Package ‘microservices’

June 12, 2021

Type Package

Title Breakdown a Monolithic Application to a Suite of Services

URL https://github.com/tidylab/microservices

BugReports https://github.com/tidylab/microservices/issues

Version 0.1.2

Date 2021-05-18

Maintainer Harel Lustiger <tidylab@gmail.com>

Description 'Microservice' architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an 'HTTP' resource 'API'. These services are built around business capabilities and independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.1.1

Language en-GB

Depends R (>= 3.5)

Suggests testthat (>= 2.3.0), usethis (>= 1.3.0), httptest (>= 3.3.0), plumber (>= 1.0.0), pkgload, jsonlite, promises, future, httr

Imports config, desc, dplyr, glue, purrr, withr

Config/testthat/edition 3

NeedsCompilation no

Author Harel Lustiger [aut, cre] (<https://orcid.org/0000-0003-2953-9598>), Tidylab [cph, fnd]

Repository CRAN

Date/Publication 2021-06-12 06:10:02 UTC

R topics documented:

add_service 2
use_microservice 3

Index 7

---

add_service — *Add a Service Route to the Microservice*

Description

Expose additional set of services on a separate URL.

Usage

```
add_service(path = ".", name, overwrite = FALSE)
```

Arguments

`path`	(`character`) Where is the project root folder?
`name`	(`character`) what is the service route name? For example, if `name` = "repository" then the set of services would become available at http://127.0.0.1:8080/repository/.
`overwrite`	(`logical`) Should existing destination files be overwritten?

Details

Lay the infrastructure for an additional set of services. That includes adding a unit test, adding an endpoint, and extending the entrypointy.

Note: `add_service` adds a service to pre-existing plumber microservice which you could deploy by calling `use_microservice`.

How It Works:

Given a `path` (.) to a folder and a `name` (repository)
When `add_service` is called
Then the function creates the following files:

```
tests/testthat/test-endpoint-plumber-repository.R
inst/endpoints/plumber-repository.R
```

And updates the following files:

```
inst/entrypoints/plumber-foreground.R
```

When to Use:

In scenarios where services are thematically linked to each other. Examples for themes that should be mounted separately:

- ‘forecasting’ and ‘anomaly detection’
- ‘user’ and ‘business’

Value

No return value, called for side effects.

See Also

Other plumber microservice: `use_microservice()`

Examples

```
path <- tempfile()
dir.create(path, showWarnings = FALSE, recursive = TRUE)
use_microservice(path)

add_service(path, name = "repository")

list.files(path, recursive = TRUE)
```

`use_microservice` *Use a plumber Microservice in an R Project*

Description

Lay the infrastructure for a microservice. That includes unit test, dependency packages, configuration file, entrypoints and utility endpoint.

Usage

```
use_microservice(path = ".", overwrite = FALSE)
```

Arguments

`path`	(`character`) Where is the project root folder?
`overwrite`	(`logical`) Should existing destination files be overwritten?

Details

How It Works:

Given a `path` to a folder

When `use_microservice(path = ".")` is called

Then the function creates the following files:

```
tests/testthat/test-endpoint-plumber-utility.R
inst/configurations/plumber.yml
inst/endpoints/plumber-utility.R
inst/entrypoints/plumber-background.R
inst/entrypoints/plumber-foreground.R
```

And updates the following files:

`tests/testthat/helpers-xyz.R`

And adds the following packages to the DESCRIPTION file:

type	package	version
Suggests	config	*
Suggests	httptest	*
Suggests	httr	*
Imports	jsonlite	*
Suggests	pkgload	*
Suggests	plumber	>= 1.0.0
Imports	purrr	*
Suggests	testthat	*
Suggests	usethis	*
Suggests	promises	*
Suggests	future	*

When to Use `plumber`:

- A Single user/machine applications.
- Scheduled tasks. For example, you could use AirFlow with HTTP Operators to automate processes.

`plumber` *Advantages:*

- Comes with familiar way to document the microservice endpoint.
- Maturing package that comes with documentation, examples and support.

`plumber` *Disadvantages:*

- Runs on a single thread. That means that parallel algorithms such as random forest, can only be run on one core.
- Serves only one caller at a time.
- Can't make inward calls for other services, That means plumber can't be re-entrant. For example, if a microservice has three endpoints,`read_table`, `write_table`, and `orchestrator`, where the `orchestrator` reads a data table, transforms it, and writes it back, then the `orchestrator` can't make inwards calls via HTTP to `read_table` and `write_table`.

Note: While `plumber` is single-threaded by nature, it is possible to perform parallel execution using the `promises` package. See links under References.

Workflow:

1. Deploy the Microservice infrastructure

```
microservices::use_microservice(path = ".")
remotes::install_deps()
devtools::document()
```

1. Spin-up the microservice by running `source("./inst/entrypoints/plumber-background.R")`
2. Run the microservice unit-test by pressing Ctrl+Shift+T on Windows

Congratulations! You have added a microservice to your application and tested that it works.

References:

- Parallel execution in plumber
- `promises` package

Value

No return value, called for side effects.

See Also

Other plumber microservice: `add_service()`

Examples

```
path <- tempfile()
use_microservice(path)

list.files(path, recursive = TRUE)

cat(read.dcf(file.path(path, "DESCRIPTION"), "Imports"))
cat(read.dcf(file.path(path, "DESCRIPTION"), "Suggests"))
```

Index

∗ **plumber microservice**
    add_service, 2
    use_microservice, 3

add_service, 2, *6*

use_microservice, *3*, 3