Package ‘rsconnect’

May 24, 2021

Type Package
Title Deployment Interface for R Markdown Documents and Shiny Applications
Version 0.8.18
Description Programmatic deployment interface for 'RPubs', 'shinyapps.io', and 'RStudio Connect'. Supported content types include R Markdown documents, Shiny applications, Plumber APIs, plots, and static web content.
Depends R (>= 3.0.0)
Imports curl, digest, jsonlite, openssl, packrat (>= 0.6), rstudioapi (>= 0.5), yaml (>= 2.1.5)
Suggests RCurl, callr, httpuv, knitr, plumber (>= 0.3.2), reticulate, rmarkdown (>= 1.1), shiny, sourcetools, testthat, xtable
License GPL-2
RoxygenNote 7.1.1
URL https://github.com/rstudio/rsconnect
BugReports https://github.com/rstudio/rsconnect/issues
NeedsCompilation no
Author Jonathan McPherson [aut, cre], JJ Allaire [aut], RStudio [cph, fnd]
Maintainer Jonathan McPherson <jonathan@rstudio.com>
Repository CRAN
Date/Publication 2021-05-24 18:40:02 UTC

R topics documented:

rsconnect-package ......................................................... 3
accounts ........................................................................... 3
accountUsage .................................................................... 4
addAuthorizedUser ............................................................ 5
R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>addLinter</td>
<td>6</td>
</tr>
<tr>
<td>appDependencies</td>
<td>7</td>
</tr>
<tr>
<td>applications</td>
<td>8</td>
</tr>
<tr>
<td>authorizedUsers</td>
<td>9</td>
</tr>
<tr>
<td>configureApp</td>
<td>10</td>
</tr>
<tr>
<td>connectApiUser</td>
<td>11</td>
</tr>
<tr>
<td>connectUser</td>
<td>12</td>
</tr>
<tr>
<td>deployAPI</td>
<td>12</td>
</tr>
<tr>
<td>deployApp</td>
<td>13</td>
</tr>
<tr>
<td>deployDoc</td>
<td>15</td>
</tr>
<tr>
<td>deployments</td>
<td>16</td>
</tr>
<tr>
<td>deploySite</td>
<td>17</td>
</tr>
<tr>
<td>deployTFModel</td>
<td>19</td>
</tr>
<tr>
<td>forgetDeployment</td>
<td>20</td>
</tr>
<tr>
<td>generateAppName</td>
<td>21</td>
</tr>
<tr>
<td>lint</td>
<td>22</td>
</tr>
<tr>
<td>linter</td>
<td>22</td>
</tr>
<tr>
<td>listBundleFiles</td>
<td>23</td>
</tr>
<tr>
<td>makeLinterMessage</td>
<td>24</td>
</tr>
<tr>
<td>purgeApp</td>
<td>24</td>
</tr>
<tr>
<td>removeAuthorizedUser</td>
<td>25</td>
</tr>
<tr>
<td>restartApp</td>
<td>26</td>
</tr>
<tr>
<td>rpubsUpload</td>
<td>27</td>
</tr>
<tr>
<td>rsconnectOptions</td>
<td>28</td>
</tr>
<tr>
<td>rsconnectPackages</td>
<td>30</td>
</tr>
<tr>
<td>servers</td>
<td>31</td>
</tr>
<tr>
<td>setAccountInfo</td>
<td>32</td>
</tr>
<tr>
<td>setProperty</td>
<td>33</td>
</tr>
<tr>
<td>showInvited</td>
<td>34</td>
</tr>
<tr>
<td>showLogs</td>
<td>34</td>
</tr>
<tr>
<td>showMetrics</td>
<td>35</td>
</tr>
<tr>
<td>showProperties</td>
<td>36</td>
</tr>
<tr>
<td>showUsage</td>
<td>37</td>
</tr>
<tr>
<td>showUsers</td>
<td>38</td>
</tr>
<tr>
<td>syncAppMetadata</td>
<td>38</td>
</tr>
<tr>
<td>taskLog</td>
<td>39</td>
</tr>
<tr>
<td>tasks</td>
<td>39</td>
</tr>
<tr>
<td>terminateApp</td>
<td>40</td>
</tr>
<tr>
<td>unsetProperty</td>
<td>41</td>
</tr>
<tr>
<td>writeManifest</td>
<td>42</td>
</tr>
</tbody>
</table>
rsconnect-package

Deployment Interface for R Markdown Documents and Shiny Applications

Description

The ‘rsconnect“ package provides a programmatic deployment interface for RPubs, shinyapps.io, and RStudio Connect. Supported contents types include R Markdown documents, Shiny applications, plots, and static web content.

Managing Applications

Deploy and manage applications with the following functions:

- `deployApp()`: Deploy a Shiny application to a server.
- `configureApp()`: Configure an application currently running on a server.
- `restartApp()`: Restart an application currently running on a server.
- `terminateApp()`: Terminate an application currently running on a server.
- `deployments()`: List deployment records for a given application directory.

More information on application management is available in the `applications()` help page.

Managing Accounts and Users

Manage accounts on the local system.

- `setAccountInfo()`: Register an account.
- `removeAccount()`: Remove an account.
- `accountInfo()`: View information for a given account.

More information on account management is available in the `accounts()` help page.

accounts

Account Management Functions

Description

Functions to enumerate and remove accounts on the local system. Prior to deploying applications you need to register your account on the local system.

Usage

```r
accounts(server = NULL)
accountInfo(name, server = NULL)
removeAccount(name, server = NULL)
```
accountUsage

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>Name of the server on which the account is registered (optional; see servers())</td>
</tr>
<tr>
<td>name</td>
<td>Name of account</td>
</tr>
</tbody>
</table>

Details

You register an account using the setAccountInfo() function (for ShinyApps) or connectUser() function (for other servers). You can subsequently remove the account using the removeAccount function.

The accounts and accountInfo functions are provided for viewing previously registered accounts.

Value

accounts returns a data frame with the names of all accounts registered on the system and the servers on which they reside. accountInfo returns a list with account details.

