Package 'rsconnect'

June 26, 2025

Type Package

Title Deploy Docs, Apps, and APIs to 'Posit Connect', 'shinyapps.io', and 'RPubs'

Version 1.5.0

Description Programmatic deployment interface for 'RPubs', 'shinyapps.io', and 'Posit Connect'. Supported content types include R Markdown documents, Shiny applications, Plumber APIs, plots, and static web content.

License GPL-2

URL https://rstudio.github.io/rsconnect/, https://github.com/rstudio/rsconnect

BugReports https://github.com/rstudio/rsconnect/issues

Depends R (>= 3.5.0)

Imports cli, curl, digest, jsonlite, lifecycle, openssl (>= 2.0.0), PKI, packrat (>= 0.6), renv (>= 1.0.0), rlang (>= 1.0.0), rstudioapi (>= 0.5), snowflakeauth, tools, yaml (>= 2.1.5), utils

Suggests Biobase, BiocManager, foreign, knitr, MASS, plumber (>= 0.3.2), quarto, RCurl, reticulate, rmarkdown (>= 1.1), shiny, testthat (>= 3.1.9), webfakes, withr

VignetteBuilder knitr, rmarkdown

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Aron Atkins [aut, cre], Toph Allen [aut], Hadley Wickham [aut],

Jonathan McPherson [aut],
JJ Allaire [aut],
Posit Software, PBC [cph, fnd]

Maintainer Aron Atkins <aron@posit.co>

Repository CRAN

Date/Publication 2025-06-26 15:50:02 UTC

Contents

`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

```
accounts(server = NULL)

accountInfo(name = NULL, server = NULL)

removeAccount(name = NULL, server = NULL)
```

Arguments

`server`	Name of the server on which the account is registered (optional; see `servers()`)
`name`	Name of account

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()`, `setAccountInfo()`

accountUsage *Show Account Usage*

Description

Show account usage

Usage

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

Arguments

account, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.
usageType	Use metric to retreive (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 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, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.
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.

See Also

removeAuthorizedUser() and showUsers()

---

addLinter *Add a Linter*

Description

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

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

Usage

```
addLinter(name, linter)
```


```
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."
))
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."
))
```

---

addServer *Server management*

Description

These functions manage the list of known servers:

- `addServer()` registers a Posit connect server. Once it has been registered, you can connect to an account on the server using `connectUser()`.
- `removeServer()` removes a server from the registry.
- `addServerCertificate()` adds a certificate to a server.

Usage

```
addServer(
  url,
  name = NULL,
  certificate = NULL,
  validate = TRUE,
  snowflakeConnectionName = NULL,
  quiet = FALSE
)

removeServer(name = NULL)

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

Arguments

`url`	URL for the server. Can be a bare hostname like `connect.mycompany.com` or a url like `http://posit.mycompany.com/connect`.
`name`	Server name. If omitted, the server hostname is used.
`certificate`	Optional. Either a path to certificate file or a character vector containing the certificate's contents.
`validate`	Validate that `url` actually points to a Posit Connect server?
`snowflakeConnectionName`	Name for the Snowflake connection parameters stored in `connections.toml`.
`quiet`	Suppress output and prompts where possible.

Examples

```
## Not run:
# 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)
```

`appDependencies` *Detect application dependencies*

Description

`appDependencies()` recursively detects all R package dependencies for an application by parsing all `.R` and `.Rmd` files and looking for calls to `library()`, `require()`, `requireNamespace()`, `::`, and so on. It then adds implicit dependencies (i.e. an `.Rmd` requires Rmarkdown) and adds all recursive dependencies to create a complete manifest of package packages need to be installed to run the app.

Usage

