Package: aurora (via r-universe)

June 24, 2026

Title Build Stateless Web Apps with 'plumber2'

Version 0.1.12

Description A scaffolding and deployment toolkit for building stateless web applications in R on top of the 'plumber2' web framework (<https://plumber2.posit.co/>). The UI is authored with 'bslib' and compiled to a static HTML asset at build time, while 'plumber2' serves the assets and exposes JSON API routes. Provides functions to scaffold app skeletons, run them locally, and generate Dockerfiles and images suitable for 'ShinyProxy' or plain Docker.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1)

Imports cli, fs, glue, htmltools, jsonlite, plumber2 (>= 0.2.0), rlang, yaml

Suggests brand.yml, bslib (>= 0.9.0), callr, config, fiery, geojsonsf, httpuv, httr2, jose, knitr, later, nanoparquet, pak, reqres, rmarkdown, sf, shiny, sodium, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/aurora-govpe/aurora-rpkg, https://aurora-govpe.github.io/aurora-rpkg/

BugReports https://github.com/aurora-govpe/aurora-rpkg/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Andre Leite [aut, cre], Marcos Wasilew [aut], Hugo Vasconcelos [aut], Carlos Amorin [aut], Diogo Bezerra [aut], Júlia Nascimento Barreto [aut]

Maintainer Andre Leite <leite@castlab.org>

Repository https://cran.r-universe.dev

Date/Publication 2026-06-24 09:10:09 UTC

RemoteUrl https://github.com/cran/aurora

RemoteRef HEAD

RemoteSha 61219d45b186f9a90470fc37f4101513e8a07c11

Contents

`aurora_add_route` *Add an API route to an aurora app*

Description

Generates an annotated **plumber2** route file under `routers/`. Because `aurora_app()` parses every file in `routers/`, no manifest update is needed. The handler's URL embeds the `mount` prefix directly in its annotation, so there is no runtime path injection.

Usage

```
aurora_add_route(name, mount = NULL, dir = ".")
```

Arguments

name	Route name; becomes routers/<name>.R.
mount	URL prefix for the route. Defaults to /api/<name>.
dir	App directory (canonical aurora layout).

Value

The created route file path, invisibly.

---

aurora_app *Assemble an aurora app as a plumber2 API*

Description

Convention-based assembly: builds the UI (optional), sources helpers/*.R, parses every routers/*.R into a **plumber2** API, and serves the www/ directory at /. No _aurora.yml is required; an optional manifest only overrides a few keys.

Usage

```
aurora_app(
  dir = ".",
  rebuild_ui = TRUE,
  host = "127.0.0.1",
  port = 8000L,
  otel = NULL,
  verbose = NULL,
  attach = NULL
)
```

Arguments

dir	App directory (canonical aurora layout).
rebuild_ui	Whether to rebuild the static UI before assembling.
host, port	Bind address/port baked into the API object. Usually overridden by aurora_run().
otel	Wire OpenTelemetry logging (api_logger(logger_otel())) so logs join **plumber2**'s automatic request spans and metrics. NULL (default) resolves from _aurora.yml (otel:) then the AURORA_OTEL environment variable, falling back to FALSE. Wiring is a no-op until the **otel** package is enabled in the environment, so it is safe to leave on.
verbose	Emit a per-step **cli** log (one line per sourced helper / parsed router, otel wiring). NULL (default) resolves from options(aurora.verbose) then the AURORA_VERBOSE env var, falling back to FALSE (quiet: a single assembly-summary line). Errors and warnings always print.

attach — Attach the runtime packages declared in `_aurora.yml` (`packages:`) before sourcing helpers, so helpers and handlers can use their functions unqualified (mirrors a plumber-v1 `library()` block). `NULL` (default) resolves from `_aurora.yml` (`attach:`) then `AURORA_ATTACH`, falling back to `FALSE`. Off by default to keep assembly thin; the `packages:` list is the runtime set (no `shiny`/`bslib`), so it is safe to enable. See ADR-012.

