score for an image you uploaded, and when your backend scores the same bytes via `/analyze-image`, you are asking the **same** product truth — different door, same house.

3.2. The score → issue → verify pipeline in detail

Score. Express accepts your media (URL, data URL, or multipart upload), runs the matching engine, meters a Check against your Pro fair-use bag when appropriate, and returns a result that includes:

- a floating `probability` (model estimate), and
- a content identity (`clean_hash` and/or `sourceSha256`) that Express can look up later.

Issue. You ask Express to **freeze** what it already knows about that hash into a durable report with a public `reportId`. Issuing does **not** re-score. Issuing does **not** burn another analysis Check. If Express has never seen that hash from a successful analyze for your account's path, issue fails with `404 report_source_unknown`.

Verify. Anyone with the `reportId` (or a Report/Dossier PDF that embeds it) can open DotCheck Verify. That path is public. It is not authenticated with your API key. It does not burn Checks.

3.3. Probability vs report score vs "percent you calculated"

- **`probability`** — number from 0 to 1 on analyze success. This is what integrators usually store for automation.
- **Report `score`** — integer 0–100 Express writes at issue time from the **server-held** analyze result (typically `round(probability * 100)` under normal model scoring).
- **Anything your client invents** — ignored at issue. If your UI shows 41% because you rounded differently, and Express holds 0.42, the report will show Express's frozen score. That is intentional: Verify must not depend on a caller-supplied percentage.

3.4. Hashes: what to keep

Treat `clean_hash` / `sourceSha256` as the join key between analyze and issue.

MODALITY	WHAT THE HASH REPRESENTS (INTEGRATOR VIEW)
Image	Identity of the scored still (from the analyze path)
Text	Hash of the scored text (`text_hash` / `sourceSha256`)
Video	Composite identity over the **frames you sent** (not a guarantee about an entire remote MP4 file)
Audio	SHA-256 of the uploaded audio bytes
Document	SHA-256 of the whole uploaded file

If you lose the hash, you can score again (another Check burn) — you cannot reconstruct an issue handle from a percent alone.

3.5. Array vs object (the most common integration bug)

ENDPOINT FAMILY	SUCCESS BODY SHAPE
`/analyze-image`, `/analyze-text`	**JSON array** — one element per input
`/analyze-video`, `/analyze-audio`, `/analyze-document`	**JSON object**
`/report`, `/reports`, `/usage`	**JSON object**

Wrong:

```
# Python — DO NOT do this for image/text
data = response.json()
p = data["probability"]   # KeyError / wrong
```

Right:

```
data = response.json()
row = data[0] if isinstance(data, list) else data
p = row["probability"]
h = row.get("clean_hash") or row.get("sourceSha256")
```

3.6. Surface `api`

Requests authenticated with a Pro Bearer key run as surface `api` for fair-use metering on analyze. You cannot set the surface with a custom header. Document scoring is metered as Checks on the document quota (see Section 5.6) but still uses your Pro account.

3.7. What this API is not (read before building)

- **Vendor Confirm** — first-party Check / Android only. Key + Confirm ⟶ `403` `confirm_first_party_only`.
- **SAMPLE reports** — interactive try-it only ⟶ `403` `sample_interactive_only`.
- **Marked file bytes** — document analyze returns scores and `markSignal` geometry for clients that paint locally; the API does not return a marked PDF/DOCX.
- **Covenant soundtrack video** — `/analyze-video` scores the frames **you** extract and send. Soundtrack windows are a first-party Check path, not this JSON body.
- **Key mint/revoke with the key** — Check + Google only.
- **Proof of authenticity** — scores are estimates. Do not tell end users “verified human” or “confirmed AI” from a DotCheck probability alone.

3.8. Fail closed

If inference or the usage store is down, Express returns `503` (`inhouse_unavailable`, `usage_unavailable`, or admission off). Free and API paths do **not** silently fall through to a vendor. Your integration should treat `503` as “retry later / show outage,” not “assume real.”