```
appDependencies(
  appDir = getwd(),
  appFiles = NULL,
  appFileManifest = NULL,
  appMode = NULL
)
```

Arguments

`appDir`	A directory containing an application (e.g. a Shiny app or plumber API). Defaults to the current directory.
`appFiles`, `appFileManifest`	Use `appFiles` to specify a character vector of files to bundle in the app or `appFileManifest` to provide a path to a file containing a list of such files. If neither are supplied, will bundle all files in `appDir`, apart from standard exclusions and files listed in a `.rscignore` file. See `listDeploymentFiles()` for more details.
`appMode`	Optional; the type of content being deployed. Provide this option when the inferred type of content is incorrect. This can happen, for example, when static HTML content includes a downloadable Shiny application `app.R`. Accepted values include `"shiny"`, `"api"`, `"rmd-static"`, `"rmd-shiny"`, `"quarto-static"`, `"quarto-shiny"`, and `"static"`. The Posit Connect API Reference contains a full set of available values. Not all servers support all types of content.

Value

A data frame with one row for each dependency (direct, indirect, and inferred), and 4 columns:

- `Package`: package name.
- `Version`: local version.
- `Source`: a short string describing the source of the package install, as described above.
- `Repository`: for CRAN and CRAN-like repositories, the URL to the repository. This will be ignored by the server if it has been configured with its own repository name -> repository URL mapping.

Dependency discovery

rsconnect use one of three mechanisms to find which packages your application uses:

1. If `renv.lock` is present, it will use the versions and sources defined in that file. If you're using the lockfile for some other purpose and don't want it to affect deployment, add `renv.lock` to `.rscignore`.
2. Otherwise, rsconnect will call `renv::snapshot()` to find all packages used by your code. If you'd instead prefer to only use the packages declared in a `DESCRIPTION` file, run `renv::settings$snapshot.type("e` to activate renv's "explicit" mode.
3. Dependency resolution using renv is a new feature in rsconnect 1.0.0, and while we have done our best to test it, it still might fail for your app. If this happens, please file an issue then set `options(rsconnect.packrat = TRUE)` to revert to the old dependency discovery mechanism.

Remote installation

When deployed, the app must first install all of these packages, and rsconnect ensures the versions used on the server will match the versions you used locally. It knows how to install packages from the following sources:

- CRAN and BioConductor (`Source: CRAN` or `Source: Bioconductor`). The remote server will ignore the specific CRAN or Bioconductor mirror that you use locally, always using the CRAN/BioC mirror that has been configured on the server.
- Other CRAN like and CRAN-like repositories. These packages will have a `Source` determined by the value of `getOptions("repos")`. For example, if you've set the following options:

  ```
  options(
     repos = c(
       CRAN = "https://cran.rstudio.com/",
       CORPORATE = "https://corporate-packages.development.company.com"
     )
  )
  ```

  Then packages installed from your corporate package repository will have source `CORPORATE`. Posit Connect can be configured to override their repository url so that (e.g.) you can use different packages versions on staging and production servers.
- Packages installed from GitHub, GitLab, or BitBucket, have `Source` `github`, `gitlab`, and `bitbucket` respectively. When deployed, the bundle contains the additional metadata needed to precisely recreated the installed version.

It's not possible to recreate the packages that you have built and installed from a directory on your local computer. This will have `Source: NA` and will cause the deployment to error. To resolve this issue, you'll need to install from one of the known sources described above.

Suggested packages

The `Suggests` field is not included when determining recursive dependencies, so it's possible that not every package required to run your application will be detected.

For example, ggplot2's `geom_hex()` requires the hexbin package to be installed, but it is only suggested by ggplot2. So if your app uses `geom_hex()` it will fail, reporting that the hexbin package is not installed.

You can overcome this problem with (e.g.) `requireNamespace(hexbin)`. This will tell rsconnect that your app needs the hexbin package, without otherwise affecting your code.

See Also

rsconnectPackages(Using Packages with rsconnect)

Examples

```
## 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

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

Arguments

`account`, `server`	Uniquely identify a remote server with either your user `account`, the `server` name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use `accounts()` to see the full list of available options.

