Package 'googleCloudRunner'

January 30, 2021

Type Package

Title R Scripts in the Google Cloud via Cloud Run, Cloud Build and Cloud Scheduler

Version 0.4.1

Description Tools to easily enable R scripts in the Google Cloud Platform. Utilise cloud services such as Cloud Run <https://cloud.google.com/run/> for R over HTTP, Cloud Build <https://cloud.google.com/cloud-build/> for Continuous Delivery and Integration services and Cloud Scheduler <https://cloud.google.com/scheduler/> for scheduled scripts.

URL https://code.markedmondson.me/googleCloudRunner/

BugReports https://github.com/MarkEdmondson1234/googleCloudRunner/issues

Depends R (>= 3.3.0)

Imports assertthat (>= 0.2.0), cli (>= 2.0.2), curl (>= 4.3), googleAuthR (>= 1.3.1), googleCloudStorageR (>= 0.5.1), httr (>= 1.4.1), jose (>= 1.0), jsonlite (>= 1.5), openssl (>= 1.4.1), plumber (>= 1.0.0), progress (>= 1.2.2), usethis (>= 1.6.0), utils, yaml (>= 2.2.0)

Suggests knitr, miniUI, rmarkdown, rstudioapi, shiny, testthat (>= 2.1.0)

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

Config/testthat/edition 3

Config/testthat/parallel true

NeedsCompilation no

Author Mark Edmondson [aut, cre] (<https://orcid.org/0000-0002-8434-3881>), Sunholo Ltd [cph]

Maintainer Mark Edmondson <r@sunholo.com>

Repository CRAN

Date/Publication 2021-01-30 17:30:02 UTC

R topics documented:

`Build` *Build Object*

Description

Build Object

Usage

```
Build(
  Build.substitutions = NULL,
  Build.timing = NULL,
  results = NULL,
  logsBucket = NULL,
  steps = NULL,
  buildTriggerId = NULL,
```

```
  id = NULL,
  tags = NULL,
  startTime = NULL,
  substitutions = NULL,
  timing = NULL,
  sourceProvenance = NULL,
  createTime = NULL,
  images = NULL,
  projectId = NULL,
  logUrl = NULL,
  finishTime = NULL,
  source = NULL,
  options = NULL,
  timeout = NULL,
  status = NULL,
  statusDetail = NULL,
  artifacts = NULL,
  secrets = NULL
)
```

Arguments

Build.substitutions	The Build.substitutions object or list of objects
Build.timing	The Build.timing object or list of objects
results	Output only
logsBucket	Google Cloud Storage bucket where logs should be written (see
steps	Required
buildTriggerId	Output only
id	Output only
tags	Tags for annotation of a 'Build'
startTime	Output only
substitutions	Substitutions data for 'Build' resource
timing	Output only
sourceProvenance	Output only
createTime	Output only
images	A list of images to be pushed upon the successful completion of all build
projectId	Output only
logUrl	Output only
finishTime	Output only
source	A Source object specifying the location of the source files to build, usually created by cr_build_source
options	Special options for this build

`timeout`	Amount of time that this build should be allowed to run, to second
`status`	Output only
`statusDetail`	Output only
`artifacts`	Artifacts produced by the build that should be uploaded upon
`secrets`	Secrets to decrypt using Cloud Key Management Service

Details

A build resource in the Cloud Build API.

At a high level, a 'Build' describes where to find source code, how to build it (for example, the builder image to run on the source), and where to store the built artifacts.

Value

Build object

Build Macros

Fields can include the following variables, which will be expanded when the build is created:-

- $PROJECT_ID: the project ID of the build.
- $BUILD_ID: the autogenerated ID of the build.
- $REPO_NAME: the source repository name specified by RepoSource.
- $BRANCH_NAME: the branch name specified by RepoSource.
- $TAG_NAME: the tag name specified by RepoSource.
- $REVISION_ID or $COMMIT_SHA: the commit SHA specified by RepoSource or resolved from the specified branch or tag.
- $SHORT_SHA: first 7 characters of $REVISION_ID or $COMMIT_SHA.

See Also

Other Cloud Build functions: `RepoSource()`, `Source()`, `StorageSource()`, `cr_build_artifacts()`, `cr_build_make()`, `cr_build_status()`, `cr_build_upload_gcs()`, `cr_build_wait()`, `cr_build_write()`, `cr_build_yaml_artifact()`, `cr_build_yaml()`, `cr_build()`

`BuildTrigger` *BuildTrigger Object*

Description

Configuration for an automated build in response to source repositorychanges.

Usage

```
BuildTrigger(
  filename = NULL,
  name = NULL,
  tags = NULL,
  build = NULL,
  ignoredFiles = NULL,
  github = NULL,
  substitutions = NULL,
  includedFiles = NULL,
  disabled = NULL,
  triggerTemplate = NULL,
  description = NULL
)
```

Arguments

`filename`	Path, from the source root, to a file whose contents is used for the build
`name`	User assigned name of the trigger
`tags`	Tags for annotation of a 'BuildTrigger'
`build`	Contents of the build template
`ignoredFiles`	ignored_files and included_files are file glob matches extended with support for "**".
`github`	a GitHubEventsConfig object - mutually exclusive with `triggerTemplate`
`substitutions`	A named list of Build macro variables
`includedFiles`	If any of the files altered in the commit pass the ignored_files
`disabled`	If true, the trigger will never result in a build
`triggerTemplate`	a RepoSource object - mutually exclusive with `github`
`description`	Human-readable description of this trigger

Value

BuildTrigger object

See Also

https://cloud.google.com/cloud-build/docs/api/reference/rest/v1/projects.triggers

Other BuildTrigger functions: GitHubEventsConfig(), cr_buildtrigger_copy(), cr_buildtrigger_delete(), cr_buildtrigger_edit(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_repo(), cr_buildtrigger_run(), cr_buildtrigger()

---

cr_bucket_set *Get/Set the Cloud Storage bucket for your Cloud Build Service*

Description

Can also use environment arg GCS_DEFAULT_BUCKET

Usage

```
cr_bucket_set(bucket)

cr_bucket_get()
```

Arguments

bucket	The GCS bucket

Examples

```
cr_bucket_get()
```

---

cr_build *Starts a build with the specified configuration.*

Description

This method returns a long-running ‘Operation‘, which includes the buildID. Pass the build ID to cr_build_status to determine the build status (such as ‘SUCCESS‘ or ‘FAILURE‘).