See Also

Other Account functions: connectApiUser(), connectUser(), setAccountInfo()

---

accountUsage  | Show Account Usage

Description

Show account usage

Usage

accountUsage(
  account = NULL,
  server = NULL,
  usageType = "hours",
  from = NULL,
  until = NULL,
  interval = NULL
)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>Account name. If a single account is registered on the system then this parameter can be omitted.</td>
</tr>
<tr>
<td>server</td>
<td>Server name. Required only if you use the same account name on multiple servers.</td>
</tr>
<tr>
<td>usageType</td>
<td>Use metric to retrieve (for example: &quot;hours&quot;)</td>
</tr>
</tbody>
</table>
addAuthorizedUser

from  Date range starting timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
until Date range ending timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
interval Summarization interval. Data points at intervals less then this will be grouped. (Number of seconds or relative time delta e.g. "1h").

Note
This function only works for ShinyApps servers.

addAuthorizedUser  Add authorized user to application

Description
Add authorized user to application

Usage
addAuthorizedUser(
  email,
  appDir = getwd(),
  appName = NULL,
  account = NULL,
  server = NULL,
  sendEmail = NULL,
  emailMessage = NULL
)

Arguments
email  Email address of user to add.
appDir  Directory containing application. Defaults to current working directory.
appName  Name of application.
account  Account name. If a single account is registered on the system then this parameter can be omitted.
server  Server name. Required only if you use the same account name on multiple servers.
sendEmail  Send an email letting the user know the application has been shared with them.
emailMessage  Optional character vector of length 1 containing a custom message to send in email invitation. Defaults to NULL, which will use default invitation message.

Note
This function works only for ShinyApps servers.
addLinter

See Also
removeAuthorizedUser() and showUsers()

Description
Add a linter, to be used in subsequent calls to lint().

Usage
addLinter(name, linter)

Arguments
name
The name of the linter, as a string.

linter
A linter().

Examples
addLinter("no.capitals", linter(

  ## Identify lines containing capital letters -- either by name or by index
  apply = function(content, ...) {
    grep("[A-Z]", content)
  },

  ## Only use this linter on R files (paths ending with .r or .R)
  takes = function(paths) {
    grep("[rR]$", paths)
  },

  # Use the default message constructor
  message = function(content, lines, ...) {
    makeLinterMessage("Capital letters found on the following lines", content, lines)
  },

  # Give a suggested prescription
  suggest = "Do not use capital letters in these documents."
))
**appDependencies**

---

**Detect Application Dependencies**

**Description**

Recursively detect all package dependencies for an application. This function parses all .R files in the application directory to determine what packages the application depends on; and for each of those packages what other packages they depend on.

**Usage**

```r
appDependencies(appDir = getwd(), appFiles = NULL)
```

**Arguments**

- `appDir`: Directory containing application. Defaults to current working directory.
- `appFiles`: The files and directories to bundle and deploy (only if upload = TRUE). Can be NULL, in which case all the files in the directory containing the application are bundled. Takes precedence over appFileManifest if both are supplied.

**Details**

Dependencies are determined by parsing application source code and looking for calls to `library`, `require`, `::`, and `:::`.

Recursive dependencies are detected by examining the `Depends`, `Imports`, and `LinkingTo` fields of the packages immediately dependend on by the application.

**Value**

Returns a data frame listing the package dependencies detected for the application:

```
  package  version
  Name of package  Version of package
```

**Note**

Since the `Suggests` field is not included when determining recursive dependencies of packages, it’s possible that not every package required to run your application will be detected.

In this case, you can force a package to be included dependency by inserting call(s) to `require` within your source directory. This code need not actually execute, for example you could create a standalone file named dependencies.R with the following code:

```r
require(xts)
require(colorspace)
```
This will force the xts and colorspace packages to be installed along with the rest of your application when it is deployed.

See Also

rsconnectPackages(Using Packages with rsconnect)

Examples

```r
## Not run:

# dependencies for the app in the current working dir
appDependencies()

# dependencies for an app in another directory
appDependencies("~/projects/shiny/app1")

## End(Not run)
```

---

### applications

**List Deployed Applications**

**Description**

List all applications currently deployed for a given account.

**Usage**

```r
applications(account = NULL, server = NULL)
```

**Arguments**

- **account**: Account name. If a single account is registered on the system then this parameter can be omitted.
- **server**: Server name. Required only if you use the same account name on multiple servers.

**Value**

Returns a data frame with the following columns:

- **id**: Application unique id
- **name**: Name of application
- **url**: URL where application can be accessed
- **status**: Current status of application. Valid values are pending, deploying, running, terminating, and terminated
- **size**: Instance size (small, medium, large, etc.) (on ShinyApps.io)
- **instances**: Number of instances (on ShinyApps.io)
- **config_url**: URL where application can be configured
authorizedUsers

Note

To register an account you call the `setAccountInfo()` function.

See Also

`deployApp()`, `terminateApp()`

Other Deployment functions: `deployAPI()`, `deployApp()`, `deployDoc()`, `deploySite()`, `deployTFModel()`

Examples

```r
## Not run:

# list all applications for the default account
applications()

# list all applications for a specific account
applications("myaccount")

# view the list of applications in the data viewer
View(applications())

## End(Not run)
```

### Description

*(Deprecated)* List authorized users for an application

### Usage

```r
authorizedUsers(appDir = getwd())
```

### Arguments

- **appDir**
  - Directory containing application. Defaults to current working directory.
configureApp  Configure an Application

Description

Configure an application running on a remote server.

Usage

configureApp(
  appName,
  appDir = getwd(),
  account = NULL,
  server = NULL,
  redeploy = TRUE,
  size = NULL,
  instances = NULL,
  logLevel = c("normal", "quiet", "verbose")
)

Arguments

  appName    Name of application to configure
  appDir     Directory containing application. Defaults to current working directory.
  account    Account name. If a single account is registered on the system then this parameter can be omitted.
  server     Server name. Required only if you use the same account name on multiple servers (see servers())
  redeploy   Re-deploy application after its been configured.
  size       Configure application instance size
  instances  Configure number of application instances
  logLevel   One of "quiet", "normal" or "verbose": indicates how much logging to the console is to be performed. At "quiet" reports no information; at "verbose", a full diagnostic log is captured.

Note

This function works only for ShinyApps servers.

See Also

  applications().deployApp()
connectApiUser

Examples

```r
## Not run:
# set instance size for an application
configureApp("myapp", size="xlarge")

## End(Not run)
```

---

### connectApiUser

**Connect Api User Account**

**Description**

Connect a user account to the package using an API key for authentication so that it can be used to deploy and manage applications on behalf of the account.