Value

Returns a data frame with the following columns:

`id`	Application unique id
`name`	Name of application
`title`	Application title
`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

Note

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

See Also

deployApp(), terminateApp()

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

Examples

```
## 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)
```

---

authorizedUsers *(Deprecated) List authorized users for an application*

Description

(Deprecated) List authorized users for an application

Usage

```
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, server`	Uniquely identify a remote server with either your user `account`, the `server` name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use `accounts()` to see the full list of available options.
`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()`

Examples

```
## Not run:

# set instance size for an application
configureApp("myapp", size="xlarge")

## End(Not run)
```

`connectApiUser` *Register account on Posit Connect*

Description

`connectUser()` and `connectApiUser()` connect your Posit Connect account to the rsconnect package so that it can deploy and manage applications on your behalf.

`connectUser()` is the easiest place to start because it allows you to authenticate in-browser to your Posit Connect server. `connectApiUser()` is appropriate for non-interactive settings; you'll need to copy-and-paste the API key from your account settings.

Usage

```
connectApiUser(account = NULL, server = NULL, apiKey, quiet = FALSE)

connectUser(
  account = NULL,
  server = NULL,
  quiet = FALSE,
  launch.browser = getOption("rsconnect.launch.browser", interactive())
)
```

Arguments

`account`	A name for the account to connect.
`server`	The server to connect to.
`apiKey`	The API key used to authenticate the user
`quiet`	Whether or not to show messages and prompts while connecting the account.
`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.

See Also

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

connectSPCSUser *Register account on Posit Connect in Snowpark Container Services*

Description

`connectSPCSUser()` connects your Posit Connect account to the rsconnect package so it can deploy and manage applications on your behalf. Configure a `connections.toml` file in the appropriate location.

Usage

```
connectSPCSUser(
  account = NULL,
  server = NULL,
  snowflakeConnectionName,
  quiet = FALSE
)
```

Arguments

`account`	A name for the account to connect.
`server`	The server to connect to.
`snowflakeConnectionName`	Name for the Snowflake connection parameters stored in `connections.toml`.
`quiet`	Whether or not to show messages and prompts while connecting the account.

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

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

Arguments

`api`	Path to the API project directory. Must contain either `entrypoint.R` or `plumber.R` (for plumber APIs) or `_server.yml` (for plumber2 APIs)
`...`	Additional arguments to `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. Alternatively, deploy a plumber2 API by supplying a directory containing `_server.yml`.

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,
  envVars = NULL,
  appId = NULL,
  appMode = NULL,
  contentCategory = NULL,
  account = NULL,
  server = NULL,
  upload = TRUE,
  recordDir = NULL,
  launch.browser = getOption("rsconnect.launch.browser", is_interactive()),
  on.failure = NULL,
  logLevel = c("normal", "quiet", "verbose"),
  lint = TRUE,
  metadata = list(),
  forceUpdate = NULL,
  python = NULL,
  forceGeneratePythonEnvironment = FALSE,
  quarto = NA,
  appVisibility = NULL,
  image = NULL,
  envManagement = NULL,
```

```
  envManagementR = NULL,
  envManagementPy = NULL,
  space = NULL
)
```

Arguments

appDir	A directory containing an application (e.g. a Shiny app or plumber API). Defaults to the current directory.
appFiles, appFileManifest	Use appFiles to specify a character vector of files to bundle in the app or appFileManifest to provide a path to a file containing a list of such files. If neither are supplied, will bundle all files in appDir, apart from standard exclusions and files listed in a .rscignore file. See listDeploymentFiles() for more details.
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	**[Deprecated]** Please use recordDir instead.
appName	Application name, a string consisting of letters, numbers, _ and -. The application name is used to identify applications on a server, so must be unique. If not specified, the first deployment will be automatically it from the appDir for directory and website, and from the appPrimaryDoc for document. On subsequent deploys, it will use the previously stored value.
appTitle	Free-form descriptive title of application. Optional; if supplied, will often be displayed in favor of the name. If ommitted, on second and subsequent deploys, the title will be unchanged.
envVars	A character vector giving the names of environment variables whose values should be synchronised with the server (currently supported by Connect only). The values of the environment variables are sent over an encrypted connection and are not stored in the bundle, making this a safe way to send private data to Connect. The names (not values) are stored in the deployment record so that future deployments will automatically update their values. Other environment variables on the server will not be affected. This means that removing an environment variable from envVars will leave it unchanged on the server. To remove it, either delete it using the Connect UI, or temporarily unset it (with Sys.unsetenv() or similar) then re-deploy. Environment variables are set prior to deployment so that your code can use them and the first deployment can still succeed. Note that means that if the deployment fails, the values will still be updated.
appId	Use this to deploy to an exact known application, ignoring all existing deployment records and appName. You can use this to update an existing application that is missing a deployment record. If you're re-deploying an application that you created it's generally easier to use appName; appId is best reserved for re-deploying apps created by someone else.