Details

Extra static mounts can be declared in `_aurora.yml` under `statics:` – a map of URL prefix to directory, served at that prefix (in addition to `www/` at `/`). This is for assets shared across apps from a server-side directory mounted as a volume; e.g. `statics: /assets: /srv/shared` serves `/srv/shared/logo.png` at `/assets/logo.png`. Relative paths resolve against the app root; a missing directory (e.g. an unmounted volume) is skipped with a warning so the app still starts. See ADR-018.

Each handler's URL is taken verbatim from its `#* @get /...` annotation, so the route prefix is embedded by `aurora_add_route()` at scaffold time – there is no runtime path injection here.

Value

An object of class `aurora_app`.

---

aurora_auth_jwt *JWT-cookie authentication scheme*

Description

aurora's auth is *pluggable* and never baked into `aurora_app()`'s core path. This is the one provided scheme: a stateless JSON Web Token signed with **jose** (HMAC) and delivered as an `HttpOnly` cookie. It is the plumber2 translation of the reference app's v1 `@filter` JWT scheme.

Usage

```
aurora_auth_jwt(
  secret = Sys.getenv("AURORA_JWT_SECRET"),
  cookie = "token",
  expiry = 28800L
)
```

Arguments

secret — Secret used to sign/verify tokens (string or raw). Prefer supplying it via an environment variable rather than hardcoding.

cookie — Name of the cookie carrying the token.

expiry — Token lifetime in seconds.

Details

The companion helpers operate on the scheme object:

- `aurora_jwt_token()` mints a signed token (at login).
- `aurora_set_auth_cookie()` / `aurora_clear_auth_cookie()` manage the cookie.
- `aurora_jwt_guard()` is the gate: call it from a `@header` handler on `/api/*` and it rejects unauthenticated requests with a `401`.

Auth is wired entirely in your app's annotated router files (a `@header` guard + public `/auth/*` routes), so `aurora_app()` needs no auth knowledge. See the `auth` template (`aurora_create_app()`).

Value

An object of class `aurora_auth_jwt`.

Examples

```
auth <- aurora_auth_jwt(secret = "dev-only-secret")
tok <- aurora_jwt_token(auth, list(user = "alice"))
aurora_jwt_decode(auth, tok)$user
```

---

aurora_build_image *Build (and optionally push) a Docker image for an aurora app*

Description

Thin wrapper around the `docker` CLI. Requires a `Dockerfile` (see `aurora_dockerfile()`) and a working Docker installation on `PATH`.

Usage

```
aurora_build_image(
  dir = ".",
  tag,
  push = FALSE,
  dockerfile = "Dockerfile",
  platform = "linux/amd64"
)
```

Arguments

`dir`	App directory (build context).
`tag`	Image tag, e.g. `"myorg/myapp:latest"`.
`push`	Whether to `docker push` after a successful build.
`dockerfile`	Path to the Dockerfile relative to `dir`.
`platform`	Target platform passed to `docker build --platform`. Defaults to `"linux/amd64"` so images built on Apple Silicon run on the usual x86-64 servers. Use `NULL` to build for the host architecture.

Value

The image tag, invisibly.

---

aurora_build_ui *Build the static UI for an aurora app*

Description

Sources the app's `build_ui.R` (root), which must define a `build_ui()` function returning an **htmltools** tag, and writes the rendered HTML to `www/index.html` via `htmltools::save_html()`. Author the layout with **bslib**; ship static HTML.

Usage

```
aurora_build_ui(dir = ".")
```

Arguments

`dir`	App directory (canonical aurora layout).

Value

The output file path, invisibly.

---

aurora_check *Check an aurora app for common problems*

Description

Static "doctor" for the canonical layout. Reports issues that otherwise surface late (often only when building or running the container):

Usage

```
aurora_check(dir = ".")
```

Arguments

`dir`	App directory (canonical aurora layout).

Details

- **UI code in runtime helpers** – **shiny/bslib/htmltools** usage in `helpers/` (sourced at request time) pulls the UI stack into the runtime image. UI code belongs in `build_ui.R`/`ui_modules/` (build time).
- **Undeclared packages** – packages referenced in `routers//helpers/` but absent from `_aurora.yml` `packages:` (when that list is present), so the image would miss them.
- **Missing prebuilt UI** – no `www/index.html` (the container serves it and does not rebuild).

