Package: tinyshinyserver (via r-universe)

November 24, 2026

Type Package

Title Tiny 'shiny' Server - Lightweight Multi-App 'shiny' Proxy

Version 0.3.0

Description A lightweight proxy server that hosts multiple 'shiny' applications, including interactive 'R Markdown' documents and 'Quarto' dashboards, each in its own R process. It forwards HTTP and 'WebSocket' traffic, keeps apps running or starts them on demand, restarts failed apps, and provides a web dashboard for monitoring and management. The server is configured with a JSON file.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports methods, shiny, callr (>= 3.7.0), ps (>= 1.7.0), jsonlite (>= 1.6), later (>= 1.4.0), curl (>= 3.3), promises, glue, httpuv (>= 1.5.5), websocket (>= 1.1.0), logger (>= 0.2.0), rmarkdown, quarto, tools, utils

Suggests testthat (>= 3.2.1), DT, dplyr, flexdashboard, plotly

Config/testthat/edition 3

URL https://github.com/lab1702/tinyshinyserver

BugReports https://github.com/lab1702/tinyshinyserver/issues

Config/roxygen2/version 8.1.1

Config/pak/sysreqs cmake make libuv1-dev libssl-dev zlib1g-dev

Repository https://lab1702.r-universe.dev

Date/Publication 2026-11-24 03:30:23 UTC

RemoteUrl https://github.com/lab1702/tinyshinyserver

RemoteRef HEAD

RemoteSha 0f0a76df232a992fea90d2b59a00d695d56f120a

Contents

tinyshinyserver-package 2
config-format 3
example-config 7
start_tss 8

Index **10**

`tinyshinyserver-package`

tinyshinyserver: Tiny Shiny Server - Lightweight Multi-App Shiny Proxy

---

Description

Serves several Shiny apps, interactive R Markdown documents, and Quarto dashboards from one R session. A lightweight proxy forwards HTTP and WebSocket traffic to the apps, each of which runs in its own R process.

Main functions

`start_tss` Start the Tiny Shiny Server with a configuration file

Key features

- One proxy for many apps, each in its own R process
- Resident apps, which start with the server and restart after a failure, and on-demand apps, which start when first opened
- Shiny apps, R Markdown documents with `runtime: shiny`, and Quarto documents with `server: shiny`
- A web dashboard and API to monitor apps and connections, restart apps, reload the configuration, and shut down
- Windows, Linux, and macOS support

Getting started

See `example-config` to copy and run the included apps and to learn what they require. Use `start_tss` to launch the server and `config-format` to configure your own apps. The README, at `https://github.com/lab1702/tinyshinyserver`, covers deployment behind a reverse proxy, the management API, and troubleshooting.

Author(s)

Maintainer: Lars Bernhardsson <cran.sherry228@passinbox.com>

Authors:

- Lars Bernhardsson <cran.sherry228@passinbox.com>

See Also

Useful links:

- https://github.com/lab1702/tinyshinyserver
- Report bugs at https://github.com/lab1702/tinyshinyserver/issues

---

`config-format` *Configuration File Format*

Description

Describes the JSON configuration file that `start_tss` reads.

Configuration Structure

The configuration file must be valid JSON with this structure. Only `apps`, `starting_port`, and `log_dir` are required; the other values shown are the defaults.

```
{
  "apps": [
    {
      "name": "app-name",
      "path": "./path/to/app",
      "resident": false,
      "appstart_timeout": 2
    }
  ],
  "starting_port": 3001,
  "log_dir": "./logs",
  "proxy_port": 3838,
  "proxy_host": "127.0.0.1",
  "management_port": 3839,
  "base_path": "",
  "title": "Tiny Shiny Server",
  "restart_delay": 5,
  "health_check_interval": 10,
  "max_request_size_mb": 100,
  "max_response_size_mb": 1024,
  "idle_timeout_minutes": 30,
  "max_unanswered_mb": 256,
  "log_level": "INFO"
}
```

Relative app paths and `log_dir` are resolved from the R working directory (`getwd()`), not from the configuration file's directory. JSON does not allow comments. Fields that the server does not recognize, such as a misspelled option name, have no effect; they are reported as a warning in `server.log` at startup and after a reload.