Usage

```
cr_build(
  x,
  source = NULL,
  timeout = NULL,
  images = NULL,
  substitutions = NULL,
  artifacts = NULL,
```

```
  options = NULL,
  projectId = cr_project_get(),
  launch_browser = interactive()
)
```

Arguments

x	A cloudbuild.yaml file location or an R object that will be turned into yaml via as.yaml or a Build object created by cr_build_make or from a previous build you want to rerun.
source	A Source object specifying the location of the source files to build, usually created by cr_build_source
timeout	Amount of time that this build should be allowed to run, to second
images	A list of images to be pushed upon the successful completion of all build
substitutions	Substitutions data for 'Build' resource
artifacts	Artifacts produced by the build that should be uploaded upon
options	Special options for this build
projectId	ID of the project
launch_browser	Whether to launch the logs URL in a browser once deployed

See Also

Google Documentation for Cloud Build

Other Cloud Build functions: Build(), RepoSource(), Source(), StorageSource(), cr_build_artifacts(), cr_build_make(), cr_build_status(), cr_build_upload_gcs(), cr_build_wait(), cr_build_write(), cr_build_yaml_artifact(), cr_build_yaml()

Examples

```
cr_project_set("my-project")
my_gcs_source <- cr_build_source(StorageSource("my_code.tar.gz",
                                               bucket = "gs://my-bucket"))
my_gcs_source

my_repo_source <- cr_build_source(RepoSource("github_username_my-repo.com",
                                             branchName="master"))
my_repo_source
## Not run:

# build from a cloudbuild.yaml file
cloudbuild_file <- system.file("cloudbuild/cloudbuild.yaml",
                               package="googleCloudRunner")

# asynchronous, will launch log browser by default
b1 <- cr_build(cloudbuild_file)

# synchronous waiting for build to finish
b2 <- cr_build_wait(b1)
```

```
# the same results
cr_build_status(b1)
cr_build_status(b2)

# build from a cloud storage source
build1 <- cr_build(cloudbuild_file,
                   source = my_gcs_source)
# build from a git repository source
build2 <- cr_build(cloudbuild_file,
                   source = my_repo_source)

# you can send in results for previous builds to trigger
# the same build under a new Id
# will trigger build2 again
cr_build(build2)

# a build with substitutions (Cloud Build macros)
cr_build(build2, substitutions = list(`_SUB` = "yo"))


## End(Not run)
```

`cr_buildstep` *Create a yaml build step*

Description

Helper for creating build steps for upload to Cloud Build

Usage

```
cr_buildstep(
  name,
  args = NULL,
  id = NULL,
  prefix = "gcr.io/cloud-builders/",
  entrypoint = NULL,
  dir = "",
  env = NULL,
  waitFor = NULL,
  volumes = NULL
)
```

Arguments

`name`	name of docker image to call appended to `prefix`
`args`	character vector of arguments

id	Optional id for the step
prefix	prefixed to name - set to "" to suppress. Will be suppressed if name starts with gcr.io
entrypoint	change the entrypoint for the docker container
dir	The directory to use, relative to /workspace e.g. /workspace/deploy/
env	Environment variables for this step. A character vector for each assignment
waitFor	Whether to wait for previous buildsteps to complete before running. Default it will wait for previous step.
volumes	volumes to connect and write to

Details

This uses R to make building steps for cloudbuild.yml files harder to make mistakes with, and also means you can program creation of cloud build steps for use in R or other languages. Various templates with common use cases of buildsteps are also available that wrap this function, refer to the "See Also" section.

WaitFor

By default each buildstep waits for the previous, but if you pass `"-"` then it will start immediately, or if you pass in a list of ids it will wait for previous buildsteps to finish who have that id. See Configuring Build Step Order for details.

Build Macros

Fields can include the following variables, which will be expanded when the build is created:-

- $PROJECT_ID: the project ID of the build.
- $BUILD_ID: the autogenerated ID of the build.
- $REPO_NAME: the source repository name specified by RepoSource.
- $BRANCH_NAME: the branch name specified by RepoSource.
- $TAG_NAME: the tag name specified by RepoSource.
- $REVISION_ID or $COMMIT_SHA: the commit SHA specified by RepoSource or resolved from the specified branch or tag.
- $SHORT_SHA: first 7 characters of $REVISION_ID or $COMMIT_SHA.

Or you can add your own custom variables, set in the Build Trigger. Custom variables always start with $_ e.g. $_MY_VAR

See Also

Creating custom build steps how-to guide

Other Cloud Buildsteps: `cr_buildstep_bash()`, `cr_buildstep_decrypt()`, `cr_buildstep_df()`, `cr_buildstep_docker()`, `cr_buildstep_edit()`, `cr_buildstep_extract()`, `cr_buildstep_gcloud()`, `cr_buildstep_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_secret()`, `cr_buildstep_slack()`

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")
# creating yaml for use in deploying cloud run
image = "gcr.io/my-project/my-image:$BUILD_ID"
cr_build_yaml(
    steps = c(
         cr_buildstep("docker", c("build","-t",image,".")),
         cr_buildstep("docker", c("push",image)),
         cr_buildstep("gcloud", c("beta","run","deploy", "test1",
                                  "--image", image))),
    images = image)

# use premade docker buildstep - combine using c()
image = "gcr.io/my-project/my-image"
cr_build_yaml(
    steps = c(cr_buildstep_docker(image),
              cr_buildstep("gcloud",
                     args = c("beta","run","deploy",
                              "test1","--image", image))
             ),
    images = image)

# list files with a new entrypoint for gcloud
cr_build_yaml(steps = cr_buildstep("gcloud", c("-c","ls -la"),
                                   entrypoint = "bash"))

# to call from images not using gcr.io/cloud-builders stem
cr_buildstep("alpine", c("-c","ls -la"), entrypoint = "bash", prefix="")

# to add environment arguments to the step
cr_buildstep("docker", "version", env = c("ENV1=env1", "ENV2=$PROJECT_ID"))

# to add volumes wrap in list()
cr_buildstep("test", "ls", volumes = list(list(name = "ssh", path = "/root/.ssh")))
```

---

cr_buildstep_bash *Run a bash script in a Cloud Build step*

Description

Helper to run a supplied bash script, that will be copied in-line

Usage

```
cr_buildstep_bash(
  bash_script,
  name = "ubuntu",
```

```
  bash_source = c("local", "runtime"),
  ...
)
```

Arguments

bash_script	bash code to run or a filepath to a file containing bash code that ends with .bash or .sh
name	The image that will run the R code
bash_source	Whether the code will be from a runtime file within the source or at build time copying over from a local file in your session
...	Other arguments passed to cr_buildstep

Details

If you need to escape build parameters in bash scripts, you need to escape CloudBuild's substitution via $$ and bash's substitution via $ e.g. $$PARAM

See Also

Other Cloud Buildsteps: cr_buildstep_decrypt(), cr_buildstep_df(), cr_buildstep_docker(), cr_buildstep_edit(), cr_buildstep_extract(), cr_buildstep_gcloud(), cr_buildstep_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_r(), cr_buildstep_secret(), cr_buildstep_slack(), cr_buildstep()

Examples

```
cr_project_set("my-project")
bs <- cr_build_yaml(
  steps = cr_buildstep_bash("echo 'Hello'")
 )

## Not run:
cr_build(bs)

## End(Not run)
```

---

cr_buildstep_decrypt *Create a build step for decrypting files via KMS*

Description

Create a build step to decrypt files using CryptoKey from Cloud Key Management Service. Usually you will prefer to use cr_buildstep_secret

Usage