**Usage**

```r
connектApiUser(account = NULL, server = NULL, apiKey = NULL, quiet = FALSE)
```

**Arguments**

- `account`: A name for the account to connect. Optional.
- `server`: The server to connect to. Optional if there is only one server registered.
- `apiKey`: The API key used to authenticate the user
- `quiet`: Whether or not to show messages and prompts while connecting the account.

**Details**

This function configures the user to connect using an apiKey in the http auth headers instead of a token. This is less secure but may be necessary when the client is behind a proxy or otherwise unable to authenticate using a token.

**See Also**

Other Account functions: `accounts()`, `connectUser()`, `setAccountInfo()`
connectUser  

**Connect User Account**

**Description**

Connect a user account to the package so that it can be used to deploy and manage applications on behalf of the account.

**Usage**

```r
connectUser(account = NULL, server = NULL, quiet = FALSE)
```

**Arguments**

- `account`: A name for the account to connect. Optional.
- `server`: The server to connect to. Optional if there is only one server registered.
- `quiet`: Whether or not to show messages and prompts while connecting the account.

**Details**

When this function is invoked, a web browser will be opened to a page on the target server where you will be prompted to enter your credentials. Upon successful authentication, your local installation of `rsconnect` and your server account will be paired, and you'll be able to deploy and manage applications using the package without further prompts for credentials.

**See Also**

Other Account functions: `accounts()`, `connectApiUser()`, `setAccountInfo()`

deployAPI  

**Deploy a Plumber API**

**Description**

Deploys an application consisting of plumber API routes. The given directory must contain a script returning a `plumb` object or a plumber API definition.

**Usage**

```r
deployAPI(api, ...)
```

**Arguments**

- `api`: Path to the API project directory. Must contain either `entrypoint.R` or `plumber.R`.
- `...`: Additional arguments to `deployApp()`.
deployApp

Details

Deploy a plumber API definition by either supplying a directory containing plumber.R (an API definition) or entrypoint.R that returns a plumb object created by plumber::plumb(). See the plumber documentation for more information.

See Also

Other Deployment functions: applications(), deployApp(), deployDoc(), deploySite(), deployTFModel()

deployApp

Deploy an Application

Description

Deploy a shiny application, an RMarkdown document, a plumber API, or HTML content to a server.

Usage

deployApp(appDir = getwd(),
appFiles = NULL,
appFileManifest = NULL,
appPrimaryDoc = NULL,
appSourceDoc = NULL,
appName = NULL,
appTitle = NULL,
appId = NULL,
contentCategory = NULL,
account = NULL,
server = NULL,
upload = TRUE,
recordDir = NULL,
launch.browser = getOption("rsconnect.launch.browser", interactive()),
logLevel = c("normal", "quiet", "verbose"),
lint = TRUE,
metadata = list(),
forceUpdate = getOption("rsconnect.force.update.apps", FALSE),
python = NULL,
on.failure = NULL,
forceGeneratePythonEnvironment = FALSE)

Arguments

appDir Directory containing application. Defaults to current working directory.
deployApp

appFiles The files and directories to bundle and deploy (only if upload = TRUE). Can be NULL, in which case all the files in the directory containing the application are bundled. Takes precedence over appFileManifest if both are supplied.

appFileManifest An alternate way to specify the files to be deployed; a file containing the names of the files, one per line, relative to the appDir.

appPrimaryDoc If the application contains more than one document, this parameter indicates the primary one, as a path relative to appDir. Can be NULL, in which case the primary document is inferred from the contents being deployed.

appSourceDoc If the application is composed of static files (e.g. HTML), this parameter indicates the source document, if any, as a fully qualified path. Deployment information returned by deployments() is associated with the source document.

appName Name of application (names must be unique within an account). Defaults to the base name of the specified appDir.

appTitle Free-form descriptive title of application. Optional; if supplied, will often be displayed in favor of the name. When deploying a new application, you may supply only the appTitle to receive an auto-generated appName.

appId If updating an application, the ID of the application being updated. Optional unless updating an app owned by another user.

collectionCategory Optional; the kind of content being deployed (e.g. "plot" or "site").

account Account to deploy application to. This parameter is only required for the initial deployment of an application when there are multiple accounts configured on the system (see accounts).

server Server name. Required only if you use the same account name on multiple servers.

upload If TRUE (the default) then the application is uploaded from the local system prior to deployment. If FALSE then it is re-deployed using the last version that was uploaded. FALSE is only supported on shinyapps.io; TRUE is required on RStudio Connect.

recordDir Directory where publish record is written. Can be NULL in which case record will be written to the location specified with appDir.

launch.browser If true, the system’s default web browser will be launched automatically after the app is started. Defaults to TRUE in interactive sessions only. If a function is passed, it will be called after the app is started, with the app URL as a parameter.

logLevel One of "quiet", "normal" or "verbose"; indicates how much logging to the console is to be performed. At "quiet" reports no information; at "verbose", a full diagnostic log is captured.

lint Lint the project before initiating deployment, to identify potentially problematic code?

metadata Additional metadata fields to save with the deployment record. These fields will be returned on subsequent calls to deployments().

forceUpdate If TRUE, update any previously-deployed app without asking. If FALSE, ask to update. If unset, defaults to the value of getOption("rsconnect.force.update.apps", FALSE).
deployDoc

Description

Deploys an application consisting of a single R Markdown document or other single file (such as an HTML or PDF document).
deployDoc(doc, ...)  

Arguments  

- `doc` Path to the document to deploy.  
- `...` Additional arguments to `deployApp()`. Do not supply `appDir`, `appFiles`, or `appPrimaryDoc`; these three parameters are automatically generated by `deployDoc` from the document.  

Details  

When deploying an R Markdown document, any files which are required to render and display the file must be deployed.  

This method discovers these additional files using `rmarkdown::find_external_resources()` from `rmarkdown`. If you find that the document is missing dependencies, either specify the dependencies explicitly in the document (the documentation for `rmarkdown::find_external_resources()` explains how to do this), or call `deployApp()` directly and specify your own file list in the `appFiles` parameter.  

See Also  

Other Deployment functions: `applications()`, `deployAPI()`, `deployApp()`, `deploySite()`, `deployTFModel()`
deploySite

Arguments

appPath The path to the content that was deployed, either a directory or an individual document.

nameFilter Return only deployments matching the given name (optional)

accountFilter Return only deployments matching the given account (optional)

serverFilter Return only deployments matching the given server (optional)

excludeOrphaned If TRUE (the default), return only deployments made by a currently registered account. Deployments made from accounts that are no longer registered (via e.g. removeAccount()) will not be returned.