3.9. List export alignment

Check History / Reports / batch **Export** (CSV · JSON · JSONL) and the example kit `results.jsonl` share the same row fields where applicable. When you warehouse API results, prefer that field vocabulary (`probability`, `kind`, timestamps, hash prefix, status) so Check exports and API rows can sit in one table.

4. Authentication

4.1. Base URL

All Pro API calls in this guide use:

```
https://dotcheck-server-c221c1f32c68.herokuapp.com
```

Use HTTPS only. Do not put the API key in the query string or in a path segment.

4.2. The Authorization header

Every authenticated request:

```
Authorization: Bearer dc_live_…
```

- Scheme is `Bearer` (capital B is conventional; send a normal HTTP Authorization header).
- One space after `Bearer`.
- No quotes around the key.
- No `X-Api-Key` alternate header — this product uses Bearer.

4.3. Creating, storing, and rotating keys

1. Create on Check → Account while signed in with Google.
2. Store plaintext in a secret manager. Inject into workers as `DOTCHECK_API_KEY` (the examples use that name).
3. Rotate by creating a new key, deploying it, then **revoking** the old key on Check. Revoked keys yield `401`.
4. Cap: **five** active keys per account.

4.4. What auth errors mean in practice

YOU SEE	WHAT TO FIX
`401` / `invalid_token`	Missing header, typo, revoked key, or `Authorization` present but not a Bearer token
`403` / `pro_required`	Account is not Pro — finish subscribe / wait for billing to reflect
`503` / `usage_unavailable`	Auth could not reach the usage store — fail closed; retry later

A **broken** Authorization header must not fall through to the anonymous free bag. If you send garbage credentials, expect `401`, not a silent guest score.

4.5. Environment checklist

□ `DOTCHECK_API_KEY` set in the process that calls Express
□ `DOTCHECK_BASE_URL` optional override for the Base URL (examples support this)
□ Logs never print the full key (log `dc_live_…` + last four only, if you must)
□ Frontend browsers do **not** embed the Pro key — call Express from your backend

5. Score media

This chapter is the reference for every analyze endpoint. Sync multi-input batches stay small: **at most 30** image URLs or texts per request on paid tiers. For bulk async when volume hurts, use Contact (Custom work) — do not invent a private pagination protocol.
Before any large job: `GET /usage` (Section 7) and confirm `remaining.left`.

5.1. Shared analyze rules

- Send `Content-Type` correctly (`application/json` vs multipart).
- On success, always branch on array vs object (Section 3).
- Persist `clean_hash` / `sourceSha256` when you may issue.
- Read `remaining` when present — it is the same bag your next call will hit.
- Do not send `confirmRetest` on the API key path.

5.2. Image — `POST /analyze-image`

5.2.1. Purpose

Score one or more still images you can address as HTTPS URLs or `data:` URLs. Express packs each still into the product pair (max-side 256 transport + native 224 center) and scores Vermeer@14.2.

5.2.2. Request

- **Method / path:** `POST /analyze-image`
- **Header:** `Authorization: Bearer …` · `Content-Type: application/json`
- **Body:** `{ "image_urls": [ "https://…", … ] }` — a string is also accepted and treated as a one-element list.
- **Batch cap:** paid ⟶ 30 (`too_many_urls` + `max` if exceeded).

5.2.3. Step by step

1. Build a JSON body with the URLs you want scored.
2. POST to `/analyze-image`.
3. Parse JSON as an **array**.
4. For each element: if it has `probability`, store it with `clean_hash`; if it carries an element-level error, handle that row without assuming a hash exists.
5. Optionally issue per hash with `kind: "image"`.

5.2.4. Success fields (per array element)

FIELD	MEANING
`probability`	Model estimate 0–1
`clean_hash`	Pass to `POST /report`
`engine` / `engine_label`	Which in-house wire/label produced the score
`remaining`	Fair-use snapshot: `left`, `limit`, `scope`, `surface`

5.2.5. curl

```
curl -sS -X POST "$BASE/analyze-image" \
  -H "Authorization: Bearer $DOTCHECK_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"image_urls":["https://example.com/photo.jpg"]}'
```

5.2.6. Billing

One Check burn per successfully scored URL (shared Pro weekly bag on surface `api`).