```
cr_buildstep_decrypt(cipher, plain, keyring, key, location = "global", ...)
```

Arguments

cipher	The file that has been encrypted
plain	The file location to decrypt to
keyring	The KMS keyring to use
key	The KMS key to use
location	The KMS location
...	Further arguments passed in to cr_buildstep

Details

Key Management Store can encrypt secret files for use within your later buildsteps.

Setup

You will need to set up the encrypted key using gcloud following the link from Google

See Also

Other Cloud Buildsteps: cr_buildstep_bash(), cr_buildstep_df(), cr_buildstep_docker(), cr_buildstep_edit(), cr_buildstep_extract(), cr_buildstep_gcloud(), cr_buildstep_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_r(), cr_buildstep_secret(), cr_buildstep_slack(), cr_buildstep()

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")
cr_buildstep_decrypt("secret.json.enc",
                     plain = "secret.json",
                     keyring = "my_keyring",
                     key = "my_key")
```

---

cr_buildstep_df *Convert a data.frame into cr_buildstep*

Description

Helper to turn a data.frame of buildsteps info into format accepted by cr_build

Usage

```
cr_buildstep_df(x)
```

Arguments

x	A data.frame of steps to turn into buildsteps, with at least name and args columns

Details

This helps convert the output of cr_build into valid cr_buildstep so it can be sent back into the API

If constructing arg list columns then I suppresses conversion of the list to columns that would otherwise break the yaml format

See Also

Other Cloud Buildsteps: cr_buildstep_bash(), cr_buildstep_decrypt(), cr_buildstep_docker(), cr_buildstep_edit(), cr_buildstep_extract(), cr_buildstep_gcloud(), cr_buildstep_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_r(), cr_buildstep_secret(), cr_buildstep_slack(), cr_buildstep()

Examples

```
y <- data.frame(name = c("docker", "alpine"),
                args = I(list(c("version"), c("echo", "Hello Cloud Build"))),
                id = c("Docker Version", "Hello Cloud Build"),
                prefix = c(NA, ""),
                stringsAsFactors = FALSE)
cr_buildstep_df(y)
```

`cr_buildstep_docker` *Create a build step to build and push a docker image*

Description

Create a build step to build and push a docker image

Usage

```
cr_buildstep_docker(
  image,
  tag = c("latest", "$BUILD_ID"),
  location = ".",
  projectId = cr_project_get(),
  dockerfile = "Dockerfile",
  kaniko_cache = FALSE,
  ...
)
```

Arguments

`image`	The image tag that will be pushed, starting with gcr.io or created by combining with `projectId` if not starting with gcr.io
`tag`	The tag or tags to be attached to the pushed image - can use `Build` macros
`location`	Where the Dockerfile to build is in relation to `dir`

projectId	The projectId
dockerfile	Specify the name of the Dockerfile found at location
kaniko_cache	If TRUE will use kaniko cache for Docker builds.
...	Further arguments passed in to cr_buildstep

Details

Setting kaniko_cache = TRUE will enable caching of the layers of the Dockerfile, which will speed up subsequent builds of that Dockerfile. See Using Kaniko cache

If building multiple tags they don't have to run sequentially - set waitFor = "-" to build concurrently

See Also

Other Cloud Buildsteps: cr_buildstep_bash(), cr_buildstep_decrypt(), cr_buildstep_df(), cr_buildstep_edit(), cr_buildstep_extract(), cr_buildstep_gcloud(), cr_buildstep_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_r(), cr_buildstep_secret(), cr_buildstep_slack(), cr_buildstep()

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")

cr_buildstep_docker("gcr.io/my-project/my-image")
cr_buildstep_docker("my-image")
cr_buildstep_docker("my-image", tag = "$BRANCH_NAME")

# setting up a build to trigger off a Git source:
my_image <- "gcr.io/my-project/my-image"
my_repo <- RepoSource("github_markedmondson1234_googlecloudrunner",
                      branchName="master")
## Not run:
docker_yaml <- cr_build_yaml(steps = cr_buildstep_docker(my_image))
built_docker <- cr_build(docker_yaml, source = my_repo)

# make a build trigger so it builds on each push to master
cr_buildtrigger("build-docker", trigger = my_repo, build = built_docker)


# add a cache to your docker build to speed up repeat builds
cr_buildstep_docker("my-image", kaniko_cache = TRUE)

# building using manual buildsteps to clone from git
bs <- c(
  cr_buildstep_gitsetup("github-ssh"),
  cr_buildstep_git(c("clone","git@github.com:MarkEdmondson1234/googleCloudRunner",".")),
  cr_buildstep_docker("gcr.io/gcer-public/packagetools",
                      dir = "inst/docker/packages/")
  )
```

```
built <- cr_build(cr_build_yaml(bs))

## End(Not run)
```

`cr_buildstep_edit` *Modify an existing buildstep with new parameters*

Description

Useful for editing existing buildsteps

Usage

```
cr_buildstep_edit(x, ...)
```

Arguments

`x`	A buildstep created previously
`...`	Arguments passed on to `cr_buildstep` `name` name of docker image to call appended to `prefix` `args` character vector of arguments `prefix` prefixed to name - set to "" to suppress. Will be suppressed if `name` starts with gcr.io `entrypoint` change the entrypoint for the docker container `dir` The directory to use, relative to /workspace e.g. /workspace/deploy/ `id` Optional id for the step `env` Environment variables for this step. A character vector for each assignment `volumes` volumes to connect and write to `waitFor` Whether to wait for previous buildsteps to complete before running. Default it will wait for previous step.

See Also

Other Cloud Buildsteps: `cr_buildstep_bash()`, `cr_buildstep_decrypt()`, `cr_buildstep_df()`, `cr_buildstep_docker()`, `cr_buildstep_extract()`, `cr_buildstep_gcloud()`, `cr_buildstep_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_secret()`, `cr_buildstep_slack()`, `cr_buildstep()`

Examples

```
package_build <- system.file("cloudbuild/cloudbuild.yaml",
                             package = "googleCloudRunner")
build <- cr_build_make(package_build)
build
cr_buildstep_extract(build, step = 1)
```

```
cr_buildstep_extract(build, step = 2)

edit_me <- cr_buildstep_extract(build, step = 2)
cr_buildstep_edit(edit_me, name = "blah")
cr_buildstep_edit(edit_me, name = "gcr.io/blah")
cr_buildstep_edit(edit_me, args = c("blah1","blah2"), dir = "meh")
```

`cr_buildstep_extract` *Extract a buildstep from a Build object*

Description

Useful if you have a step from an existing cloudbuild.yaml you want in another

Usage

```
cr_buildstep_extract(x, step = NULL)
```

Arguments

`x`	A Build object
`step`	The numeric step number to extract

See Also

Other Cloud Buildsteps: `cr_buildstep_bash()`, `cr_buildstep_decrypt()`, `cr_buildstep_df()`, `cr_buildstep_docker()`, `cr_buildstep_edit()`, `cr_buildstep_gcloud()`, `cr_buildstep_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_secret()`, `cr_buildstep_slack()`, `cr_buildstep()`

Examples