Value

Returns a data frame with at least following columns:

- name Name of deployed application
- account Account owning deployed application
- bundleId Identifier of deployed application’s bundle
- url URL of deployed application
- when When the application was deployed (in seconds since the epoch)
- lastSyncTime When the application was last synced (in seconds since the epoch)
- deploymentFile Name of configuration file

If additional metadata has been saved with the deployment record using the metadata argument to deployApp(), the frame will include additional columns.

See Also

applications() to get a list of deployments from the server, and deployApp() to create a new deployment.

Examples

## Not run:

# Return all deployments of the ~/r/myapp directory made with the 'abc'
# account
deployments("~/r/myapp", accountFilter="abc")

## End(Not run)

---

deploySite Deploy a Website
deploySite

Description

Deploy an R Markdown website to a server.

Usage

deploySite(
    siteDir = getwd(),
    siteName = NULL,
    account = NULL,
    server = NULL,
    render = c("none", "local", "server"),
    launch.browser =getOption("rsconnect\.launch\.browser", interactive()),
    logLevel = c("normal", "quiet", "verbose"),
    lint = FALSE,
    metadata = list(),
    python = NULL,
    ...
)

Arguments

siteDir Directory containing website. Defaults to current working directory.

siteName Name for the site (names must be unique within an account). Defaults to the
base name of the specified siteDir, (or to a name provided by a custom site
 generation function).

account Account to deploy application to. This parameter is only required for the initial
deployment of an application when there are multiple accounts configured on
the system (see accounts).

server Server name. Required only if you use the same account name on multiple
 servers.

render Rendering behavior for site: "none" to upload a static version of the current
contents of the site directory; "local" to render the site locally then upload it;
"server" to render the site on the server. Note that for "none" and "local" R
scripts (.R) and markdown documents (.Rmd and .md) will not be uploaded to
the server.

launch.browser If true, the system’s default web browser will be launched automatically after
the app is started. Defaults to TRUE in interactive sessions only. If a function is
passed, it will be called after the app is started, with the app URL as a parameter.

logLevel One of "quiet", "normal" or "verbose"; indicates how much logging to the
console is to be performed. At "quiet" reports no information; at "verbose",
a full diagnostic log is captured.

lint Lint the project before initiating deployment, to identify potentially problematic
code?

metadata Additional metadata fields to save with the deployment record. These fields will
be returned on subsequent calls to deployments().
deployTFModel

Full path to a python binary for use by reticulate. Required if reticulate is a dependency of the app being deployed. If python = NULL, and RETICULATE_PYTHON is set in the environment, its value will be used. The specified python binary will be invoked to determine its version and to list the python packages installed in the environment.

Additional arguments to deployApp(). Do not supply appDir, appFiles, or appSourceDoc; these three parameters are automatically generated by deploySite.

See Also

Other Deployment functions: applications(), deployAPI(), deployApp(), deployDoc(), deployTFModel()

deployTFModel

Deploy a TensorFlow saved model

Description

Deploys a directory containing a TensorFlow saved model file.

Usage

deployTFModel(modelDir, ...)

Arguments

modelDir Path to the saved model directory. MUST contain saved_model.pb or saved_model.pbtxt

... Additional arguments to deployApp().

Details

Deploy a single TensorFlow saved model as a bundle. Should be passed a directory that contains the saved_model.pb or saved_model.pbtxt file, as well as any variables and assets necessary to load the model.

A saved model directory might look like this:

./1/
./1/saved_model.pb or ./1/saved_model.pbtxt
./1/variables/
./1/variables/variables.data-00000-of-00001
./1/variables/variables.index

For information on creating saved models, see the Keras method keras::export_savedmodel.keras.engine.training.Model() or the TensorFlow method tensorflow::export_savedmodel(). If using the TensorFlow package for R, the official TensorFlow guide for saving and restoring models may be useful.

References

https://www.tensorflow.org/guide/saved_model
forgetDeployment

 forgetApplication Deployment

Description

Forgets about an application deployment. This is useful if the application has been deleted on the server, or the local deployment information needs to be reset.

Usage

```r
forgetDeployment(
  appPath = getwd(),
  name = NULL,
  account = NULL,
  server = NULL,
  dryRun = FALSE,
  force = !interactive()
)
```

Arguments

- `appPath` The path to the content that was deployed, either a directory or an individual document.
- `name` The name of the content that was deployed (optional)
- `account` The name of the account to which the content was deployed (optional)
- `server` The name of the server to which the content was deployed (optional)
- `dryRun` Set to TRUE to preview the files/directories to be removed instead of actually removing them. Defaults to FALSE.
- `force` Set to TRUE to remove files and directories without prompting. Defaults to FALSE in interactive sessions.

Details

This method removes from disk the file containing deployment metadata. If "name", "account", and "server" are all NULL, then all of the deployments for the application are forgotten; otherwise, only the specified deployment is forgotten.

Value

NULL, invisibly.
generateAppName  Generate Application Name

Description

Generate a short name (identifier) for an application given an application title.

Usage

generateAppName(appTitle, appPath = NULL, account = NULL, unique = TRUE)

Arguments

appTitle A descriptive title for the application.
appPath The path to the application’s content, either a directory or an individual document. Optional.
account The account where the application will be deployed. Optional.
unique Whether to try to generate a unique name.

Details

This function modifies the title until it forms a suitable application name. Suitable application names are 3 - 64 characters long and contain only alphanumeric characters.

The function is intended to be used to find a name for a new application. If appPath and account are both specified, then the returned name will also be unique among locally known deployments of the directory (note that it is not guaranteed to be unique on the server). This behavior can be disabled by setting unique = FALSE.

Value

Returns a valid short name for the application.

Examples

## Not run:
# Generate a short name for a sample application
generateAppName("My Father's Country", "/fathers-country", "myacct")

## End(Not run)
**lint**

*Lint a Project*

**Description**

Takes the set of active linters (see *addLinter()*), and applies them to all files within a project.

**Usage**

```
lint(project, files = NULL, appPrimaryDoc = NULL)
```

**Arguments**

- **project**: Path to a project directory.
- **files**: Specific files to lint. Can be NULL, in which case all the files in the directory will be linted.
- **appPrimaryDoc**: The primary file in the project directory. Can be NULL, in which case it’s inferred (if possible) from the directory contents.