You can find the appId in the following places:

- On shinyapps.io, it's the id listed on the applications page.
- For Posit Connect, it's guid from the info tab on the content page.

appMode — Optional; the type of content being deployed. Provide this option when the inferred type of content is incorrect. This can happen, for example, when static HTML content includes a downloadable Shiny application app.R. Accepted values include "shiny", "api", "rmd-static", "rmd-shiny", "quarto-static", "quarto-shiny", and "static". The Posit Connect API Reference contains a full set of available values. Not all servers support all types of content.

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

account, server — Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one.

Use accounts() to see the full list of available options.

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 Posit Connect.

recordDir — Directory where deployment record is written. The default, NULL, uses appDir, since this is usually where you want the deployment data to be stored. This argument is typically only needed when deploying a directory of static files since you want to store the record with the code that generated those files, not the files themselves.

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 paramter.

on.failure — Function to be called if the deployment fails. If a deployment log URL is available, it's passed 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().

Multi-value fields are recorded as comma-separated values and returned in that form. Custom value serialization is the responsibility of the caller.

forceUpdate — What should happen if there's no deployment record for the app, but there's an app with the same name on the server? If TRUE, will always update the previously-deployed app. If FALSE, will ask the user what to do, or fail if not in an interactive context.

Defaults to TRUE when called automatically by the IDE, and FALSE otherwise. You can override the default by setting option rsconnect.force.update.apps.

python — 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 or RETICULATE_PYTHON_FALLBACK 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.

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

quarto — Should the deployed content be built by quarto? (TRUE, FALSE, or NA). The default, NA, will use quarto if there are .qmd files in the bundle, or if there is a _quarto.yml and .Rmd files.

(This option is ignored and quarto will always be used if the metadata contains quarto_version and quarto_engines fields.)

appVisibility — One of NULL, "private", or "public"; the visibility of the deployment. When NULL, no change to visibility is made. Currently has an effect only on deployments to shinyapps.io.

image — Optional. The name of the image to use when building and executing this content. If none is provided, Posit Connect will attempt to choose an image based on the content requirements. You can override the default by setting the environment variable RSCONNECT_IMAGE.

envManagement — Optional. Should Posit Connect install R and Python packages for this content? (TRUE, FALSE, or NULL). The default, NULL, will not write any values to the bundle manifest, and Connect will fall back to the application default environment management strategy, or the server default if no application default is defined.

(This option is a shorthand flag which overwrites the values of both envManagementR and envManagementPy.)

envManagementR — Optional. Should Posit Connect install R packages for this content? (TRUE, FALSE, or NULL). The default, NULL, will not write any values to the bundle manifest, and Connect will fall back to the application default R environment management strategy, or the server default if no application default is defined.

(This option is ignored when envManagement is non-NULL.)

envManagementPy — Optional. Should Posit Connect install Python packages for this content? (TRUE, FALSE, or NULL). The default, NULL, will not write any values to the bundle manifest, and Connect will fall back to the application default Python environment management strategy, or the server default if no application default is defined.

(This option is ignored when envManagement is non-NULL.)

space — Optional. For Posit Cloud, the id of the space where the content should be deployed. If none is provided, content will be deployed to the deploying user's workspace or deployed to the same space in case of redeploy.