5.2.7. Common mistakes

- Treating the body as an object.
- Issuing with a hash you invented locally instead of the returned `clean_hash`.
- Passing an HTTP (non-TLS) URL where HTTPS is required ⟶ `invalid_url`.
- Setting `confirmRetest` ⟶ `403 confirm_first_party_only`.

5.2.8. Errors to expect

`400 image_urls_required` · `invalid_url` · `too_many_urls` · `confirm_kind_unsupported` (GIF Confirm path) · `403 confirm_first_party_only` · `429 daily_free_limit` · `503 inhouse_unavailable` / `usage_unavailable`.

5.3. Text — `POST /analyze-text`

5.3.1. Purpose

Score one or more text strings for AI-likeness of the **writing**, not truth of the claims.

5.3.2. Request

- **Body:** `{ "texts": [ "…", … ] }`
- **Languages scored:** English, Chinese, Spanish, French, Portuguese, German, Italian, Dutch.
- One array element is one Check. Latin scores from **520** characters; Simplified Chinese from **120**. Express packs hops inside the string (Latin 520–1100 / zh-Hans 120–1000) and returns one probability. Pastes above **60,000** characters are truncated (`truncated: true`).
- Unsupported language ⟶ that **element** reports `error: unsupported_language` and does **not** burn a Check for that item.

5.3.3. Step by step

1. Send one language per string when possible (do not concatenate mixed-language blobs if you can avoid it).
2. Parse the **array** response.
3. For unsupported rows, translate offline or skip — then resend.
4. Keep `clean_hash` (same hex as `text_hash` / `sourceSha256`) for issue with `kind: "text"`.

5.3.4. curl

```
curl -sS -X POST "$BASE/analyze-text" \
  -H "Authorization: Bearer $DOTCHECK_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"texts":["Sample paragraph to score."]}'
```

5.3.5. Common mistakes

- Assuming every array element always has `probability`.

- Issuing `kind: "image"` for a text hash.
- Expecting DetectGPT-style "claim verification" — Valla scores writing style/features, not factual correctness.

5.3.6. Errors

`400` `texts_required` · `too_many_texts` · `403` `confirm_first_party_only` · `429` / `503` as image.

5.4. Video — `POST /analyze-video`

5.4.1. Purpose

Score a video **from frames you extract**. You are responsible for sampling frames and sending them as image URLs or `data:` URLs.

5.4.2. Request

- **Body JSON:** `video_url` (stable locator: HTTPS media/CDN, YouTube, or `xvid:` / `x:` id) plus `frames` (array). Site chrome such as `https://x.com/home` is refused (`unstable_video_bag`).
- **Frame cap:** maximum **12** (`too_many_frames` + `max`). Short clips use 2 stamps.
- **Success shape: object** (not array) with aggregated `probability`, composite `clean_hash`, `frame_results`, `remaining`.

5.4.3. Step by step

1. Decide a sampling strategy (pair2: two inner stamps on short clips). Stay ≤ 12.
2. Encode frames as HTTPS or `…`.
3. POST `/analyze-video`.
4. Read top-level `probability` and `clean_hash`.
5. Issue with `kind: "video"` if needed.
6. File Verify (`hashBasis: file`, `POST /report/verify` → `match: strong`) only if this issue request attached native MP4/GIF bytes (`multipart` field `file`) or the score was a GIF raw SHA. Frames-only / identity locators stay `derived` (link/ID). A watch URL or junk body is not a native file — issue still `201` as derived.

5.4.4. Honesty

Soundtrack Covenant windows are **not** part of this endpoint. If you need that first-party path, use Check — do not pretend `/analyze-video` did it.