**linter**

*Create a Linter*

**Description**

Generate a linter, which can identify errors or problematic regions in a project.

**Usage**

```
linter(apply, takes, message, suggestion)
```

**Arguments**

- **apply**: Function that, given the content of a file, returns the indices at which problems were found.
- **takes**: Function that, given a set of paths, returns the subset of paths that this linter uses.
- **message**: Function that, given content and lines, returns an informative message for the user. Typically generated with *makelinterMessage()*.
- **suggestion**: String giving a prescribed fix for the linted problem.
Examples
addLinter("no.capitals", linter(
    ## Identify lines containing capital letters -- either by name or by index
    apply = function(content, ...) {
        grep("[A-Z]", content)
    },

    ## Only use this linter on R files (paths ending with .r or .R)
    takes = function(paths) {
        grep("[rR]$", paths)
    },

    # Use the default message constructor
    message = function(content, lines, ...) {
        makeLinterMessage("Capital letters found on the following lines", content, lines)
    },

    # Give a suggested prescription
    suggest = "Do not use capital letters in these documents."
))

listBundleFiles  List Files to be Bundled

Description
Given a directory containing an application, returns the names of the files to be bundled in the application.

Usage
listBundleFiles(appDir)

Arguments
appDir Directory containing the application.

Details
This function computes results similar to a recursive directory listing from list.files(), with the following constraints:

1. If the total size of the files exceeds the maximum bundle size, no more files are listed. The maximum bundle size is controlled by the rsconnect.max.bundle.size option.
2. If the total size number of files exceeds the maximum number to be bundled, no more files are listed. The maximum number of files in the bundle is controlled by the rsconnect.max.bundle.files option.
3. Certain files and folders that don’t need to be bundled, such as those containing internal version control and RStudio state, are excluded.
Value

Returns a list containing the following elements:

- contents: A list of the files to be bundled
- totalSize: The total size of the files

---

makeLinterMessage  
**Construct a Linter Message**

Description

Pretty-prints a linter message. Primarily used as a helper for constructing linter messages with `linter()`.

Usage

`makeLinterMessage(header, content, lines)`

Arguments

- header: A header message describing the linter.
- content: The content of the file that was linted.
- lines: The line numbers from `content` that contain lint.

---

purgeApp  
**Purge an Application**

Description

Purge a currently archived ShinyApps application.

Usage

`purgeApp(appName, account = NULL, server = NULL, quiet = FALSE)`

Arguments

- appName: Name of application to purge
- account: Account name. If a single account is registered on the system then this parameter can be omitted.
- server: Server name. Required only if you use the same account name on multiple servers (see `servers()`)
- quiet: Request that no status information be printed to the console during the termination.
removeAuthorizedUser

Note

This function only works for ShinyApps servers.

See Also

applications(), deployApp(), and restartApp()

Examples

## Not run:

# purge an application
purgeApp("myapp")

## End(Not run)

removeAuthorizedUser  Remove authorized user from an application

Description

Remove authorized user from an application

Usage

removeAuthorizedUser(
  user,
  appDir = getwd(),
  appName = NULL,
  account = NULL,
  server = NULL
)

Arguments

user        The user to remove. Can be id or email address.
appDir      Directory containing application. Defaults to current working directory.
appName     Name of application.
account     Account name. If a single account is registered on the system then this parameter can be omitted.
server      Server name. Required only if you use the same account name on multiple servers.

Note

This function works only for ShinyApps servers.
restartApp

Description

Restart an application currently running on a remote server.

Usage

restartApp(appName, account = NULL, server = NULL, quiet = FALSE)

Arguments

- **appName** Name of application to restart
- **account** Account name. If a single account is registered on the system then this parameter can be omitted.
- **server** Server name. Required only if you use the same account name on multiple servers (see servers())
- **quiet** Request that no status information be printed to the console during the operation.

Note

This function works only for ShinyApps servers.

See Also

- applications(), deployApp(), and terminateApp()

Examples

```r
## Not run:

# restart an application
restartApp("myapp")

## End(Not run)
```
rpubsUpload

Upload a file to RPubs

Description

This function publishes a file to rpubs.com. If the upload succeeds a list that includes an id and continueUrl is returned. A browser should be opened to the continueUrl to complete publishing of the document. If an error occurs then a diagnostic message is returned in the error element of the list.

Usage

```r
rpubsUpload(title, contentFile, originalDoc, id = NULL, properties = list())
```

Arguments

- `title`: The title of the document.
- `contentFile`: The path to the content file to upload.
- `originalDoc`: The document that was rendered to produce the contentFile. May be NULL if the document is not known.
- `id`: If this upload is an update of an existing document then the id parameter should specify the document id to update. Note that the id is provided as an element of the list returned by successful calls to rpubsUpload.
- `properties`: A named list containing additional document properties (RPubs doesn't currently expect any additional properties, this parameter is reserved for future use).

Value

A named list. If the upload was successful then the list contains a id element that can be used to subsequently update the document as well as a continueUrl element that provides a URL that a browser should be opened to in order to complete publishing of the document. If the upload fails then the list contains an error element which contains an explanation of the error that occurred.

Examples

```r
## Not run:
# upload a document
result <- rpubsUpload("My document title", "Document.html")
if (!is.null(result$continueUrl))
  browseURL(result$continueUrl)
else
  stop(result$error)

# update the same document with a new title
updateResult <- rpubsUpload("My updated title", "Document.html",
                           id = result$id)

## End(Not run)
```
**Description**

The **rsconnect** package supports several options that control the method used for http communications, the printing of diagnostic information for http requests, and the launching of an external browser after deployment.

**Details**

Supported global options include:

- **rsconnect.ca.bundle** Path to a custom bundle of Certificate Authority root certificates to use when connecting to servers via SSL. This option can also be specified in the environment variable `RCONNECT_CA_BUNDLE`. Leave undefined to use your system’s default certificate store.

- **rsconnect.check.certificate** Whether to check the SSL certificate when connecting to a remote host; defaults to TRUE. Setting to FALSE is insecure, but will allow you to connect to hosts using invalid certificates as a last resort.

- **rsconnect.http** Http implementation used for connections to the back-end service:
  - **libcurl** Secure https using the curl R package
  - **rcurl** Secure https using the Rcurl R package (deprecated)
  - **curl** Secure https using the curl system utility
  - **internal** Insecure http using raw sockets

  If no option is specified then libcurl is used by default.