Required Fields

`apps` Array of apps to host. Each app must have a `name` and a `path`; see 'Application Configuration'.

`starting_port` First port to try when assigning ports to apps; a whole number from 1 to 65535. See 'Port Assignment'.

`log_dir` Directory for the server and app logs. The server writes `server.log`, and each app writes `<name>_output.log` and `<name>_error.log`; the previous run's logs are kept as `<name>_output.prev.log` and `<name>_error.prev.log`. For a Quarto app, the server also records its renders of the document in `<name>_render.rds`.

Optional Fields

`proxy_port` Port for the proxy, which serves the landing page and the apps (default: 3838).

`proxy_host` Network interface the proxy listens on (default: `"127.0.0.1"`); see 'Host Configuration'.

`management_port` Port for the management dashboard and API (default: 3839).

`base_path` Public URL prefix when a reverse proxy serves the landing page and apps under a path such as `"/shiny"` (default: `""`, the site root). It must start with `/` and be at most 100 characters long; a trailing `/` is removed. Path segments may contain only ASCII letters, digits, and `. _ ~ -`, may not be `.` or `..`, and the first segment may not be `proxy`, `api`, `templates`, or `health` in any letter case. The proxy accepts requests with or without the prefix, so the reverse proxy may either pass it through or strip it. Links, redirects, and cookie paths use the prefix.

`title` Name shown in the browser tab and top bar of the landing and management pages; a non-empty string of up to 100 characters (default: `"Tiny Shiny Server"`).

`restart_delay` Seconds to wait before restarting a failed app; a non-negative number of at most 86400, one day (default: 5). A resident app restarts then, and an on-demand app at the next request after that; an earlier request receives HTTP 503 with a `Retry-After` header. A restart requested from the management dashboard or API does not wait, and ends the wait of an on-demand app.

`health_check_interval` Seconds between health checks, which find app processes that have exited; a positive number of at most 86400 (default: 10).

`max_request_size_mb` Largest HTTP request body, in megabytes, that the proxy accepts; a positive finite number (default: 100). Larger bodies, and bodies sent with chunked transfer encoding, are rejected with HTTP 413 before they are read. Apps that raise `shiny.maxRequestSize` above this need a larger value.

`max_response_size_mb` Largest app response, in megabytes, that the proxy forwards; a positive finite number (default: 1024). The proxy stops receiving a larger response and answers HTTP 502.

`idle_timeout_minutes` Minutes without a WebSocket message, in either direction, after which a session is closed and the browser shows the app as disconnected; a non-negative finite number (default: 30). `0` keeps idle sessions open. Idle sessions are checked every 5 minutes.

`max_unanswered_mb` The most data, in megabytes, that an app may send a WebSocket session without a message from the browser in between; a non-negative finite number (default: 256). `0` sets no limit. Above it, the session is closed and the browser shows the app as disconnected; reloading the page reconnects it. The proxy cannot tell how much of what it sends a browser

has been read, so this limits the memory that one browser that stops reading, and sends nothing, can make the server use. Each message from the browser starts the count again, so a client that keeps sending messages while it does not read is not limited. A dashboard left open without interaction that receives more than this needs a larger value, or `0`.

`log_level` Least severe messages written to `server.log`: `"DEBUG"`, `"INFO"`, `"WARN"`, or `"ERROR"`, in any letter case (default: `"INFO"`). `"DEBUG"` also records the first 100 bytes of every WebSocket message, which can include what users type.

Application Configuration

Each application in the `apps` array can have:

`name` Unique identifier used in URLs and log file names: 1–50 ASCII letters, digits, underscores, or hyphens. Names must differ by more than letter case. Required.

`path` Path to the app directory, either absolute or relative to the R working directory; see 'Application Directories'. Required.

`resident` Boolean. If `true`, the app starts with the server and is restarted if its process fails. If `false` (the default), the app starts on demand and stops once it has had no WebSocket connection and no HTTP request in progress for 30 seconds. Restarting an on-demand app from the management dashboard or API stops it until it is next opened.