```
package_build <- system.file("cloudbuild/cloudbuild.yaml",
                             package = "googleCloudRunner")
build <- cr_build_make(package_build)
build
cr_buildstep_extract(build, step = 1)
cr_buildstep_extract(build, step = 2)
```

`cr_buildstep_gcloud` *A buildstep template for gcloud*

Description

This enables an optimised version of gcloud docker for your buildstep such as `gcr.io/google.com/cloudsdktool/cloud-s`

Usage

```
cr_buildstep_gcloud(component = c("gcloud", "bq", "gsutil", "kubectl"), ...)
```

Arguments

`component`	What gcloud service you need, such as "gcloud", "bq" or "gsutil"
`...`	Arguments passed on to `cr_buildstep`
	`name` name of docker image to call appended to `prefix`
	`args` character vector of arguments
	`prefix` prefixed to name - set to "" to suppress. Will be suppressed if `name` starts with gcr.io
	`entrypoint` change the entrypoint for the docker container
	`dir` The directory to use, relative to /workspace e.g. /workspace/deploy/
	`id` Optional id for the step
	`env` Environment variables for this step. A character vector for each assignment
	`volumes` volumes to connect and write to
	`waitFor` Whether to wait for previous buildsteps to complete before running. Default it will wait for previous step.

See Also

https://github.com/GoogleCloudPlatform/cloud-builders/tree/master/gcloud

Other Cloud Buildsteps: `cr_buildstep_bash()`, `cr_buildstep_decrypt()`, `cr_buildstep_df()`, `cr_buildstep_docker()`, `cr_buildstep_edit()`, `cr_buildstep_extract()`, `cr_buildstep_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_secret()`, `cr_buildstep_slack()`, `cr_buildstep()`

`cr_buildstep_gitsetup` *Create a build step for authenticating with Git*

Description

This creates steps to configure git to use an ssh created key.

This creates steps to use git with an ssh created key.

Usage

```
cr_buildstep_gitsetup(secret, post_setup = NULL)

cr_buildstep_git(
  git_args = c("clone", "git@github.com:[GIT-USERNAME]/[REPOSITORY]", "."),
  ...
)

git_volume()
```

Arguments

`secret`	The name of the secret on Google Secret Manager for the git ssh private key
`post_setup`	Steps that occur after git setup
`git_args`	The arguments to send to git
`...`	Further arguments passed in to cr_buildstep

Details

The ssh private key should be uploaded to Google Secret Manager first

`cr_buildstep` must come after `cr_buildstep_gitsetup`

Use `git_volume` to add the git credentials folder to other buildsteps

See Also

Accessing private GitHub repositories using Cloud Build (google article)

Other Cloud Buildsteps: `cr_buildstep_bash()`, `cr_buildstep_decrypt()`, `cr_buildstep_df()`, `cr_buildstep_docker()`, `cr_buildstep_edit()`, `cr_buildstep_extract()`, `cr_buildstep_gcloud()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_secret()`, `cr_buildstep_slack()`, `cr_buildstep()`

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")

# assumes you have previously saved git ssh key called "github-ssh"
cr_build_yaml(
     steps = c(
          cr_buildstep_gitsetup("github-ssh"),
          cr_buildstep_git(c("clone",
                             "git@github.com:github_name/repo_name"))
     )
 )
```

`cr_buildstep_mailgun` *Send an email in a Cloud Build step via MailGun.org*

Description

This uses Mailgun to send emails. It calls an R script that posts the message to MailGuns API.

Usage

```
cr_buildstep_mailgun(
  message,
  to,
  subject,
  from,
  mailgun_url = "$_MAILGUN_URL",
  mailgun_key = "$_MAILGUN_KEY",
  ...
)
```

Arguments

`message`	The message markdown
`to`	to email
`subject`	subject email
`from`	from email
`mailgun_url`	The Mailgun API base URL. Default assumes you set this in Build substitution macros
`mailgun_key`	The Mailgun API key. Default assumes you set this in Build substitution macros
`...`	Other arguments passed to cr_buildstep_r

Details

Requires an account at Mailgun: https://mailgun.com Pre-verification you can only send to a whitelist of emails you configure - see Mailgun website for details.

See Also

Other Cloud Buildsteps: cr_buildstep_bash(), cr_buildstep_decrypt(), cr_buildstep_df(), cr_buildstep_docker(), cr_buildstep_edit(), cr_buildstep_extract(), cr_buildstep_gcloud(), cr_buildstep_gitsetup(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_r(), cr_buildstep_secret(), cr_buildstep_slack(), cr_buildstep()

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")
mailgun_url <- "https://api.mailgun.net/v3/sandboxXXX.mailgun.org"
mailgun_key <- "key-XXXX"

## Not run:
# assumes you have verified the email
cr_build(
  cr_build_yaml(steps = cr_buildstep_mailgun(
                            "Hello from Cloud Build",
                            to = "me@verfied_email.com",
                            subject = "Hello",
                            from = "googleCloudRunner@example.com"),
                substitutions = list(
                  `_MAILGUN_URL` = mailgun_url,
                  `_MAILGUN_KEY` = mailgun_key)
          ))

## End(Not run)
```

---

cr_buildstep_nginx_setup

Setup nginx for Cloud Run in a buildstep

Description

Setup nginx for Cloud Run in a buildstep

Usage

```
cr_buildstep_nginx_setup(html_folder, ...)
```

Arguments

html_folder	The folder that will hold the HTML for Cloud Run This uses a premade bash script that sets up a Docker container ready for Cloud Run running nginx
...	Other arguments passed to cr_buildstep_bash

See Also

Other Cloud Buildsteps: cr_buildstep_bash(), cr_buildstep_decrypt(), cr_buildstep_df(), cr_buildstep_docker(), cr_buildstep_edit(), cr_buildstep_extract(), cr_buildstep_gcloud(), cr_buildstep_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_r(), cr_buildstep_secret(), cr_buildstep_slack(), cr_buildstep()

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")
cr_region_set("europe-west1")

html_folder <- "my_html"
run_image <- "gcr.io/my-project/my-image-for-cloudrun"
cr_build_yaml(
 steps = c(
  cr_buildstep_nginx_setup(html_folder),
  cr_buildstep_docker(run_image, dir = html_folder),
  cr_buildstep_run(name = "running-nginx",
                   image = run_image,
                   concurrency = 80)
                   )
           )
```

---

cr_buildstep_packagetests

Do R package tests and upload to Codecov

Description

This lets you run R package tests and is intended to be used in a trigger when you push to a repository so you can monitor code quality.