- **rsconnect.http.trace** When TRUE, trace http calls (prints the method, path, and total milliseconds for each http request)

- **rsconnect.http.trace.json** When TRUE, trace JSON content (shows JSON payloads sent to and received from the server)

- **rsconnect.http.verbose** When TRUE, print verbose output for http connections (useful only for debugging SSL certificate or http connection problems)

- **rsconnect.rcurl.options** A named list of additional cURL options to use when using the RCurl HTTP implementation in R. Run `RCurl::curlOptions()` to see available options.

- **rsconnect.libcurl.options** A named list of additional cURL options to use when using the curl HTTP implementation in R. Run `curl::curl_options()` to see available options.

- **rsconnect.error.trace** When TRUE, print detailed stack traces for errors occurring during deployment.

- **rsconnect.launch.browser** When TRUE, automatically launch a browser to view applications after they are deployed

- **rsconnect.locale.cache** When FALSE, disable the detected locale cache (Windows only).

- **rsconnect.locale** Override the detected locale.
rsconnectOptions

rsconnect.max.bundle.size  The maximum size, in bytes, for deployed content. If not set, defaults to 3 GB.
rsconnect.max.bundle.files The maximum number of files to deploy. If not set, defaults to 10,000.
rsconnect.force.update.apps  When TRUE, bypasses the prompt to confirm whether you wish to update previously-deployed content.
rsconnect.pre.deploy  A function to run prior to deploying content; it receives as an argument the path to the content that’s about to be deployed.
rsconnect.post.deploy  A function to run after successfully deploying content; it receives as an argument the path to the content that was just deployed.
rsconnect.python.enabled  When TRUE, use the python executable specified by the RETICULATE_PYTHON environment variable and add a python section to the deployment manifest. By default, python is enabled when deploying to RStudio Connect and disabled when deploying to shinyapps.io.

When deploying content from the RStudio IDE, the rsconnect package’s deployment methods are executed in a vanilla R session that doesn’t execute startup scripts. This can make it challenging to ensure options are set properly prior to push-button deployment, so the rsconnect package has a parallel set of “startup” scripts it runs prior to deploying. The follow are run in order, if they exist, prior to deployment:

$R_HOME/etc/rsconnect.site  Like Rprofile.site; for site-wide pre-flight and options.
~/.rsconnect_profile  Like .Rprofile; for user-specific content.
$PROJECT/.rsconnect_profile  Like .Rprofile for projects; $PROJECT here refers to the root directory of the content being deployed.

Note that, unlike .Rprofile, these files don’t replace each other; all three will be run if they exist.

Examples

## Not run:

# use curl for http connections
options(rsconnect.http = "curl")

# trace http requests
options(rsconnect.http.trace = TRUE)

# print verbose output for http requests
options(rsconnect.http.verbose = TRUE)

# print JSON content
options(rsconnect.http.trace.json = TRUE)

# don’t automatically launch a browser after deployment
options(rsconnect.launch.browser = FALSE)

## End(Not run)
Description

Deployed applications can depend on any package available on CRAN as well as any package hosted in a public GitHub repository.

When an application is deployed it’s source code is scanned for dependencies using the `appDependencies()` function. The list of dependencies is sent to the server along with the application source code and these dependencies are then installed alongside the application.

Note that the Suggests dependencies of packages are not automatically included in the list of dependent packages. See the Note section of the documentation of the `appDependencies()` function for details on how to force packages to be included in the dependency list.

CRAN Packages

When satisfying CRAN package dependencies, the server will build the exact versions of packages that were installed on the system from which the application is deployed.

If a locally installed package was not obtained from CRAN (e.g. was installed from R-Forge) and as a result doesn’t have a version that matches a version previously published to CRAN then an error will occur. It’s therefore important that you run against packages installed directly from CRAN in your local configuration.

GitHub Packages

It’s also possible to depend on packages hosted in public GitHub repositories, so long as they are installed via the `devtools::install_github()` function from the devtools package.

This works because install_github records the exact Github commit that was installed locally, making it possible to download and install the same source code on the deployment server.

Note that in order for this to work correctly you need to install the very latest version of devtools from Github. You can do this as follows:

```r
library(devtools)
install_github("r-lib/devtools")
```

See Also

`appDependencies()`
**servers**

**Server Management Functions**

**Description**

Functions to manage the list of known servers to which *rsconnect* can deploy and manage applications.

**Usage**

```r
servers(local = FALSE)
```

```r
discoverServers(quiet = FALSE)
```

```r
addConnectServer(url, name = NULL, certificate = NULL, quiet = FALSE)
```

```r
addServer(url, name = NULL, certificate = NULL, quiet = FALSE)
```

```r
removeServer(name)
```

```r
serverInfo(name)
```

```r
addServerCertificate(name, certificate, quiet = FALSE)
```

**Arguments**

- `local` Return only local servers (i.e. not shinyapps.io)
- `quiet` Suppress output and prompts where possible.
- `name` Optional nickname for the server. If none is given, the nickname is inferred from the server’s hostname.
- `certificate` Optional; a path a certificate file to be used when making SSL connections to the server. The file’s contents are copied and stored by the *rsconnect* package. Can also be a character vector containing the certificate’s contents.

**Details**

Register a server with `addServer` or `discoverServers` (the latter is useful only if your administrator has configured server autodiscovery). Once a server is registered, you can connect to an account on the server using `connectUser()`.

The `servers` and `serverInfo` functions are provided for viewing previously registered servers.

There is always at least one server registered (the shinyapps.io server)

**Value**

`servers` returns a data frame with registered server names and URLs. `serverInfo` returns a list with details for a particular server.
Examples

```r
# register a local server
addServer("http://myrsconnect/", "myserver")

# list servers
servers(local = TRUE)

# connect to an account on the server
connectUser(server = "myserver")

## End(Not run)
```

---

**setAccountInfo**  
*Set ShinyApps Account Info*

**Description**

Configure a ShinyApps account for publishing from this system.

**Usage**

```r
setAccountInfo(name, token, secret)
```

**Arguments**

- `name`  
  Name of account to save or remove
- `token`  
  User token for the account
- `secret`  
  User secret for the account

**See Also**

Other Account functions: `accounts()`, `connectApiUser()`, `connectUser()`

**Examples**

```r
# register an account
setAccountInfo("user", "token", "secret")

# remove the same account
removeAccount("user")

## End(Not run)
```
setProperty

Set Application property

Description
Set a property on currently deployed ShinyApps application.