Value

Invisibly, a data frame of findings (`level`, `message`); also printed via **cli**.

`aurora_component` *Wire a UI element to a JSON API endpoint*

Description

A thin markup helper: emits an **htmltools** element carrying its API endpoint as a `data-endpoint` attribute (plus any extra attributes you pass). Your app's feature JavaScript reads `element.dataset.endpoint`, fetches it with the `window.aurora` runtime (`aurora.json(...)`), and renders however it likes.

Usage

```
aurora_component(endpoint, ..., id = NULL, tag = "div")
```

Arguments

`endpoint`	API path the feature JS will fetch, e.g. `"api/sales/data"`. Resolved against the app base path by `aurora.url()` in the runtime.
`...`	Passed to the underlying **htmltools** tag. Named arguments become attributes (e.g. `class`, `style`, or extra `data-*` attributes such as `"data-page-size" = "25"`); unnamed arguments become child tags.
`id`	Element id, so your JS can find it (`document.getElementById`). Optional but recommended.
`tag`	HTML tag name to emit. Defaults to `"div"`.

Details

aurora deliberately ships no rendering JavaScript: this keeps the runtime thin and leaves charts, tables, and maps fully under the app's control. Use it to avoid hand-writing the `data-*` plumbing on every element.

Value

An **htmltools** tag.

Examples

```
aurora_component("api/sales/data", id = "vendas", style = "height:360px;")
```

`aurora_config` *Read the app's* `data/config.yml`*, anchored to the app root*

Description

Thin wrapper over `config::get()` that resolves `data/config.yml` relative to the **app directory** (an absolute path), instead of the **config** package's default search from the current working directory. This avoids the cwd pitfall where a helper or handler is evaluated with a working directory other than the app root and `config::get()` cannot find the file.

Usage

```
aurora_config(
  value = NULL,
  ...,
  dir = ".",
  file = NULL,
  config = Sys.getenv("R_CONFIG_ACTIVE", "default")
)
```

Arguments

`value`	Config value to read; `NULL` (default) returns the whole active configuration as a list.
`...`	Passed to `config::get()`.
`dir`	App directory (used to locate `data/config.yml`).
`file`	Explicit path to the config file; overrides `dir` when supplied.
`config`	Active configuration name; defaults to the `R_CONFIG_ACTIVE` environment variable or `"default"`.

Details

`data/config.yml` (app runtime config: DB credentials, environment profiles, service URLs) is intentionally separate from `_aurora.yml` (aurora wiring); see the project decision records. This helper just makes reading it robust.

Value

The requested config value (or the whole config list).

aurora_create_app *Scaffold a new aurora app*

Description

Creates the canonical aurora layout from a bundled template: `api.R`, `build_ui.R`, `helpers/`, `routers/`, `ui_modules/`, `www/` (with `style.css` and `js/app.js`), `data/config.yml`, and `.dockerignore`. The JS runtime is copied fresh into `www/js/core.js`. A first UI build is attempted so the app is immediately runnable.

Usage

```
aurora_create_app(
  path,
  template = c("minimal", "dashboard", "auth"),
  engine = "plumber2"
)
```

Arguments

`path`	Directory to create. Must not exist or must be empty.
`template`	Bundled template: `"minimal"` (bare app) or `"auth"` (JWT-cookie login gating `/api/*`). `"dashboard"` is planned.
`engine`	Web engine. Only `"plumber2"` is supported.

Value

The app path, invisibly.

aurora_data_get *Read a dataset from a store, reloading if the file changed*

Description

Returns the named dataset, re-reading from disk if its modification time has advanced since the last read (hot reload), otherwise returning the cached value.

Usage

```
aurora_data_get(store, name)
```

Arguments

`store`	An `aurora_data_store()`.
`name`	Registered dataset name.

Value

The dataset as returned by its reader.