Usage

```
cr_buildstep_packagetests(
  test_script = NULL,
  codecov_script = NULL,
  codecov_token = "$_CODECOV_TOKEN",
  build_image = "gcr.io/gcer-public/packagetools:latest",
  env = c("NOT_CRAN=true")
)
```

Arguments

test_script	The script that will perform tests. If NULL a default script is used in system.file("r_buildsteps","dev
codecov_script	The script that will perform coverage. If NULL a default script is used in system.file("r_buildsteps","'
codecov_token	If using codecov, supply your codecov token here.
build_image	The docker image that will be used to run the R code for the test scripts
env	Environment arguments to be set during the test script runs

Examples

```
cr_buildstep_packagetests()
```

---

cr_buildstep_pkgdown *Create buildsteps for deploying an R pkgdown website to GitHub*

Description

Create buildsteps for deploying an R pkgdown website to GitHub

Usage

```
cr_buildstep_pkgdown(
  github_repo,
  git_email,
  secret,
  env = NULL,
  build_image = "gcr.io/gcer-public/packagetools:latest",
  post_setup = NULL,
  post_clone = NULL
)
```

Arguments

github_repo	The GitHub repo to deploy pkgdown website from and to.
git_email	The email the git commands will be identifying as
secret	The name of the secret on Google Secret Manager for the git ssh private key
env	A character vector of env arguments to set for all steps
build_image	A docker image with pkgdown installed
post_setup	Steps that occur after git setup
post_clone	A cr_buildstep that occurs after the repo is cloned

Details

Its convenient to set some of the above via Build macros, such as `github_repo=$_GITHUB_REPO` and `git_email=$_BUILD_EMAIL` in the Build Trigger web UI

To commit the website to git, cr_buildstep_gitsetup is used for which you will need to add your git ssh private key to Google Secret Manager

The R package is installed via install before running build_site

See Also

Other Cloud Buildsteps: `cr_buildstep_bash()`, `cr_buildstep_decrypt()`, `cr_buildstep_df()`, `cr_buildstep_docker()`, `cr_buildstep_edit()`, `cr_buildstep_extract()`, `cr_buildstep_gcloud()`, `cr_buildstep_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_secret()`, `cr_buildstep_slack()`, `cr_buildstep()`

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")

# set github repo directly to write it out via cr_build_write()
cr_buildstep_pkgdown("MarkEdmondson1234/googleCloudRunner",
                     git_email = "cloudbuild@google.com",
                     secret = "github-ssh")

# github repo set via build trigger macro _GITHUB_REPO
cr_buildstep_pkgdown("$_GITHUB_REPO",
                     git_email = "cloudbuild@google.com",
                     secret = "github-ssh")

# example including environment arguments for pkgdown build step
cr_buildstep_pkgdown("$_GITHUB_REPO",
                     git_email = "cloudbuild@google.com",
                     secret = "github-ssh",
                     env = c("MYVAR=$_MY_VAR", "PROJECT=$PROJECT_ID"))
```

---

cr_buildstep_r *Run an R script in a Cloud Build R step*

Description

Helper to run R code within build steps, from either an existing local R file or within the source of the build.

Usage

```
cr_buildstep_r(
  r,
  name = "r-base",
  r_source = c("local", "runtime"),
  prefix = "rocker/",
  ...
)
```

Arguments

r	R code to run or a file containing R code ending with .R, or the gs:// location on Cloud Storage of the R file you want to run
name	The docker image that will run the R code, usually from rocker-project.org
r_source	Whether the R code will be from a runtime file within the source or at build time copying over from a local R file in your session
prefix	prefixed to name - set to "" to suppress. Will be suppressed if name starts with gcr.io
...	Other arguments passed to cr_buildstep

Details

If r_source="runtime" then r should be the location of that file within the source or image that will be run by the R code from image

If r_source="local" then it will copy over from a character string or local file into the build step directly.

If the R code location starts with gs:// then an extra buildstep will be added that will download the R script from that location then run it as per r_source="runtime". This will consequently override your setting of r_source

See Also

Other Cloud Buildsteps: cr_buildstep_bash(), cr_buildstep_decrypt(), cr_buildstep_df(), cr_buildstep_docker(), cr_buildstep_edit(), cr_buildstep_extract(), cr_buildstep_gcloud(), cr_buildstep_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_secret(), cr_buildstep_slack(), cr_buildstep()

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")

# create an R buildstep inline
cr_buildstep_r(c("paste('1+1=', 1+1)", "sessionInfo()"))

## Not run:

# create an R buildstep from a local file
```

```
cr_buildstep_r("my-r-file.R")

# create an R buildstep from a file within the source of the Build
cr_buildstep_r("inst/schedule/schedule.R", r_source = "runtime")


## End(Not run)

# use a different Rocker image e.g. rocker/verse
cr_buildstep_r(c("library(dplyr)",
                 "mtcars %>% select(mpg)",
                 "sessionInfo()"),
               name = "verse")

# use your own R image with custom R
my_r <- c("devtools::install()", "pkgdown::build_site()")
br <-  cr_buildstep_r(my_r, name= "gcr.io/gcer-public/packagetools:latest")
```

`cr_buildstep_run` *Create buildsteps to deploy to Cloud Run*

Description

Create buildsteps to deploy to Cloud Run

Usage

```
cr_buildstep_run(
  name,
  image,
  allowUnauthenticated = TRUE,
  region = cr_region_get(),
  concurrency = 80,
  port = NULL,
  max_instances = "default",
  memory = "256Mi",
  cpu = 1,
  env_vars = NULL,
  ...
)
```

Arguments

`name`	Name for deployment on Cloud Run
`image`	The name of the image to create or use in deployment - `gcr.io`

allowUnauthenticated	TRUE if can be reached from public HTTP address.
region	The endpoint region for deployment
concurrency	How many connections each container instance can serve. Can be up to 80.
port	Container port to receive requests at. Also sets the $PORT environment variable. Must be a number between 1 and 65535, inclusive. To unset this field, pass the special value "default".
max_instances	the desired maximum nuimber of container instances. "default" is 1000, you can get more if you requested a quota instance. For Shiny instances on Cloud Run, this needs to be 1.
memory	The format for size is a fixed or floating point number followed by a unit: G, M, or K corresponding to gigabyte, megabyte, or kilobyte, respectively, or use the power-of-two equivalents: Gi, Mi, Ki corresponding to gibibyte, mebibyte or kibibyte respectively. The default is 256Mi
cpu	1 or 2 CPUs for your instance
env_vars	Environment arguments passed to the Cloud Run container at runtime. Distinct from env that run at build time.
...	passed on to cr_buildstep

See Also

Other Cloud Buildsteps: cr_buildstep_bash(), cr_buildstep_decrypt(), cr_buildstep_df(), cr_buildstep_docker(), cr_buildstep_edit(), cr_buildstep_extract(), cr_buildstep_gcloud(), cr_buildstep_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_r(), cr_buildstep_secret(), cr_buildstep_slack(), cr_buildstep()

---

cr_buildstep_secret *Create a buildstep for using Secret Manager*

Description

This is the preferred way to manage secrets, rather than cr_buildstep_decrypt, as it stores the encrypted file in the cloud rather than in your project workspace.

Usage

```
cr_buildstep_secret(secret, decrypted, version = "latest", ...)
```

Arguments

secret	The secret data name in Secret Manager
decrypted	The name of the file the secret will be decrypted into
version	The version of the secret
...	Other arguments sent to cr_buildstep_bash

Details

This is for downloading encrypted files from Google Secret Manager. You will need to add the Secret Accessor Cloud IAM role to the Cloud Build service account to use it. Once you have uploaded your secret file and named it, it is available for Cloud Build to use.