5.4.5. curl

```
curl -sS -X POST "$BASE/analyze-video" \
  -H "Authorization: Bearer $DOTCHECK_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"video_url":"https://example.com/clip.mp4","frames":["…"]}'
```

5.4.6. Billing

One Check burn for the call (not one per frame).

5.4.7. Errors

`400` `video_url_required` · `frames_required` · `too_many_frames` · `unstable_video_bag` · `confirm_kind_unsupported` · `429` · `503` `inhouse_unavailable`.

5.5. Audio — `POST /analyze-audio`

5.5.1. Purpose
Score an uploaded audio file (speech/music clip as supported by the product).

5.5.2. Request
- **Multipart field name:** `audio`
- **Accepted:** MIME `audio/*` or extensions `.wav` · `.mp3` · `.m4a` · `.ogg` · `.webm`
- **Success:** object with `probability`, `clean_hash`, `sourceSha256` (file hash), `engine`, `remaining`, optional `cached`

5.5.3. Step by step
1. POST multipart with the file field named exactly `audio`.
2. On success, store hash + probability.
3. Issue with `kind: "audio"` if needed.
4. Note: a warm cache hit still burns a Check — caching speeds the answer; it does not make analyses free.

5.5.4. curl

```
curl -sS -X POST "$BASE/analyze-audio" \
  -H "Authorization: Bearer $DOTCHECK_API_KEY" \
  -F "audio=@sample.wav"
```

5.5.5. Errors
`400` `no_audio` · `invalid_audio_type` · `empty_audio` · `429` · `503` `usage_unavailable` · `inhouse_unavailable` · `audio_engine_unavailable`.

5.6. Document — `POST /analyze-document`

5.6.1. Purpose
Score a PDF / DOCX / PPTX / TXT workshop file. Express extracts scoreable text/images/video parts, meters Checks for items that actually receive a probability, and returns aggregate fields plus `markSignal` geometry for first-party painters.

5.6.2. Request
- **Multipart field name:** `document`
- **Auth:** Bearer key must resolve to a uid (`401` `auth_required` otherwise).
- **Success:** object — count-weighted `probability`, modality means / arrays, `markSignal`, `sourceSha256` (whole file), `confirmEligible`, `remaining`, `documents`, `checksBilled`, `checksIncluded`.

5.6.3. Step by step
1. Ensure the file is not encrypted/corrupt and fits paid caps (product facts: up to **600** pages and **400** items on paid).
2. POST multipart `document=@file`.
3. Read `checksBilled` — that is how many Checks this file consumed.
4. Read `sourceSha256` for issue with `kind: "document"`.
5. Do **not** expect a marked PDF in the JSON. If your product paints marks, use `markSignal` on your side (as Check does) — that painting is outside this API response.

5.6.4. Billing honesty

Express **gates** before inference when it can estimate required Checks (`checksRequired` on some error paths). Items skipped (too short, unsupported language, etc.) are not billed. Warm cache hits still bill like fresh scores for items that count.

5.6.5. curl

```
curl -sS -X POST "$BASE/analyze-document" \
  -H "Authorization: Bearer $DOTCHECK_API_KEY" \
  -F "document=@sample.pdf"
```

5.6.6. Errors

`400` `no_document` · `page_limit` · `unsupported_format` · `document_encrypted` · `document_corrupt` · `document_no_scoreable_content` · `document_too_many_items` (+ `limit`) · `408` `document_timeout` · `413` `file_too_large` · `429` (+ `checksRequired`) · `503` `ffmpeg_unavailable` · `inhouse_unavailable` · `usage_unavailable`.

6. Reports and verify

Scoring answers “what does the model say right now?” Issuing answers “can I freeze that into a record someone else can open later?” Keep those jobs separate in your code.

6.1. Issue — `POST /report`

6.1.1. When to issue

After a **successful** analyze for content you want to hand over. Skip issue for purely internal automation that never shows Verify.