Details

Deployment records:

When deploying an app, deployApp() will save a deployment record that makes it easy to update the app on server from your local source code. This generally means that you need to only need to supply important arguments (e.g. appName, appTitle, server/account) on the first deploy, and rsconnect will reuse the same settings on subsequent deploys.

The metadata needs to make this work is stored in {appDir}/rsconnect/. You should generally check these files into version control to ensure that future you and other collaborators will publish to the same location.

If you have lost this directory, all is not lost, as deployApp() will attempt to rediscover existing deployments. This is easiest if you are updating an app that you created, as you can just supply the appName (and server/account if you have multiple accounts) and deployApp() will find the existing application account. If you need to update an app that was created by someone else (that you have write permission) for, you'll instead need to supply the appId.

See Also

applications(), terminateApp(), and restartApp()

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

Examples

```
## Not run:

# deploy the application in the current working dir
deployApp()

# deploy an application in another directory
deployApp("~/projects/shiny/app1")

# deploy using an alternative application name and title
deployApp("~/projects/shiny/app1", appName = "myapp",
          appTitle = "My Application")

# deploy specifying an explicit account name, then
# redeploy with no arguments (will automatically use
# the previously specified account)
deployApp(account = "jsmith")
deployApp()

# deploy but don't launch a browser when completed
deployApp(launch.browser = FALSE)

# deploy a Quarto website, using the quarto package to
# find the Quarto binary
deployApp("~/projects/quarto/site1")

# deploy application with environment variables
# (e.g., `SECRET_PASSWORD=XYZ` is set via an ~/.Renviron file)
rsconnect::deployApp(envVars = c("SECRET_PASSWORD"))

## End(Not run)
```

deployDoc *Deploy a single document*

Description

Deploys a single R Markdown, Quarto document, or other file (e.g. `.html` or `.pdf`).

When deploying an `.Rmd`, `.Qmd`, or `.html`, `deployDoc()` will attempt to automatically discover dependencies using `rmarkdown::find_external_resources()`, and include an `.Rprofile` if present. If you find that the document is missing dependencies, either specify the dependencies explicitly in the document (see `rmarkdown::find_external_resources()` for details), or call `deployApp()` directly and specify your own file list in `appFiles`.

Usage

```
deployDoc(doc, ..., logLevel = c("normal", "quiet", "verbose"))
```

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.
`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.

See Also

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

Examples

```
## Not run:
deployDoc("my-report.Rmd")
deployDoc("static-file.html")

## End(Not run)
```

deployments *List Application Deployments*

Description

List deployment records for a given application.

Usage

```
deployments(
  appPath = ".",
  nameFilter = NULL,
  accountFilter = NULL,
  serverFilter = NULL,
  excludeOrphaned = TRUE
)
```

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
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*

Description

Deploy an R Markdown or quarto website to a server.

Usage

```
deploySite(
  siteDir = getwd(),
  siteName = NULL,
  siteTitle = 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,
  recordDir = NULL,
  ...
)
```

Arguments

`siteDir`	Directory containing website. Defaults to current directory.
`siteName`	Name for the site (names must be unique within an account). Defaults to the base name of the specified `siteDir` or to the name provided by a custom site generation function.
`siteTitle`	Title for the site. For quarto sites only, if not supplied uses the title recorded in `_quarto.yml`.
`account, server`	Uniquely identify a remote server with either your user `account`, the `server` name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use `accounts()` to see the full list of available options.

render	Rendering behavior for site: • "none" uploads a static version of the current contents of the site directory. • "local" renders the site locally then uploads it. • "server" uploads the source of the site to render on the server. Note that for "none" and "local" source files (e.g. .R, .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 paramter.
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(). Multi-value fields are recorded as comma-separated values and returned in that form. Custom value serialization is the responsibility of the caller.
python	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 or RETICULATE_PYTHON_FALLBACK 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.
recordDir	The default, NULL, uses siteDir.
...	Additional arguments to deployApp(). Do not supply appDir or appFiles; these 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.