---

aurora_data_names *Names of the datasets registered in a store*

Description

Names of the datasets registered in a store

Usage

```
aurora_data_names(store)
```

Arguments

store	An aurora_data_store().

Value

A character vector of dataset names.

---

aurora_data_register *Register a dataset in a data store*

Description

Register a dataset in a data store

Usage

```
aurora_data_register(store, name, path, reader = NULL)
```

Arguments

store	An aurora_data_store().
name	Dataset name used in aurora_data_get().
path	File path (resolved against the store's dir).
reader	Optional reader function function(path). If NULL, inferred from the file extension.

Value

The store, invisibly.

`aurora_data_store` *Create a hot-reloading data store*

Description

Registers named datasets backed by files on disk and hands them to route handlers without globals or `<<-`. Each `aurora_data_get()` checks the file's modification time and transparently re-reads it if an external process (e.g. an ETL job) has rewritten it – the stateless equivalent of the reference app's `carregar_bases()` mtime trick.

Usage

```
aurora_data_store(..., dir = ".", readers = list())
```

Arguments

`...`	Named file paths to register (e.g. `sales = "data/sales.rds"`). The reader is inferred from the file extension.
`dir`	Base directory that relative dataset paths are resolved against. Resolved to an absolute path **once, when the store is created** (not at read time), so later changes to the working directory cannot break reads. Defaults to `"."`; since the store is normally created while a `helpers/*.R` file is sourced (cwd = app root), relative paths like `"data/x.rds"` anchor to the app root. Absolute dataset paths are used as-is regardless of `dir`.
`readers`	Named list of reader functions keyed by lowercase file extension, merged over (and overriding) the built-ins (`rds`, `csv`, `parquet`).

Details

Define the store once in a `helpers/*.R` file (sourced before routers are parsed) and read from it in handlers:

```
# helpers/data.R
store <- aurora_data_store(sales = "data/sales.rds", dir = ".")

# routers/sales.R
#* @get /api/sales
#* @serializer json
function() aurora_data_get(store, "sales")
```

Value

An object of class `aurora_data_store`.

Examples

```
f <- tempfile(fileext = ".rds")
saveRDS(data.frame(x = 1:3), f)
store <- aurora_data_store(demo = f)
aurora_data_get(store, "demo")
```

`aurora_dockerfile` *Generate a Dockerfile (and .dockerignore) for an aurora app*

Description

Writes a Dockerfile that installs system deps, R packages (incl. **plumber2** and **aurora**), copies the app, and runs `api.R`. The image suits plain Docker or a ShinyProxy container. A `.dockerignore` is written if absent.

Usage

```
aurora_dockerfile(
  dir = ".",
  flavor = c("debian", "alpine"),
  base = NULL,
  sysdeps = "auto",
  port = 8000L,
  aurora_source = "aurora-govpe/aurora-rpkg@v0.1.12",
  tz = "America/Recife",
  locale = "pt_BR.UTF-8",
  write = TRUE
)
```

Arguments

`dir`	App directory.
`flavor`	Base-image flavor: `"debian"` (rocker + PPM binaries) or `"alpine"` (r-minimal + source builds).
`base`	Base Docker image. `NULL` (default) resolves per `flavor`: `rocker/r-ver:4.4.1` (debian) or `rhub/r-minimal` (alpine).
`sysdeps`	`"auto"` uses a curated default set of system packages for the flavor (covers the plumber2 + bslib baseline plus common TLS/curl/db/geo/ graphics needs); or pass a character vector to set them explicitly. For `"alpine"` this is the **build** (`-t`) set; runtime libs use a curated default. (`pkg_sysreqs()` auto-resolution proved unreliable, so aurora ships comprehensive defaults instead – extra `-dev` packages are build-time only.)
`port`	Port exposed and bound (via the `AURORA_PORT` env var).