See Also

How to set up secrets using Secret Manager

Other Cloud Buildsteps: `cr_buildstep_bash()`, `cr_buildstep_decrypt()`, `cr_buildstep_df()`, `cr_buildstep_docker()`, `cr_buildstep_edit()`, `cr_buildstep_extract()`, `cr_buildstep_gcloud()`, `cr_buildstep_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_slack()`, `cr_buildstep()`

Examples

```
cr_buildstep_secret("my_secret", decrypted = "/workspace/secret.json")
```

---

cr_buildstep_slack *Send a Slack message to a channel from a Cloud Build step*

Description

This uses https://github.com/technosophos/slack-notify to send Slack messages

Usage

```
cr_buildstep_slack(
  message,
  title = "CloudBuild - $BUILD_ID",
  channel = NULL,
  username = "googleCloudRunnerBot",
  webhook = "$_SLACK_WEBHOOK",
  icon = NULL,
  colour = "#efefef"
)
```

Arguments

message	The body of the message
title	The title of the message
channel	The channel to send the message to (if omitted, use Slack-configured default)
username	The name of the sender of the message. Does not need to be a "real" username
webhook	The Slack webhook to send to
icon	A URL to an icon (squares between 512px and 2000px)
colour	The RGB colour for message formatting

Details

You will need to set up a Slack webhook first, via this Slack guide on using incoming webhooks.

Once set, the default is to set this webhook to a Build macro called _SLACK_WEBHOOK, or supply it to the webhook argument.

See Also

Other Cloud Buildsteps: cr_buildstep_bash(), cr_buildstep_decrypt(), cr_buildstep_df(), cr_buildstep_docker(), cr_buildstep_edit(), cr_buildstep_extract(), cr_buildstep_gcloud(), cr_buildstep_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_r(), cr_buildstep_secret(), cr_buildstep()

Examples

```
# send a message to googleAuthRverse Slack
webhook <-
 "https://hooks.slack.com/services/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"
cr_buildstep_slack("Hello Slack", webhook = webhook)

## Not run:

bs <- cr_build_yaml(steps = cr_buildstep_slack("Hello Slack"))

cr_build(bs, substitutions = list(`_SLACK_WEBHOOK` = webhook))


## End(Not run)
```

---

cr_buildtrigger *Creates a new 'BuildTrigger'.This API is experimental.*

Description

Creates a new 'BuildTrigger'.This API is experimental.

Usage

```
cr_buildtrigger(
  build,
  name,
  trigger,
  description = paste("cr_buildtrigger: ", Sys.time()),
  disabled = FALSE,
  substitutions = NULL,
  ignoredFiles = NULL,
  includedFiles = NULL,
  trigger_tags = NULL,
```

```
  projectId = cr_project_get()
)
```

Arguments

build	The build to trigger created via cr_build_make, or the file location of the cloudbuild.yaml within the trigger source
name	User assigned name of the trigger
trigger	The trigger source created via cr_buildtrigger_repo
description	Human-readable description of this trigger
disabled	If true, the trigger will never result in a build
substitutions	A named list of Build macro variables
ignoredFiles	ignored_files and included_files are file glob matches extended with support for "**".
includedFiles	If any of the files altered in the commit pass the ignored_files
trigger_tags	Tags for the buildtrigger listing
projectId	ID of the project for which to configure automatic builds

Details

Any source specified in the build will be overwritten to use the trigger as a source (GitHub or Cloud Source Repositories)

See Also

Other BuildTrigger functions: BuildTrigger(), GitHubEventsConfig(), cr_buildtrigger_copy(), cr_buildtrigger_delete(), cr_buildtrigger_edit(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_repo(), cr_buildtrigger_run()

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")
cloudbuild <- system.file("cloudbuild/cloudbuild.yaml",
                          package = "googleCloudRunner")
bb <- cr_build_make(cloudbuild)

# repo hosted on GitHub
gh_trigger <- cr_buildtrigger_repo("MarkEdmondson1234/googleCloudRunner")

# repo mirrored to Cloud Source Repositories
cs_trigger <- cr_buildtrigger_repo("github_markedmondson1234_googlecloudrunner",
                                   type = "cloud_source")

## Not run:
# build with in-line build code
cr_buildtrigger(bb, name = "bt-github-inline", trigger = gh_trigger)
```

```
# build with in-line build code using Cloud Source Repository
cr_buildtrigger(bb, name = "bt-github-inline", trigger = cs_trigger)

# build pointing to cloudbuild.yaml within the GitHub repo
cr_buildtrigger("inst/cloudbuild/cloudbuild.yaml",
                name = "bt-github-file", trigger = gh_trigger)

# build with repo mirror from file
cr_buildtrigger("inst/cloudbuild/cloudbuild.yaml",
                name = "bt-cs-file", trigger = cs_trigger)

## End(Not run)
```

`cr_buildtrigger_copy` *Copy a buildtrigger*

Description

This lets you use the response from cr_buildtrigger_get for an existing buildtrigger to copy over settings to a new buildtrigger.

Usage

```
cr_buildtrigger_copy(
  buildtrigger,
  filename = NULL,
  name = NULL,
  tags = NULL,
  build = NULL,
  ignoredFiles = NULL,
  github = NULL,
  substitutions = NULL,
  includedFiles = NULL,
  disabled = NULL,
  triggerTemplate = NULL,
  projectId = cr_project_get()
)
```

Arguments

`buildtrigger`	A `CloudBuildTriggerResponse` object from cr_buildtrigger_get
`filename`	Path, from the source root, to a file whose contents is used for the build
`name`	User assigned name of the trigger
`tags`	Tags for annotation of a 'BuildTrigger'
`build`	Contents of the build template
`ignoredFiles`	ignored_files and included_files are file glob matches extended with support for "**".

`github`	a GitHubEventsConfig object - mutually exclusive with `triggerTemplate`
`substitutions`	A named list of Build macro variables
`includedFiles`	If any of the files altered in the commit pass the ignored_files
`disabled`	If true, the trigger will never result in a build
`triggerTemplate`	a RepoSource object - mutually exclusive with `github`
`projectId`	The projectId you are copying to

Details

Overwrite settings for the build trigger you are copying by supplying it as one of the other arguments from BuildTrigger.

See Also

Other BuildTrigger functions: `BuildTrigger()`, `GitHubEventsConfig()`, `cr_buildtrigger_delete()`, `cr_buildtrigger_edit()`, `cr_buildtrigger_get()`, `cr_buildtrigger_list()`, `cr_buildtrigger_repo()`, `cr_buildtrigger_run()`, `cr_buildtrigger()`

Examples

```
## Not run:
#copying a GitHub buildtrigger across projects and git repos
bt <- cr_buildtrigger_get("my-trigger", projectId = "my-project-1")

# a new GitHub project
gh <- GitHubEventsConfig("username/new-repo",
                         event = "push",
                         branch = "^master$")