Usage

```
deployTFModel(...)
```

Arguments

...	Additional arguments to deployApp().

See Also

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

`forgetDeployment` *Forget Application 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

```
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.

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.

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

Usage

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

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."
))
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."
))
```

---

listAccountEnvVars *Maintain environment variables across multiple applications*

Description

- `listAccountEnvVars()` lists the environment variables used by every application published to the specified account.

- `updateAccountEnvVars()` updates the specified environment variables with their current values for every app that uses them.

Secure environment variable are currently only supported by Posit Connect so other server types will generate an error.

Usage

```
listAccountEnvVars(server = NULL, account = NULL)

updateAccountEnvVars(envVars, server = NULL, account = NULL)
```

Arguments

`account`, `server`	Uniquely identify a remote server with either your user `account`, the `server` name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use `accounts()` to see the full list of available options.
`envVars`	Names of environment variables to update. Their values will be automatically retrieved from the current process. If you specify multiple environment variables, any application that uses any of them will be updated with all of them.

Value

`listAccountEnvVars()` returns a data frame with one row for each data frame. It has variables `id`, `guid`, `name`, and `envVars`. `envVars` is a list-column.

`listDeploymentFiles` *Gather files to be bundled with an app*

Description

Given an app directory, and optional `appFiles` and `appFileManifest` arguments, returns vector of paths to bundle in the app. (Note that documents follow a different strategy; see `deployDoc()` for details.)

When neither `appFiles` nor `appFileManifest` is supplied, `listDeploymentFiles()` will include all files under `appDir`, apart from the following:

- Certain files and folders that don't need to be bundled, such as version control directories, internal config files, and RStudio state, are automatically excluded.
- You can exclude additional files by listing them in in a `.rscignore` file. This file must have one file or directory per line (with path relative to the current directory). It doesn't support wildcards, or ignoring files in subdirectories.

`listDeploymentFiles()` will throw an error if the total file size exceeds the maximum bundle size (as controlled by option `rsconnect.max.bundle.size`), or the number of files exceeds the maximum file limit (as controlled by option `rsconnect.max.bundle.files`). This prevents you from accidentally bundling a very large direcfory (i.e. you home directory).

Usage

```
listDeploymentFiles(
  appDir,
  appFiles = NULL,
  appFileManifest = NULL,
  error_call = caller_env()
)
```

Arguments

appDir	A directory containing an application (e.g. a Shiny app or plumber API). Defaults to the current directory.
appFiles, appFileManifest	Use appFiles to specify a character vector of files to bundle in the app or appFileManifest to provide a path to a file containing a list of such files. If neither are supplied, will bundle all files in appDir, apart from standard exclusions and files listed in a .rscignore file. See listDeploymentFiles() for more details.
error_call	The call or environment for error reporting; expert use only.

Value

Character of paths to bundle, relative to appDir.

---

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.

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, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.

Note

This function works only for ShinyApps servers.

See Also

addAuthorizedUser() and showUsers()

resendInvitation *Resend invitation for invited users of an application*

Description

Resend invitation for invited users of an application

Usage

```
resendInvitation(
  invite,
  regenerate = FALSE,
  appDir = getwd(),
  appName = NULL,
  account = NULL,
  server = NULL
)
```

Arguments

invite	The invitation to resend. Can be id or email address.
regenerate	Regenerate the invite code. Can be helpful is the invitation has expired.
appDir	Directory containing application. Defaults to current working directory.
appName	Name of application.
account, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.

Note

This function works only for ShinyApps servers.

See Also

showInvited()

---

restartApp *Restart an Application*

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

```
## 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

```
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

```
## 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)
```

`rsconnectOptions` *Package Options*

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 specied in the environment variable `RSCONNECT_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.tar` By default, `rsconnect` uses R's internal `tar` implementation to compress content bundles. This may cause invalid bundles in some environments. In those cases, use this option to specify a path to an alternate `tar` executable. This option can also be specified in the environment variable `RSCONNECT_TAR`. Leave undefined to use the default `tar` implementation.

