HTTP server and Docker

For every other language, and for a service of your own: the server's Docker image puts Weblitz and SSRS2's reports behind one HTTP API. A page or an address to PDF, Excel, Word, PowerPoint or a picture; a report with its rows as JSON to any of SSRS2's formats; a report's problems; Markdown to a deck or a document; another tool's report to RDLC. The file it answers is the file the .NET packages write with the same options and data.

The image

```
docker run --rm -p 8080:8080 -e SSRS2_LICENSE ghcr.io/antoinegou/ssrs2-weblitz-server:2.0.0-rc.2
```

`ghcr.io/antoinegou/ssrs2-weblitz-server:2.0.0-rc.2`, public on GitHub's container registry, for linux/amd64 and linux/arm64, answers everything; the two smaller editions below answer a part. It carries the Noto fonts for the scripts and emoji the packages do not bring, listens on port 8080 and runs as the image's own user, never root; put TLS in front of it. Each request runs in a worker process of the server's, so a report whose code never ends, or brings its process down, costs that worker and not the server. `GET /health` answers 503 while the engines warm up, then 200.

The image comes in three editions from the same code, each public, for linux/amd64 and linux/arm64:

Image	Edition	Answers	Status
`ghcr.io/antoinegou/ssrs2-server:2.0.0-rc.2`	reports	/v1/report, /v1/check, /v1/markdown, /v1/import	published
`ghcr.io/antoinegou/weblitz-server:2.0.0-rc.2`	html	/v1/html	published
`ghcr.io/antoinegou/ssrs2-weblitz-server:2.0.0-rc.2`	full	every /v1 route	published

From 2.0.0-beta.11 on, `ssrs2-server` answers reports only: a service printing HTML takes `weblitz-server`, or `ssrs2-weblitz-server` for both. An operation an edition does not have answers 404 with the code `unknownOp`, and `GET /v1/info` says its `edition`.

A page and a report

```
# a page as it is, with the engine's defaults
curl -H 'Content-Type: text/html' --data-binary @page.html
http://localhost:8080/v1/html -o page.pdf

# with options: Puppeteer's page.pdf() names, and the engine's own
curl -H 'Content-Type: application/json' \
     -d '{"html": "<h1>Hello</h1>", "options": {"format": "A4", "margin": "15mm",
"printBackground": true}}' \
     http://localhost:8080/v1/html -o hello.pdf

# a web page as a picture
curl -H 'Content-Type: application/json' \
     -d '{"url": "https://example.com", "output": "png", "options": {"viewport":
{"width": 1280, "deviceScaleFactor": 2}}}' \
     http://localhost:8080/v1/html -o example.png

# a report with its rows: the request as a field, the definition as a file
curl -F 'request={"format": "PDF", "datasets": {"Items": [{"Description": "Widget",
"Price": 9.5, "Qty": 2}]}}' \
     -F report=@Invoice.rdlc http://localhost:8080/v1/report -o invoice.pdf

# Markdown to a PowerPoint deck; another tool's report to RDLC
curl -F 'request={"format": "PPTX", "layout": "slides"}' -F deck=@deck.md
http://localhost:8080/v1/markdown -o deck.pptx
curl -F file=@Orders.frx http://localhost:8080/v1/import -o Orders.rdlc
```

Endpoints

Request	What it does
`POST /v1/html`	A page (`html`) or an address (`url`) to `pdf` (the default), `xlsx`, `docx`, `pptx`, `png`, `jpeg`, `webp` or `tiff` (`output`)
`POST /v1/report`	A report (`definition`) to a `format`, with `datasets`, `parameters`, `deviceInfo`, `subreports`

Request	What it does
`POST /v1/check`	A report's problems as notes, no file
`POST /v1/markdown`	Markdown to a deck (`PPTX`) or a document (`PDF`, `WORDOPENXML`)
`POST /v1/import`	Another tool's report, or a document, to RDLC
`GET /v1/info`	Versions, operations, formats, licence and platform
`GET /health`	`503` while the engines warm up, then `200`

A request is JSON (camelCase names; a field the request does not have is refused, so a typo is not ignored), or multipart with a `request` field holding the JSON and the files beside it, or, for `/v1/html`, a page sent as `text/html`. Every request may give `timeoutMs`, which the server's limit caps.

What it answers

The file as the body, with its `Content-Type`, and the result in headers: `SSRS2-Pages`, `SSRS2-Notes` (how many notes), `SSRS2-Trial` (true without a licence key), `SSRS2-Elapsed-Ms`, and `SSRS2-Result`, the whole result as one line of JSON. With `Accept: application/json` the answer is the result as JSON, the file in base64: ask for it when the notes matter. A page's or a report's own problems are notes, never errors.

An error is `{"ok": false, "error": {"code", "message"}}` with its status: 400 `badRequest`, 401 `unauthorized`, 403 `refused` (a private address, a `file:` base URL, a network the server does not allow), 404 `unknownOp`, 413 `tooLarge`, 422 `invalidReport`, 500 `renderFailed` or `workerCrashed`, 503 `busy`, 504 `timeout`.

Settings

Setting	
`SSRS2_LICENSE`	The licence key. Without it, PDF and image pages carry the trial watermark.
`SSRS2_SERVER_TOKEN`	When set, every `/v1` request carries `Authorization: Bearer <token>`; `/health` stays open.
`Server__Workers`	Worker processes the requests run in (1 by default).

Setting	
`Service__Concurrency`	Renders at once, across the workers (half the processors by default).
`Service__MaxTimeoutMs`	The longest a request may take (120,000 by default).
`Service__Network__AllowedHosts__0`	Hosts a page may load beyond its own address's, one setting each.

Settings come from the environment, a double underscore for each level. The defaults are for callers the server does not know: a page given as `html` loads nothing from the network and an address loads only from its own host, unless `Service__Network__AllowedHosts` opens more; private addresses and `file:` base URLs are refused; report code is restricted; a worker sees none of the server's secrets. Without a token the server answers anyone who reaches it: keep it on a private network then.