Usage
```
setProperty(
    propertyName,  # Name of property to set
    propertyValue,  # Value to set property to
    appPath = getwd(),  # Directory or file that was deployed. Defaults to current working directory.
    appName = NULL,  # Name of application
    account = NULL,  # Account name. If a single account is registered on the system then this parameter can be omitted.
    force = FALSE  # Forcibly set the property
)
```

Arguments
- `propertyName`: Name of property to set
- `propertyValue`: Value to set property to
- `appPath`: Directory or file that was deployed. Defaults to current working directory.
- `appName`: Name of application
- `account`: Account name. If a single account is registered on the system then this parameter can be omitted.
- `force`: Forcibly set the property

Note
This function only works for ShinyApps servers.

Examples
```
## Not run:
# set instance size for an application
setProperty("application.instances.count", 1)

# disable application package cache
setProperty("application.package.cache", FALSE)
```

## End(Not run)
showInvited

List invited users for an application

Description
List invited users for an application

Usage
showInvited(appDir = getwd(), appName = NULL, account = NULL, server = NULL)

Arguments
appDir Directory containing application. Defaults to current working directory.
appName Name of application.
account Account name. If a single account is registered on the system then this parameter can be omitted.
server Server name. Required only if you use the same account name on multiple servers.

Note
This function works only for ShinyApps servers.

See Also
addAuthorizedUser() and showUsers()

showLogs

Show Application Logs

Description
Show the logs for a deployed ShinyApps application.

Usage
showLogs(
  appPath = getwd(),
  appFile = NULL,
  appName = NULL,
  account = NULL,
  entries = 50,
  streaming = FALSE
  )
showMetrics

Arguments

**appPath**  
The path to the directory or file that was deployed.

**appFile**  
The path to the R source file that contains the application (for single file applications).

**appName**  
The name of the application to show logs for. May be omitted if only one application deployment was made from `appPath`.

**account**  
The account under which the application was deployed. May be omitted if only one account is registered on the system.

**entries**  
The number of log entries to show. Defaults to 50 entries.

**streaming**  
Whether to stream the logs. If `TRUE`, then the function does not return; instead, log entries are written to the console as they are made, until R is interrupted. Defaults to `FALSE`.

Note  
This function only uses the `libcurl` transport, and works only for ShinyApps servers.

---

**showMetrics**  
*Show Application Metrics*

Description

Show application metrics of a currently deployed application

Usage

```r
showMetrics(
  metricSeries,  
  metricNames,  
  appDir = getwd(),  
  appName = NULL,  
  account = NULL,  
  server = NULL,  
  from = NULL,  
  until = NULL,  
  interval = NULL  
)
```

Arguments

**metricSeries**  
Metric series to query. Refer to the `shinyapps.io documentation` for available series.

**metricNames**  
Metric names in the series to query. Refer to the `shinyapps.io documentation` for available metrics.

**appDir**  
Directory containing application. Defaults to current working directory.
showProperties

appName  Name of application
account  Account name. If a single account is registered on the system then this parameter can be omitted.
server  Server name. Required only if you use the same account name on multiple servers.
from  Date range starting timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
until  Date range ending timestamp (Unix timestamp or relative time delta such as "2d" or "3w").
interval  Summarization interval. Data points at intervals less then this will be grouped. (Relative time delta e.g. "120s" or "1h" or "30d").

Note

This function only works for ShinyApps servers.

showProperties  Show Application property

Description

Show properties of an application deployed to ShinyApps.

Usage

showProperties(appPath = getwd(), appName = NULL, account = NULL)

Arguments

appPath  Directory or file that was deployed. Defaults to current working directory.
appNam  Name of application
account  Account name. If a single account is registered on the system then this parameter can be omitted.

Note

This function works only for ShinyApps servers.
showUsage

Description

Show application usage of a currently deployed application

Usage

showUsage(
    appDir = getwd(),
    appName = NULL,
    account = NULL,
    server = NULL,
    usageType = "hours",
    from = NULL,
    until = NULL,
    interval = NULL
)

Arguments

appDir    Directory containing application. Defaults to current working directory.
appName   Name of application
account   Account name. If a single account is registered on the system then this parameter
can be omitted.
server    Server name. Required only if you use the same account name on multiple
          servers.
usagetype  Use metric to retrieve (for example: "hours")
from      Date range starting timestamp (Unix timestamp or relative time delta such as
          "2d" or "3w").
until     Date range ending timestamp (Unix timestamp or relative time delta such as
          "2d" or "3w").
interval  Summarization interval. Data points at intervals less than this will be grouped.
          (Relative time delta e.g. "120s" or "1h" or "30d").

Note

This function only works for ShinyApps servers.
showUsers

List authorized users for an application

Description

List authorized users for an application

Usage

showUsers(appDir = getwd(), appName = NULL, account = NULL, server = NULL)

Arguments

appDir
Directory containing application. Defaults to current working directory.

appName
Name of application.

account
Account name. If a single account is registered on the system then this parameter can be omitted.

server
Server name. Required only if you use the same account name on multiple servers.

Note

This function works only for ShinyApps servers.

See Also

addAuthorizedUser() and showInvited()

csyncAppMetadata

Sync Application Metadata

Description

Update the metadata for requested application across all deployments

Usage

csyncAppMetadata(appPath)

Arguments

appPath
The path to the directory or file that was deployed.

Note

This function does not update metadata for Shiny and rpubs apps
taskLog

Show task log

Description

 Writes the task log for the given task

Usage

 taskLog(taskId, account = NULL, server = NULL, output = NULL)

Arguments

 taskId Task Id
 account Account name. If a single account is registered on the system then this parameter can be omitted.
 server Server name. Required only if you use the same account name on multiple servers (see servers())
 output Where to write output. Valid values are NULL or stderr

See Also

 tasks()

Examples

 ## Not run:

 # write task log to stdout
 taskLog(12345)

 # write task log to stderr
 taskLog(12345, output="stderr")

 ## End(Not run)

tasks

List Tasks

Description

 List Tasks

Usage

 tasks(account = NULL, server = NULL)
**terminateApp**

Terminate an Application

**Description**

Terminate and archive a currently deployed ShinyApps application.

**Usage**

```r
terminateApp(appName, account = NULL, server = NULL, quiet = FALSE)
```

**Arguments**

- **appName**
  - Name of application to terminate

- **account**
  - Account name. If a single account is registered on the system then this parameter can be omitted.