`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.

`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 directory containing the content about to be deployed.

`rsconnect.post.deploy` A function to run after successfully deploying content; it receives as an argument the directory containing the content about to be 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 Posit 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)
```

---

servers *Server metadata*

Description

`servers()` lists all known servers; `serverInfo()` gets metadata about a specific server. Cloud servers `shinyapps.io` and `posit.cloud` are always automatically registered and available.

Usage

```
servers(local = FALSE)

serverInfo(name = NULL)
```

Arguments

`local`	Return only local servers? (i.e. not automatically registered cloud servers)
`name`	Server name. If omitted, you'll be prompted to pick a server.

Value

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

Examples

```
# List all registered servers
servers()

# Get information about a server
serverInfo("posit.cloud")
```

`setAccountInfo` *Register account on shinyapps.io or posit.cloud*

Description

Configure a ShinyApps or Posit Cloud account for publishing from this system.

Usage

```
setAccountInfo(name, token, secret, server = "shinyapps.io")
```

Arguments

`name`	Name of account to save or remove
`token`	User token for the account
`secret`	User secret for the account
`server`	Server to associate account with.

See Also

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

Examples

```
## Not run:

# 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,
  propertyValue,
  appPath = getwd(),
  appName = NULL,
  account = NULL,
  server = NULL,
  force = FALSE
)
```

Arguments

propertyName	Name of property
propertyValue	Property value
appPath	Directory or file that was deployed. Defaults to current working directory.
appName	Name of application
account, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.
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, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.

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,
  server = NULL,
  entries = 50,
  streaming = FALSE
)
```

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.
server	Server name. Required only if you use the same account name on multiple servers.
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. This function only works for ShinyApps servers.

Usage

```
showMetrics(
  metricSeries,
  metricNames,
  appDir = getwd(),
  appName = NULL,
  account = NULL,
  server = "shinyapps.io",
  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	A directory containing an application (e.g. a Shiny app or plumber API). Defaults to the current directory.
appName	Application name, a string consisting of letters, numbers, _ and -. The application name is used to identify applications on a server, so must be unique. If not specified, the first deployment will be automatically it from the appDir for directory and website, and from the appPrimaryDoc for document. On subsequent deploys, it will use the previously stored value.
account, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.
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").

---

showProperties *Show Application property*

Description

Show properties of an application deployed to ShinyApps.

Usage

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

Arguments

appPath	Directory or file that was deployed. Defaults to current working directory.
appName	Name of application
account, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.

Note

This function works only for ShinyApps servers.

---

showUsage *Show Application Usage*

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, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.
usageType	Use metric to retreive (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 then 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, server`	Uniquely identify a remote server with either your user `account`, the `server` name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use `accounts()` to see the full list of available options.

Note

This function works only for ShinyApps servers.

See Also

`addAuthorizedUser()` and `showInvited()`

`syncAppMetadata` *Update deployment records*

Description

Update the deployment records for applications published to Posit Connect. This updates application title and URL, and deletes records for deployments where the application has been deleted on the server.

Usage

```
syncAppMetadata(appPath = ".")
```

Arguments

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

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, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.
output	Where to write output. Valid values are NULL or stderr

Note

This function works only with shinyapps.io and posit.cloud.

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)
```

Arguments

account, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.

Value

Returns a data frame with the following columns:

id	Task id
action	Task action
status	Current task status
created_time	Task creation time
finished_time	Task finished time

Note

This function works only with shinyapps.io and posit.cloud.

See Also

taskLog()

Examples

```
## Not run:

# list tasks for the default account
tasks()


## End(Not run)
```

terminateApp *Terminate an Application*

Description

Terminate and archive a currently deployed ShinyApps application.