aurora_source	pak/remotes spec used to install aurora in the image. Defaults to the pinned release "aurora-govpe/aurora-rpkg@v0.1.12" for reproducible builds. Avoid an unpinned moving ref (a branch): it interacts badly with Docker's layer cache, which can silently keep an old commit on rebuild. Bump this default (or pass an explicit @tag) per release.
tz	Timezone baked into the image as ENV TZ= (set before R starts, so it applies to system logs, R's date-times, and DB drivers). Defaults to "America/Recife"; pass NULL (or "") to omit the line. On "alpine" the image also gets tzdata and TZDIR so the zone actually resolves.
locale	Locale baked as ENV LANG=/LC_ALL=. Defaults to "pt_BR.UTF-8". "C.UTF-8" is UTF-8 and available everywhere (no generation needed). A specific locale like "pt_BR.UTF-8" needs the matching locale in the image: on "debian" (glibc) it is present; on "alpine" aurora adds the musl-locales/musl-locales-lang packages so it works – though musl's locale support is partial (charset and messages apply, but collation falls back to C).
write	Write the file (TRUE) or return its text (FALSE).

Details

Two flavors:

- "debian" (default) – rocker/r-ver; installs R packages as **binaries** from Posit Package Manager (fast builds; amd64). Best for heavy/geo apps and fast CI. Larger image.
- "alpine" – rhub/r-minimal; a tiny image (~25 MB base) that **compiles** every package from source via installr (no CRAN binaries; builds are slower). Best when image size matters or you need native arm64. Tune the Alpine build/runtime system deps via sysdeps if your packages need more.

Runtime R packages are detected by scanning routers/, helpers/, and api.R. To pin them explicitly (reproducible images), set a packages: list in _aurora.yml; plumber2 and aurora are always added.

Value

The Dockerfile path (if written) or its contents, invisibly.

---

aurora_geojson *Encode an sf object as GeoJSON for a JSON response, NULL-safe*

Description

Returns an unboxed GeoJSON string for an **sf** object, or NULL if the input is NULL or not an sf object (so an absent layer serializes as JSON null, not [] or an error).

Usage

```
aurora_geojson(x)
```

Arguments

x	An **sf** object, or NULL.

Value

An unboxed GeoJSON string, or NULL.

---

aurora_jwt_decode *Decode and verify a JWT*

Description

Returns the payload if the signature is valid and the token has not expired; otherwise NULL (never throws).

Usage

```
aurora_jwt_decode(auth, token)
```

Arguments

auth	An aurora_auth_jwt() scheme.
token	The token string (e.g. request$cookies$token).

Value

The decoded payload list, or NULL.

---

aurora_jwt_guard *Guard a request, aborting with 401 unless it carries a valid token*

Description

Reads the scheme's cookie from request, verifies it, and – if invalid or absent – stops handling with a 401 Unauthorized via reqres::abort_unauthorized(). On success it returns the payload invisibly. Use it inside a @header handler on /api/*, returning plumber2::Next to continue:

Usage

```
aurora_jwt_guard(auth, request)
```

Arguments

auth	An aurora_auth_jwt() scheme.
request	The reqres request object (the request handler argument).

Details

```
#* @any /api/*
#* @header
function(request) {
  aurora_jwt_guard(auth, request)
  plumber2::Next
}
```

Value

The decoded payload, invisibly (or aborts).

---

aurora_jwt_token *Mint a signed JWT for an auth scheme*

Description

Signs `claims` (plus an `exp` expiry computed from the scheme) with HMAC.

Usage

```
aurora_jwt_token(auth, claims = list())
```

Arguments

`auth`	An `aurora_auth_jwt()` scheme.
`claims`	A named list of claims to embed (e.g. `list(user = "alice")`).

Value

The signed token as a string.

---

aurora_run *Run an aurora app locally*

Description

Rebuilds the UI (optional), assembles the `aurora_app()` from convention, and starts the **plumber2** server. Development-time equivalent of `shiny::runApp()`. The generated `api.R` calls this function, so local dev and container entry share one assembly path.