# give 'Cloud Build Editor' role to your service auth key in new project
# then copy configuration across
cr_buildtrigger_copy(bt, github = gh, projectId = "my-new-project")



## End(Not run)
```

---

cr_buildtrigger_delete

Deletes a 'BuildTrigger' by its project ID and trigger ID.This API is experimental.

Description

Deletes a 'BuildTrigger' by its project ID and trigger ID.This API is experimental.

Usage

```
cr_buildtrigger_delete(triggerId, projectId = cr_project_get())
```

Arguments

triggerId	ID of the 'BuildTrigger' to get or a BuildTriggerResponse object
projectId	ID of the project that owns the trigger

See Also

Other BuildTrigger functions: BuildTrigger(), GitHubEventsConfig(), cr_buildtrigger_copy(), cr_buildtrigger_edit(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_repo(), cr_buildtrigger_run(), cr_buildtrigger()

---

cr_buildtrigger_edit *Updates a 'BuildTrigger' by its project ID and trigger ID.This API is experimental.*

Description

Seems not to work at the moment (issue #16)

Usage

```
cr_buildtrigger_edit(BuildTrigger, triggerId, projectId = cr_project_get())
```

Arguments

BuildTrigger	The BuildTrigger object to update to
triggerId	ID of the 'BuildTrigger' to edit or a previous BuildTriggerResponse object that will be edited
projectId	ID of the project that owns the trigger

See Also

Other BuildTrigger functions: BuildTrigger(), GitHubEventsConfig(), cr_buildtrigger_copy(), cr_buildtrigger_delete(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_repo(), cr_buildtrigger_run(), cr_buildtrigger()

Examples

```
## Not run:

github <- GitHubEventsConfig("MarkEdmondson1234/googleCloudRunner",
                             branch = "master")
bt2 <- cr_buildtrigger("trig2",
                       trigger = github,
                       build = "inst/cloudbuild/cloudbuild.yaml")
bt3 <- BuildTrigger(
  filename = "inst/cloudbuild/cloudbuild.yaml",
  name = "edited1",
  tags = "edit",
  github = github,
  disabled = TRUE,
  description = "edited trigger")

edited <- cr_buildtrigger_edit(bt3, triggerId = bt2)


## End(Not run)
```

`cr_buildtrigger_get` *Returns information about a 'BuildTrigger'.This API is experimental.*

Description

Returns information about a 'BuildTrigger'.This API is experimental.

Usage

```
cr_buildtrigger_get(triggerId, projectId = cr_project_get())
```

Arguments

`triggerId`	ID of the 'BuildTrigger' to get or a `BuildTriggerResponse` object
`projectId`	ID of the project that owns the trigger

See Also

Other BuildTrigger functions: `BuildTrigger()`, `GitHubEventsConfig()`, `cr_buildtrigger_copy()`, `cr_buildtrigger_delete()`, `cr_buildtrigger_edit()`, `cr_buildtrigger_list()`, `cr_buildtrigger_repo()`, `cr_buildtrigger_run()`, `cr_buildtrigger()`

cr_buildtrigger_list *Lists existing 'BuildTrigger's.This API is experimental.*

Description

Lists existing 'BuildTrigger's.This API is experimental.

Usage

```
cr_buildtrigger_list(projectId = cr_project_get())
```

Arguments

projectId	ID of the project for which to list BuildTriggers

See Also

Other BuildTrigger functions: BuildTrigger(), GitHubEventsConfig(), cr_buildtrigger_copy(), cr_buildtrigger_delete(), cr_buildtrigger_edit(), cr_buildtrigger_get(), cr_buildtrigger_repo(), cr_buildtrigger_run(), cr_buildtrigger()

cr_buildtrigger_repo *Create a buildtrigger repo object*

Description

Create a repository trigger object for use in build triggers

Usage

```
cr_buildtrigger_repo(
  repo_name,
  branch = ".*",
  tag = NULL,
  type = c("github", "cloud_source"),
  github_secret = NULL,
  ...
)
```

Arguments

repo_name	Either the GitHub username/repo_name or the Cloud Source repo_name
branch	Regex of the branches that will trigger a build. Ignore if tag is not NULL
tag	Regex of tags that will trigger a build
type	Whether trigger is GitHub or Cloud Source repoistory
github_secret	If you need to pull from a private GitHub repo, add the github secret from Google Secret Manager which will be used via cr_buildstep_secret
...	Other arguments passed to either GitHubEventsConfig or RepoSource

See Also

Other BuildTrigger functions: BuildTrigger(), GitHubEventsConfig(), cr_buildtrigger_copy(), cr_buildtrigger_delete(), cr_buildtrigger_edit(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_run(), cr_buildtrigger()

---

cr_buildtrigger_run *Runs a 'BuildTrigger' at a particular source revision.*

Description

Runs a 'BuildTrigger' at a particular source revision.

Usage

```
cr_buildtrigger_run(triggerId, RepoSource, projectId = cr_project_get())
```

Arguments

triggerId	ID of the 'BuildTrigger' to get or a BuildTriggerResponse object
RepoSource	The RepoSource object to pass to this method
projectId	ID of the project

See Also

Other BuildTrigger functions: BuildTrigger(), GitHubEventsConfig(), cr_buildtrigger_copy(), cr_buildtrigger_delete(), cr_buildtrigger_edit(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_repo(), cr_buildtrigger()

`cr_build_artifacts` *Download artifacts from a build*

Description

If a completed build includes artifact files this downloads them to local files

Usage

```
cr_build_artifacts(
  build,
  download_folder = getwd(),
  overwrite = FALSE,
  path_regex = NULL
)
```

Arguments

`build`	A Build object that includes the artifact location
`download_folder`	Where to download the artifact files
`overwrite`	Whether to overwrite existing local data
`path_regex`	A regex of files to fetch from the artifact bucket location. This is due to not being able to support the path globs

Details

If your artifacts are using file glob (e.g. `myfolder/**`) to decide which workspace files are uploaded to Cloud Storage, you will need to create a path_regex of similar functionality (`"^myfolder/"`). This is not needed if you use absolute path names such as `"myfile.csv"`

See Also

Storing images and artifacts

Other Cloud Build functions: `Build()`, `RepoSource()`, `Source()`, `StorageSource()`, `cr_build_make()`, `cr_build_status()`, `cr_build_upload_gcs()`, `cr_build_wait()`, `cr_build_write()`, `cr_build_yaml_artifact()`, `cr_build_yaml()`, `cr_build()`

Examples

```
r <- "write.csv(mtcars,file = 'artifact.csv')"
ba <- cr_build_yaml(
    steps = cr_buildstep_r(r),
    artifacts = cr_build_yaml_artifact('artifact.csv', bucket = "my-bucket")
    )
ba
```

```
## Not run:
build <- cr_build(ba)
built <- cr_build_wait(build)

cr_build_artifacts(built)

## End(Not run)
```

`cr_build_make` *Make a Cloud Build object out of a cloudbuild.yml file*

Description

This creates a Build object via the standard cloudbuild.yaml format

Usage