6.1.2. Request

- **JSON:** Bearer + `Content-Type: application/json`. Body (required): `kind` ∈ `image` · `video` · `audio` · `text` · `document`, and a hash via `clean_hash` **or** `sourceSha256` (aliases like `cleanHash` / `source_sha256` are accepted).
- **Multipart (optional, video file Verify):** `multipart/form-data` with the same fields plus `file` = native MP4 (ISO BMFF `ftyp`) or GIF. Express hashes those bytes only when magic matches; otherwise the report stays `derived` (still `201`).
- **Body (optional):** `fileName`, `reference`, `format`, `batchId`, `batchSize`, `supersedes`.
- **Ignored:** client `probability` / `score` / Confirm payloads — Express freezes server-held truth only.

6.1.3. Step by step

1. Take `clean_hash` / `sourceSha256` from the analyze response (array element or object).
2. Choose the matching `kind` (mismatch ⟶ `invalid_kind` or a useless record).
3. POST `/report` (JSON, or multipart when attaching native video bytes).
4. On `201`, store `report.reportId`, `hashBasis`, `stampCount`, `reportFormatVersion` (v2) and build the Verify URL.
5. On `404 report_source_unknown`, you are issuing a hash Express does not hold — score again, then issue; do not retry issue in a tight loop with a made-up hash.

6.1.4. Success (`201`)

```
{
  "ok": true,
  "report": {
    "reportId": "DC-XXXX-XXXX",
    "score": 42,
    "probability": 0.42,
    "kind": "image",
    "hashBasis": "prepared",
    "stampCount": null,
    "reportFormatVersion": 2,
    "issuedAt": "2026-08-07T12:00:05.000Z",
    "sourceSha256": "495ad70c…"
  }
}
```

`hashBasis` is `file` · `prepared` · `derived`. Text is always `derived`. Video is `file` only when this request attached native MP4/GIF (or GIF raw SHA from analyze). `stampCount` is 1–12 on video when Express froze stamp count.

Owner issue/list shapes may also include engines, Confirm metadata when present from first-party flows, withdraw fields, `sourceSha256Prefix`, batch fields, and format version. Public verify **strips** the full `sourceSha256`.

6.1.5. Verify URL (construct yourself)

```
https://dotcheck.ai/v/{reportId}
```

The issue JSON does not include `verifyUrl`. Concatenate the origin and `reportId` exactly (preserve the `DC-` prefix; no trailing spaces).

6.1.6. curl — JSON issue

```
curl -sS -X POST "$BASE/report" \
  -H "Authorization: Bearer $DOTCHECK_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"clean_hash":"495ad70c…","kind":"image"}'
```

6.1.7. curl — video attach then file Verify

```
curl -sS -X POST "$BASE/report" \
  -H "Authorization: Bearer $DOTCHECK_API_KEY" \
  -F "kind=video" \
  -F "clean_hash=495ad70c…" \
  -F "file=@clip.mp4"

curl -sS -X POST "$BASE/report/verify" \
  -F "reportId=DC-XXXX-XXXX" \
  -F "media=@clip.mp4"
```

Expect `hashBasis: file` on issue and `match: "strong"` on verify when the bytes are the same native MP4/GIF.

6.1.8. Issue errors

`401 auth_required` · `403 premium_required` (paid tier — Pro keys qualify) · `404 report_source_unknown` · `400 invalid_hash` · `invalid_kind` · `invalid_supersedes` · `503 usage_unavailable`.

6.2. List — `GET /reports`

List reports for the authenticated account. Query: optional `q`, `kind`, `limit`, `skip`.

```
curl -sS -H "Authorization: Bearer $DOTCHECK_API_KEY" "$BASE/reports"
```

```
{ "ok": true, "total": 1, "items": [{ "reportId": "DC-XXXX-XXXX", "kind": "image", "score":
42 }] }
```

Use this for admin UIs. For warehouses, you may also mirror Check list Export formats on the website.

6.3. Withdraw — `POST /report/:id/withdraw`

Marks a report withdrawn for future Verify checks. It does **not** delete history for your account UI.