- **server**
  - Server name. Required only if you use the same account name on multiple servers (see `servers()`)

**Value**

Returns a data frame with the following columns:

<table>
<thead>
<tr>
<th>id</th>
<th>Task id</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Task action</td>
</tr>
<tr>
<td>status</td>
<td>Current task status</td>
</tr>
<tr>
<td>created_time</td>
<td>Task creation time</td>
</tr>
<tr>
<td>finished_time</td>
<td>Task finished time</td>
</tr>
</tbody>
</table>

**See Also**

`taskLog()`

**Examples**

```r
## Not run:

# list tasks for the default account
tasks()

## End(Not run)
```
**server**  
Server name. Required only if you use the same account name on multiple servers (see `servers()`)

**quiet**  
Request that no status information be printed to the console during the termination.

**Note**  
This function only works for ShinyApps servers.

**See Also**  
`applications()`, `deployApp()`, and `restartApp()`

**Examples**  
```r
## Not run:

# terminate an application
terminateApp("myapp")

## End(Not run)
```

---

**unsetProperty**  
Unset Application property

**Description**  
Unset a property on currently deployed ShinyApps application (restoring to its default value)

**Usage**  
```r
unsetProperty(
  propertyName,
  appPath = getwd(),
  appName = NULL,
  account = NULL,
  force = FALSE
)
```

**Arguments**  
- `propertyName`: Name of property to unset
- `appPath`: Directory or file that was deployed. Defaults to current working directory.
- `appName`: Name of application
- `account`: Account name. If a single account is registered on the system then this parameter can be omitted.
- `force`: Forcibly unset the property
writeManifest

Note

This function only works for ShinyApps servers.

Examples

```r
## Not run:

# unset application package cache property to revert to default
unsetProperty("application.package.cache")

## End(Not run)
```

writeManifest

Create a manifest.json describing deployment requirements.

Description

Given a directory content targeted for deployment, write a manifest.json into that directory describing the deployment requirements for that content.

Usage

```r
writeManifest(
  appDir = getwd(),
  appFiles = NULL,
  appPrimaryDoc = NULL,
  contentCategory = NULL,
  python = NULL,
  forceGeneratePythonEnvironment = FALSE,
  verbose = FALSE
)
```

Arguments

- `appDir`  | Directory containing the content (Shiny application, R Markdown document, etc).
- `appFiles` | Optional. The full set of files and directories to be included in future deployments of this content. Used when computing dependency requirements. When NULL, all files in `appDir` are considered.
- `appPrimaryDoc` | Optional. Specifies the primary document in a content directory containing more than one. If NULL, the primary document is inferred from the file list.
- `contentCategory` | Optional. Specifies the kind of content being deployed (e.g. "plot" or "site").
**python**

Full path to a python binary for use by reticulate. The specified python binary will be invoked to determine its version and to list the python packages installed in the environment. If python = NULL, and RETICULATE_PYTHON is set in the environment, its value will be used.

**forceGeneratePythonEnvironment**

Optional. If an existing requirements.txt file is found, it will be overwritten when this argument is TRUE.

**verbose**

If TRUE, prints progress messages to the console
Index

* **Account functions**
  - accounts, 3
  - connectApiUser, 11
  - connectUser, 12
  - setAccountInfo, 32

* **Deployment functions**
  - applications, 8
  - deployAPI, 12
  - deployApp, 13
  - deployDoc, 15
  - deploySite, 17
  - deployTFModel, 19

* **package**
  - rsconnect-package, 3

  accountInfo (accounts), 3
  accountInfo(), 3
  accounts, 3, 11, 12, 14, 18, 32
  accounts(), 3
  accountUsage, 4
  addAuthorizedUser, 5
  addAuthorizedUser(), 26, 34, 38
  addConnectServer (servers), 31
  addLinter, 6
  addLinter(), 22
  addServer (servers), 31
  addServerCertificate (servers), 31
  appDependencies, 7
  appDependencies(), 30
  applications, 8, 13, 15, 16, 19, 20
  applications(), 3, 10, 15, 17, 25, 26, 41
  authorizedUsers, 9

  configureApp, 10
  configureApp(), 3
  connectApiUser, 4, 11, 12, 32
  connectUser, 4, 11, 12, 32
  connectUser(), 4, 31

  deployAPI, 9, 12, 15, 16, 19, 20

  deployApp, 9, 13, 16, 19, 20
  deployApp(), 3, 9, 10, 12, 16, 17, 19, 25, 26, 41
  deployDoc, 9, 13, 15, 19, 20
  deployments, 16
  deployments(), 3, 14, 18
  deploySite, 9, 13, 15, 16, 17, 20
  deployTFModel, 9, 13, 15, 16, 19, 19
  devtools::install_github(), 30
  discoverServers (servers), 31

  forgetDeployment, 20

  generateAppName, 21

  keras::export_savedmodel.keras.engine.training.Model(), 19

  lint, 22
  lint(), 6
  linter, 22
  linter(), 6, 24
  list.files(), 23
  listBundleFiles, 23

  makeLinterMessage, 24
  makeLinterMessage(), 22

  purgeApp, 24

  removeAccount (accounts), 3
  removeAccount(), 3, 17
  removeAuthorizedUser, 25
  removeAuthorizedUser(), 6
  removeServer (servers), 31
  restartApp, 26
  restartApp(), 3, 15, 25, 41
  RMarkdown, 13
  rmarkdown::find_external_resources(), 16
  rpubsUpload, 27
INDEX

rsconnect (rsconnect-package), 3
rsconnect-package, 3
rsconnectOptions, 28
rsconnectPackages, 8, 30

serverInfo (servers), 31
servers, 31
servers(), 4, 10, 24, 26, 39–41
setAccountInfo, 4, 11, 12, 32
setAccountInfo(), 3, 4, 9
setProperty, 33
shiny, 13
showInvited, 34
showInvited(), 38
showLogs, 34
showMetrics, 35
showProperties, 36
showUsage, 37
showUsers, 38
showUsers(), 6, 26, 34
syncAppMetadata, 38

taskLog, 39
taskLog(), 40
tasks, 39
tasks(), 39
tensorflow::export_savedmodel(), 19
terminateApp, 40
terminateApp(), 3, 9, 15, 26

unsetProperty, 41

writeManifest, 42