Usage

```
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())
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

```
## 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

```
unsetProperty(
  propertyName,
  appPath = getwd(),
  appName = NULL,
  account = NULL,
  server = NULL,
  force = FALSE
)
```

Arguments

propertyName	Name of property
appPath	Directory or file that was deployed. Defaults to current working directory.
appName	Name of application
account, server	Uniquely identify a remote server with either your user account, the server name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use accounts() to see the full list of available options.
force	Forcibly unset the property

Note

This function only works for ShinyApps servers.

Examples

```
## Not run:

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


## End(Not run)
```

writeManifest *Create a* manifest.json

Description

Use writeManifest() to generate a manifest.json. Among other things, you can commit this file to git to activate Git-Backed content for Posit Connect.

manifest.json contains a list of all files in the app along with their dependencies, so you will need to re-run writeManifest() when either of these change.

Usage

```
writeManifest(
  appDir = getwd(),
  appFiles = NULL,
  appFileManifest = NULL,
  appPrimaryDoc = NULL,
  appMode = NULL,
  contentCategory = NULL,
  python = NULL,
  forceGeneratePythonEnvironment = FALSE,
  quarto = NA,
  image = NULL,
  envManagement = NULL,
  envManagementR = NULL,
  envManagementPy = NULL,
  verbose = FALSE,
  quiet = FALSE
)
```

Arguments

appDir	A directory containing an application (e.g. a Shiny app or plumber API). Defaults to the current directory.
appFiles, appFileManifest	Use appFiles to specify a character vector of files to bundle in the app or appFileManifest to provide a path to a file containing a list of such files. If neither are supplied, will bundle all files in appDir, apart from standard exclusions and files listed in a .rscignore file. See listDeploymentFiles() for more details.
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.
appMode	Optional; the type of content being deployed. Provide this option when the inferred type of content is incorrect. This can happen, for example, when static

HTML content includes a downloadable Shiny application `app.R`. Accepted values include `"shiny"`, `"api"`, `"rmd-static"`, `"rmd-shiny"`, `"quarto-static"`, `"quarto-shiny"`, and `"static"`. The Posit Connect API Reference contains a full set of available values. Not all servers support all types of content.

`contentCategory`
: Set this to `"site"` if you'd deploy with `deploySite()`; otherwise leave as is.

`python`
: 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 or RETICULATE_PYTHON_FALLBACK 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.

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

`quarto`
: Should the deployed content be built by quarto? (`TRUE`, `FALSE`, or `NA`). The default, `NA`, will use quarto if there are `.qmd` files in the bundle, or if there is a `_quarto.yml` and `.Rmd` files.

  (This option is ignored and quarto will always be used if the `metadata` contains `quarto_version` and `quarto_engines` fields.)

`image`
: Optional. The name of the image to use when building and executing this content. If none is provided, Posit Connect will attempt to choose an image based on the content requirements. You can override the default by setting the environment variable `RSCONNECT_IMAGE`.

`envManagement`
: Optional. Should Posit Connect install R and Python packages for this content? (`TRUE`, `FALSE`, or `NULL`). The default, `NULL`, will not write any values to the bundle manifest, and Connect will fall back to the application default environment management strategy, or the server default if no application default is defined.

  (This option is a shorthand flag which overwrites the values of both `envManagementR` and `envManagementPy`.)

`envManagementR`
: Optional. Should Posit Connect install R packages for this content? (`TRUE`, `FALSE`, or `NULL`). The default, `NULL`, will not write any values to the bundle manifest, and Connect will fall back to the application default R environment management strategy, or the server default if no application default is defined.

  (This option is ignored when `envManagement` is non-`NULL`.)

`envManagementPy`
: Optional. Should Posit Connect install Python packages for this content? (`TRUE`, `FALSE`, or `NULL`). The default, `NULL`, will not write any values to the bundle manifest, and Connect will fall back to the application default Python environment management strategy, or the server default if no application default is defined.

  (This option is ignored when `envManagement` is non-`NULL`.)

`verbose`
: If `TRUE`, prints detailed progress messages.

`quiet`
: If `FALSE`, prints progress messages.

Index