- Optional body: `reason` / `withdrawnReason`
- Success: `{ "ok": true, "report": … }` with withdraw timestamps set

Tell recipients that a withdrawn report should not be treated as a live hand-over.

6.4. Public verify (no API key)

Recipients and auditors use DotCheck Verify without your secret.

CALL	ROLE
Open `/v/{reportId}`	Human Verify desk in the browser
`GET /report/:id`	Machine lookup — public shape, no full hash
`POST /report/verify`	Multipart `media` (+ optional `reportId`) — match / siblings; no Check burn

Strong file-match Verify (`POST /report/verify` → `match: strong`) applies when `hashBasis` is `file` or `prepared` (image / audio / document; video only if this issue attached native MP4/GIF or GIF raw SHA). Text stays `derived` (link/ID). Frames-only video stays `derived`. Burst pressure may rate-limit — back off and retry.

6.5. SAMPLE is not part of the key contract

`POST /report/sample` exists for interactive try-it on Reports. Via API key it returns `403 sample_interactive_only`. Do not build production flows on SAMPLE.

7. Usage and limits

7.1. Why poll `/usage`

`429` means you already hit a wall. `GET /usage` lets you schedule work **before** the wall: drain overnight queues, pause workers, or show "budget low" in your admin UI.

7.2. `GET /usage`

```
curl -sS -H "Authorization: Bearer $DOTCHECK_API_KEY" "$BASE/usage"
```

Typical success (abbreviated):

```
{
  "tier": "pro",
  "surface": "api",
  "remaining": { "left": 29999, "limit": 30000, "scope": "week", "surface": "api" }
}
```

- `tier` - should reflect Pro for key users
- `surface` - `"api"` when the Bearer key authenticated the call
- `remaining` - `left`, `limit`, `scope`, `surface` (Pro may also expose Confirm remaining for the **account**, even though Confirm is not callable via the key)
- `documents` / `shares` - adjacent bags when applicable

7.3. Limits that matter for integrators

LIMIT	VALUE / BEHAVIOR
Pro analyses / week	Up to **30,000** on the shared Pro bag
HTTP analyze batch	**30** URLs or texts per request (paid)
Video frames / request	**12** maximum
Active API keys	**5** per account
Document pages / items (paid)	**600** pages · **400** items (product facts)
Paid burst limiter	**15,000** requests / 15 minutes → `fair_use_limit`

7.4. Weekly budget vs Retry-After

- **Weekly FUP 429** (`daily_free_limit`) — read `remaining.left` and `remaining.scope: week`. Resets on the next weekly window. Do **not** invent an immediate `Retry-After` that claims "try in 2 seconds." The error code stays `daily_free_limit` for compatibility.
- **Admission / large-body pressure** — may send `Retry-After: 2` with `503`. That is a short pause, not a weekly budget signal. Handle the two differently in code.

7.5. Concurrency advice

Start with modest parallelism. Watch `remaining.left` and burst `fair_use_limit`. Prefer fewer larger batches (≤ 30) over thousands of single-URL hammering if your network allows — but never exceed the batch cap.

7.6. When you outgrow sync HTTP

Contact dotcheck.ai/contact with subject **Custom work** for bulk/async needs. Do not scrape undocumented endpoints.

8. Errors

8.1. How to read an error

Express returns JSON `{ "error": "<code>" }` and a matching HTTP status. Log **both**. Branch your retry logic on the code, not only on "non-200."

8.2. Status families

- `2xx` — success (`201` on issue)
- `4xx` — your request, auth, or quota
- `5xx` — infrastructure / fail closed