`appstart_timeout` Seconds, measured from app startup, that a request waits for the app to become ready before the proxy returns HTTP 503; a positive number of at most 86400 (default: 2). Fractional seconds are supported.

A field may be given only once, at the top level and within each app: a file that repeats one is invalid.

Application Directories

An app directory must hold one of these entry points, in order of preference: '`app.R`', or '`ui.R`' and '`server.R`'; an R Markdown document ('`.Rmd`') with a Shiny runtime; a Quarto document ('`.qmd`') with `server: shiny`; or, as `shiny::runApp()` accepts them, the Shiny file names in another letter case or '`server.R`' alone. A document whose name starts with _ does not count, because R Markdown and Quarto treat it as a partial that other documents include. An app whose directory is missing or holds no entry point is not started, and the reason is written to `server.log`. An R Markdown app serves only its entry document; a request for another '`.Rmd`' or '`.qmd`' document in its directory receives HTTP 404.

The app's R process starts in its directory and reads the '`.Renviron`' and '`.Rprofile`' there, if there are any, such as a profile that activates renv. As at R's startup, the files that `R_ENVIRON_USER` and `R_PROFILE_USER` name take precedence, a variable that is set but empty names no file (on Windows, which cannot hold an empty variable, it counts as not set), and a directory without these files falls back to the user's '`~/.Renviron`' and '`~/.Rprofile`'.

Two apps may not use the same directory if its document is rendered into that directory before it is served: a Quarto document, or an R Markdown document with `runtime: shinyrmd`, `runtime: shiny_prerendered`, or `server: shiny`. Two renders at once can leave a broken page, so give each app, and each server instance, its own copy of such a directory.

Host Configuration

The `proxy_host` field controls the proxy's listening interface. The management server and backend apps always listen on `127.0.0.1`. Supported proxy hosts are:

- `"127.0.0.1"` or `"localhost"`: Localhost only (most secure)
- `"::1"`: IPv6 localhost
- `"0.0.0.0"`: All IPv4 interfaces (allows external access)
- `"::"`: All IPv6 interfaces, and on most systems all IPv4 interfaces too

To protect against DNS rebinding, the management server rejects requests whose `Host` header is not `localhost`, `127.0.0.1`, or `[::1]`, and so does the proxy when `proxy_host` is a loopback address. A reverse proxy on the same machine must therefore forward the upstream address as the host and set `X-Forwarded-Host` to the public host. A reverse proxy on another machine, used with `proxy_host` set to `"0.0.0.0"` or `"::"`, must instead preserve the public `Host` header. App WebSocket connections are accepted only when the browser's `Origin` matches the public host.

App processes listen on their own loopback ports without these checks, so a DNS-rebinding page in a browser on the server machine can still reach a running app directly on its port.

Port Assignment

Apps are assigned ports in configuration order, starting from `starting_port` and skipping `proxy_port`, `management_port`, and ports already in use. For example, with `starting_port` set to 3001 and no conflicts, three apps receive ports 3001, 3002, and 3003. All ports must be whole numbers from 1 to 65535.

Reloading

The management dashboard's *Reload Config & Restart All* button reads the configuration file again, stops every app, and starts the resident apps under the new configuration; ports are assigned again at that point. The file is checked first: an invalid file, a `log_dir` where the server log cannot be written, apps that cannot all be given a port, or a change to `proxy_host`, `proxy_port`, or `management_port` is reported without stopping any app. Those three settings take effect only when the server restarts.

See Also

`start_tss` for starting the server with a configuration file, and `example-config` for a complete example configuration.

The README, at https://github.com/lab1702/tinyshinyserver, explains how apps start and stop, how to deploy behind a reverse proxy, including under a URL prefix, and the proxy's limits.