Usage

```
aurora_run(
  dir = ".",
  port = 8000L,
  host = "127.0.0.1",
  rebuild_ui = NULL,
  watch = FALSE,
  watch_interval = 1,
  otel = NULL,
  verbose = NULL,
  attach = NULL,
  on_exit = NULL
)
```

Arguments

dir	App directory (canonical aurora layout).
port	Port to bind.
host	Host/interface to bind.
rebuild_ui	Whether to rebuild the static UI before running. NULL (default) resolves from the AURORA_REBUILD_UI environment variable (default TRUE locally). Containers set it to FALSE: the UI is compiled at build time and shipped as www/index.html, so the runtime image serves it without the UI build dependencies (bslib, and transitively shiny).
watch	Live-reload for development. When TRUE, aurora polls the UI source files (build_ui.R and ui_modules/) and rebuilds the static www/index.html on change – refresh the browser to see it. Changes to routers//helpers/ are detected but cannot be hot-swapped into a running server, so they log an advisory to restart. Requires the **later** package.
watch_interval	Polling interval in seconds when watch = TRUE.
otel	Enable OpenTelemetry logging. Passed to aurora_app(); NULL (default) resolves from _aurora.yml then the AURORA_OTEL env var.
verbose	Per-step **cli** logging. Passed to aurora_app(); NULL (default) resolves from options(aurora.verbose) then AURORA_VERBOSE.
attach	Attach the _aurora.yml packages: before sourcing helpers. Passed to aurora_app(); NULL (default) resolves from _aurora.yml (attach:) then AURORA_ATTACH. See ADR-012.
on_exit	Optional cleanup function function() run when the server stops (registered on plumber2's "cleanup" lifecycle event). Use it to release resources opened in a helper – e.g. pool::poolClose(con_base) – replacing the v1 pr_hook("exit", ...). Errors in the handler are reported but do not block shutdown.

Value

The result of plumber2::api_run(), invisibly.

`aurora_ruscker_yaml` *Emit a Ruscker app-spec block for an aurora image*

Description

Generates the YAML spec entry that goes under `proxy.specs` in a Ruscker configuration, pointing at a built aurora image (see `aurora_build_image()`). Ruscker is a reverse proxy and container orchestrator (a lightweight ShinyProxy alternative) that reads the ShinyProxy `application.yml` schema and adds its own fields. An aurora app is a *stateless* 'plumber2' API, so this emits a `type: api` spec – Ruscker load-balances a replica pool of the container instead of running one container per session (contrast `aurora_shinyproxy_yaml()`, which emits the interactive Docker-backed spec).

Usage

```
aurora_ruscker_yaml(
  image,
  dir = ".",
  id = NULL,
  display_name = NULL,
  description = NULL,
  port = 8000L,
  docs_path = "/__docs__",
  health_path = "/__healthz__",
  rate_limit = NULL,
  cors = NULL,
  min_replicas = 0L,
  max_replicas = 3L,
  env = NULL,
  wrap = FALSE,
  write = FALSE,
  file = NULL
)
```

Arguments

`image`	Container image tag, e.g. `"org/meu_app:latest"` (required).
`dir`	App directory, used only to default `id`/`display_name` from the app name.
`id`	Spec id. Defaults to the app name.
`display_name`	Human-facing name. Defaults to the app name.
`description`	Optional one-line description.
`port`	Port the app listens on inside the container, emitted as `api.port` (the aurora default is `8000`).
`docs_path`	OpenAPI/Swagger UI location, emitted as `api.docs-path`. Defaults to `"/__docs__"` (the 'plumber2'/aurora default).
`health_path`	Readiness-check endpoint, emitted as `api.health-path`. Defaults to `"/__healthz__"`.

rate_limit	Optional per-IP throttle, emitted as api.rate-limit, e.g. "100/min". Omitted when NULL.
cors	Optional logical; when not NULL, emitted as api.cors to toggle Ruscker's permissive CORS headers.
min_replicas	Always-running instances, emitted as min-replicas. Defaults to 0 (spawn on demand).
max_replicas	Auto-scale ceiling, emitted as max-replicas. Defaults to 3.
env	Optional named list/vector of container-env variables, e.g. list(AURORA_ENV = "prod").
wrap	If TRUE, wrap the entry under proxy: specs: so the output is a complete, paste-ready snippet. If FALSE (default), emit just the - id: ... list item to add under your existing proxy.specs.
write	If TRUE, also write the YAML to file.
file	Output path. Required when write = TRUE (there is no default path – pass an explicit location, e.g. one under tempdir()); NULL otherwise.