8.3. Stable product codes

CODE	HTTP	WHEN
`invalid_token`	`401`	Missing or revoked key
`auth_required`	`401`	Uid required (document / report)
`pro_required`	`403`	Not on Pro
`premium_required`	`403`	Paid tier required for issue
`confirm_first_party_only`	`403`	Confirm via API key
`sample_interactive_only`	`403`	SAMPLE via API key
`firebase_auth_required`	`403`	Key management with Bearer key
`report_source_unknown`	`404`	Issue hash not known
`image_urls_required` / `texts_required` / `frames_required` / `no_audio` / `no_document`	`400`	Missing input
`too_many_urls` / `too_many_texts` / `too_many_frames`	`400`	Batch / frame cap (+ `max`)
`unstable_video_bag`	`400`	Video locator is a site page such as `/home`, not a media id
`unsupported_language`	—	Text element not scored (no burn)
`daily_free_limit`	`429`	Weekly analysis budget — read `remaining`
`fair_use_limit`	`429`	Short burst limiter
`inhouse_unavailable`	`503`	Inference down — fail closed
`usage_unavailable`	`503`	Usage store down — fail closed
`audio_engine_unavailable`	`503`	Audio path not available

8.4. Symptom playbook

SYMPTOM	LIKELY FIX
`probability` is undefined	You forgot to unwrap the **array** on image/text
`report_source_unknown`	Issue only after analyze; use returned hash; matching `kind`
Always `pro_required`	Billing/account not Pro; wrong Google account
`confirm_first_party_only`	Remove Confirm from the API integration — use Check
Sudden `429` with `left: 0`	Wait for the weekly reset or reduce volume; poll `/usage`
`503` with `Retry-After: 2`	Short backoff, then retry; do not mark weekly budget exhausted
`503` without Retry-After	Outage / fail closed — pause and alert, do not invent vendor fallback

9. Recipes

9.1. Recipe A — Score only (automation)

Use when you never show Verify.

1. Call the matching `/analyze-*`.
2. Persist `probability`, hash, `engine`, `remaining.left`, and your own job id.
3. Stop. No `/report`.

9.2. Recipe B — Score → issue → verify (hand-over)

Use when a human must open a record later.

1. Analyze and extract hash from the correct shape (array vs object).
2. `POST /report` with hash + `kind`.
3. Store `reportId`.
4. Send `https://dotcheck.ai/v/{reportId}` (email, ticket, CMS field).
5. Optionally `GET /report/{reportId}` from a backend that does not hold the key in the browser.

9.3. Recipe C — Poll before bulk

1. `GET /usage`.
2. If `remaining.left` < planned burns, split the job across weeks or stop.
3. Run batches of ≤ 30.
4. On `429` `daily_free_limit`, checkpoint progress and wake after the weekly reset.
5. On `503` + `Retry-After: 2`, sleep 2s and retry the **same** request budgeted item.

9.4. Recipe D — Text language refusal

1. Send strings in supported languages only when you can.
2. If an element returns `unsupported_language`, do not treat it as probability 0 — it was not scored.
3. Translate or filter, then resend that element.

9.5. Recipe E — Video frames you control

1. Extract ≤ 12 representative JPEGs (pair2).
2. Send as `frames` with a `video_url` locator.
3. Issue as `kind: "video"` if handing over.
4. To let a recipient file-Verify the **container**, attach native MP4/GIF as multipart `file` on that same `POST /report`. Then `POST /report/verify` with `media` = those bytes ⟶ `match: strong` when `hashBasis` is `file`.
5. Do not claim soundtrack analysis happened.
6. Do not attach a watch URL or text as `file` — magic must be GIF or ISO BMFF `ftyp`; otherwise the report stays derived.

9.6. Recipe F — Document then issue

1. `POST /analyze-document` with field `document`.
2. Read `checksBilled` and `sourceSha256`.
3. `POST /report` with `kind: "document"`.
4. Paint marks only in your own client if required — the API will not return a marked file.

9.7. Recipe G — Safe logging

Log request id / your job id, HTTP status, `error` code, `remaining.left`, and hash **prefixes** (first 12 chars). Never log the full Bearer token or full raw data URLs.

10. Related

- Live storefront: dotcheck.ai/api
- Engine numbers (not this guide): Model Card · Technical Report
- Runnable examples: examples/pro-api
- Plans: dotcheck.ai plans
- Custom wiring / higher volume: Contact (subject Custom work)

)