Examples

```
if (interactive()) {
  (function() {
    example_dir <- tempfile("tss-example-")
    dir.create(example_dir)
    old_dir <- setwd(example_dir)
```

```
    on.exit({
      setwd(old_dir)
      unlink(example_dir, recursive = TRUE)
    }, add = TRUE)
    examples_path <- system.file("examples", package = "tinyshinyserver")
    file.copy(examples_path, ".", recursive = TRUE)
    config_content <- '{
      "apps": [
        {"name": "sales", "path": "./examples/sales", "resident": true},
        {"name": "inventory", "path": "./examples/inventory", "resident": false}
      ],
      "starting_port": 3001,
      "proxy_port": 3838,
      "management_port": 3839,
      "log_dir": "./logs"
    }'
    writeLines(config_content, "my-config.json")
    start_tss(config = "my-config.json")
  })()
}
```

`example-config` *Example Configuration File*

Description

The package includes an example configuration file, '`config.json`', and the four apps that it serves. Use them to try the server, or as a starting point for your own configuration. See `config-format` for a description of each field.

Details

The example configuration uses the default proxy and management ports (3838 and 3839), assigns app ports from 3001, and serves these apps:

`sales` A Shiny app in one file, '`app.R`'. Resident: it starts with the server.

`inventory` A Shiny app in two files, '`ui.R`' and '`server.R`'. On demand: it starts when first opened.

`reports` An interactive R Markdown flexdashboard. On demand. Needs Pandoc and the **DT**, **plotly**, and **flexdashboard** packages.

`dashboard` A Quarto dashboard. Resident, so the server needs the Quarto CLI from the start. Needs the **DT**, **plotly**, and **dplyr** packages.

The app paths and the log directory are relative, so copy the '`examples`' directory and run the server from the directory that contains the copy, as in the example below. Locate the installed files with `system.file("examples", package = "tinyshinyserver")`.

See Also

- start_tss for starting the server
- config-format for configuration file format details

Examples

```
if (interactive()) {
  (function() {
    example_dir <- tempfile("tss-example-")
    dir.create(example_dir)
    old_dir <- setwd(example_dir)
    on.exit({
      setwd(old_dir)
      unlink(example_dir, recursive = TRUE)
    }, add = TRUE)
    examples_path <- system.file("examples", package = "tinyshinyserver")
    file.copy(examples_path, ".", recursive = TRUE)
    cat(readLines("examples/config.json"), sep = "\n")
    start_tss(config = "examples/config.json")
  })()
}
```

---

start_tss *Start Tiny Shiny Server*

Description

Starts the proxy, the management server, and the resident apps listed in a JSON configuration file, then serves requests until the server is shut down. On-demand apps start when they are first accessed.

Usage

```
start_tss(config = "config.json")
```

Arguments

config	Path to the JSON configuration file, as a character string. Defaults to 'config.json' in the current working directory. See config-format for the file format.

Details

See config-format for the required fields, defaults, app lifecycle settings, and port assignment. Relative app paths and the log directory are resolved from the R working directory, not from the configuration file's directory.

With the default ports, the server is reached at:

- Landing page: `http://localhost:3838`
- Management dashboard: `http://localhost:3839`
- Each app: `http://localhost:3838/proxy/{app_name}/`

Value

The internal server object (a reference class object of class `TinyShinyServer`), invisibly, after the server stops. The function is called for its side effect: it blocks the R session until it is interrupted (for example, with Ctrl-C) or shut down through the management dashboard or API. If an unexpected error stops the server, the function signals an error once the servers and apps have stopped, so that a script run with `Rscript` exits with a failure status. The returned object cannot be started again.

Examples

```
if (interactive()) {
  (function() {
    example_dir <- tempfile("tss-example-")
    dir.create(example_dir)
    old_dir <- setwd(example_dir)
    on.exit({
      setwd(old_dir)
      unlink(example_dir, recursive = TRUE)
    }, add = TRUE)
    examples_path <- system.file("examples", package = "tinyshinyserver")
    file.copy(examples_path, ".", recursive = TRUE)
    start_tss(config = "examples/config.json")
  })()
}
```

Index

∗ **package**
tinyshinyserver-package, 2

config-format, 3

example-config, 7

start_tss, *2*, *3*, *6*, 8, *8*

tinyshinyserver
(tinyshinyserver-package), 2
tinyshinyserver-package, 2