Value

The YAML block as a single string, invisibly.

See Also

aurora_shinyproxy_yaml() for the interactive ShinyProxy spec.

Examples

```
cat(aurora_ruscker_yaml(image = "org/meu_app:latest", id = "meu_app"))
```

---

aurora_set_auth_cookie

Set or clear the auth cookie on a response

Description

aurora_set_auth_cookie() writes an HttpOnly cookie carrying the token (at login); aurora_clear_auth_cookie() removes it (at logout). In production (HTTPS) pass secure = TRUE for Secure; SameSite=Strict; in development the default uses SameSite=Lax so it works over plain HTTP on a different port.

Usage

```
aurora_set_auth_cookie(auth, response, token, secure = FALSE)

aurora_clear_auth_cookie(auth, response, secure = FALSE)
```

Arguments

auth	An aurora_auth_jwt() scheme.
response	The reqres response object (the response handler argument).
token	The token string from aurora_jwt_token().
secure	Whether to set Secure + SameSite=Strict (use behind HTTPS).

Value

The response, invisibly.

---

aurora_shinyproxy_yaml

Emit a ShinyProxy app-spec block for an aurora image

Description

Generates the YAML spec entry that goes under proxy.specs in a ShinyProxy configuration, pointing at a built aurora image (see aurora_build_image()). aurora apps are ordinary Docker-backed apps from ShinyProxy's point of view: the spec just needs the image and the port the app listens on.

Usage

```
aurora_shinyproxy_yaml(
  image,
  dir = ".",
  id = NULL,
  display_name = NULL,
  description = NULL,
  port = 8000L,
  env = NULL,
  wrap = FALSE,
  write = FALSE,
  file = NULL
)
```

Arguments

image	Container image tag, e.g. "org/meu_app:latest" (required).
dir	App directory, used only to default id/display_name from the app name.
id	Spec id. Defaults to the app name.
display_name	Human-facing name. Defaults to the app name.
description	Optional one-line description.
port	Port the app listens on inside the container (the aurora default is 8000).

env	Optional named list/vector of container-env variables, e.g. list(AURORA_ENV = "prod").
wrap	If TRUE, wrap the entry under proxy: specs: so the output is a complete, paste-ready snippet. If FALSE (default), emit just the - id: ... list item to add under your existing proxy.specs.
write	If TRUE, also write the YAML to file.
file	Output path. Required when write = TRUE (there is no default path – pass an explicit location, e.g. one under tempdir()); NULL otherwise.

Value

The YAML block as a single string, invisibly.

Examples

```
cat(aurora_shinyproxy_yaml(image = "org/meu_app:latest", id = "meu_app"))
```

---

aurora_unbox *Unbox a scalar for a JSON response, NULL-safe*

Description

Wraps jsonlite::unbox() so a length-1 value serializes as a JSON scalar instead of a 1-element array, while NULL/empty input returns NULL (which serializes as JSON null or is dropped) instead of []. The [] case is a common footgun: a frontend doing if (x) or x[0] misbehaves on an empty array where it expected a scalar or null.

Usage

```
aurora_unbox(x)
```

Arguments

x	A length-1 value, or NULL/length-0.

Value

jsonlite::unbox(x) for a scalar, or NULL.

Examples

```
aurora_unbox("2026-06-02")
aurora_unbox(NULL)
```

aurora_unique *Sorted unique non-missing values, for filter options*

Description

Convenience for building dropdown/filter option lists from a column: sorted, unique, with NA dropped. Returns an empty character vector for NULL/empty input (serializes as [], the right shape for an empty list).

Usage

```
aurora_unique(x)
```

Arguments

x	A vector, or NULL.

Value

The sorted unique non-NA values.

Examples

```
aurora_unique(c("b", "a", NA, "a"))
```

Index