```
cr_build_make(
  yaml,
  source = NULL,
  timeout = NULL,
  images = NULL,
  artifacts = NULL,
  options = NULL,
  substitutions = NULL
)
```

Arguments

`yaml`	A `Yaml` object created from cr_build_yaml or a file location of a .yaml/.yml cloud build file
`source`	A Source object specifying the location of the source files to build, usually created by cr_build_source
`timeout`	Amount of time that this build should be allowed to run, to second
`images`	A list of images to be pushed upon the successful completion of all build
`artifacts`	Artifacts that may be built via cr_build_yaml_artifact
`options`	Options
`substitutions`	Substitutions data for 'Build' resource

See Also

https://cloud.google.com/cloud-build/docs/build-config

Other Cloud Build functions: `Build()`, `RepoSource()`, `Source()`, `StorageSource()`, `cr_build_artifacts()`, `cr_build_status()`, `cr_build_upload_gcs()`, `cr_build_wait()`, `cr_build_write()`, `cr_build_yaml_artifact()`, `cr_build_yaml()`, `cr_build()`

Examples

```
cloudbuild <- system.file("cloudbuild/cloudbuild.yaml",
                          package = "googleCloudRunner")
cr_build_make(cloudbuild)
```

`cr_build_schedule_http` *Create a Cloud Scheduler HTTP target from a Cloud Build object*

Description

This enables Cloud Scheduler to trigger Cloud Builds

Usage

```
cr_build_schedule_http(
  build,
  email = cr_email_get(),
  projectId = cr_project_get()
)
```

Arguments

`build`	A Build object created via cr_build_make or cr_build
`email`	The email that will authenticate the job set via cr_email_set
`projectId`	The projectId

Details

Ensure you have a service email with cr_email_set of format `service-{project-number}@gcp-sa-cloudscheduler.iam.g` with Cloud Scheduler Service Agent role as per https://cloud.google.com/scheduler/docs/http-target-auth#add

Value

A HttpTarget object for use in cr_schedule

See Also

https://cloud.google.com/cloud-build/docs/api/reference/rest/v1/projects.builds/create

Other Cloud Scheduler functions: `HttpTarget()`, `Job()`, `cr_schedule_delete()`, `cr_schedule_get()`, `cr_schedule_list()`, `cr_schedule_pause()`, `cr_schedule_run()`, `cr_schedule()`

Examples

```
cloudbuild <- system.file("cloudbuild/cloudbuild.yaml", package = "googleCloudRunner")
build1 <- cr_build_make(cloudbuild)
build1

## Not run:
cr_schedule("cloud-build-test1", schedule="15 5 * * *",
            httpTarget = cr_build_schedule_http(build1))

# a cloud build you would like to schedule
itworks <- cr_build("cloudbuild.yaml", launch_browser = FALSE)

# once working, pass in the build to the scheduler
cr_schedule("itworks-schedule", schedule = "15 5 * * *",
            httpTarget = cr_build_schedule_http(itworks))


## End(Not run)
```

---

cr_build_source *Build a source object*

Description

Build a source object

Usage

```
cr_build_source(x)

## S3 method for class 'gar_RepoSource'
cr_build_source(x)

## S3 method for class 'gar_StorageSource'
cr_build_source(x)
```

Arguments

x	A RepoSource or a StorageSource object

Examples

```
repo <- RepoSource("my_repo", branchName = "master")
gcs  <- StorageSource("my_code.tar.gz","gs://my-bucket")

cr_build_source(repo)
cr_build_source(gcs)
```

```
my_gcs_source <- cr_build_source(gcs)
my_repo_source <- cr_build_source(repo)

## Not run:

build1 <- cr_build("cloudbuild.yaml", source = my_gcs_source)
build2 <- cr_build("cloudbuild.yaml", source = my_repo_source)


## End(Not run)
```

`cr_build_status` *Returns information about a previously requested build.*

Description

The 'Build' that is returned includes its status (such as 'SUCCESS','FAILURE', or 'WORKING'), and timing information.

Usage

```
cr_build_status(id = .Last.value, projectId = cr_project_get())
```

Arguments

`id`	ID of the build or a `BuildOperationMetadata` object
`projectId`	ID of the project

Value

A gar_Build object Build

See Also

https://cloud.google.com/cloud-build/docs/api/reference/rest/Shared.Types/Status

Other Cloud Build functions: `Build()`, `RepoSource()`, `Source()`, `StorageSource()`, `cr_build_artifacts()`, `cr_build_make()`, `cr_build_upload_gcs()`, `cr_build_wait()`, `cr_build_write()`, `cr_build_yaml_artifact()`, `cr_build_yaml()`, `cr_build()`

`cr_build_upload_gcs` *Create a StorageSource*

Description

This creates a StorageSource object after uploading to Google Cloud Storage

Usage

```
cr_build_upload_gcs(
  local,
  remote = paste0(local, format(Sys.time(), "%Y%m%d%H%M%S"), ".tar.gz"),
  bucket = cr_bucket_get(),
  predefinedAcl = "bucketOwnerFullControl",
  deploy_folder = "deploy"
)
```

Arguments

`local`	Local directory containing the Dockerfile etc. you wish to deploy
`remote`	The name of the folder in your bucket
`bucket`	The Google Cloud Storage bucket to upload to
`predefinedAcl`	The ACL rules for the object uploaded. Set to "bucketLevel" for buckets with bucket level access enabled
`deploy_folder`	Which folder to deploy from

Details

It copies the files into a folder call "deploy" in your working directory, then tars it for upload

Value

A Source object

See Also

Other Cloud Build functions: `Build()`, `RepoSource()`, `Source()`, `StorageSource()`, `cr_build_artifacts()`, `cr_build_make()`, `cr_build_status()`, `cr_build_wait()`, `cr_build_write()`, `cr_build_yaml_artifact()`, `cr_build_yaml()`, `cr_build()`

Examples

```
## Not run:
cr_project_set("my-project")
cr_bucket_set("my-bucket")
my_gcs_source <- cr_build_upload_gcs("my_folder")
```

```
build1 <- cr_build("cloudbuild.yaml", source = my_gcs_source)


## End(Not run)
```

`cr_build_wait` *Wait for a Build to run*

Description

This will repeatedly call cr_build_status whilst the status is STATUS_UNKNOWN, QUEUED or WORKING

Usage

```
cr_build_wait(op = .Last.value, projectId = cr_project_get())
```

Arguments

`op`	The operation build object to wait for
`projectId`	The projectId

Value

A gar_Build object Build

See Also

Other Cloud Build functions: `Build()`, `RepoSource()`, `Source()`, `StorageSource()`, `cr_build_artifacts()`, `cr_build_make()`, `cr_build_status()`, `cr_build_upload_gcs()`, `cr_build_write()`, `cr_build_yaml_artifact()`, `cr_build_yaml()`, `cr_build()`

`cr_build_write` *Write out a Build object to cloudbuild.yaml*

Description

Write out a Build object to cloudbuild.yaml

Usage

```
cr_build_write(x, file = "cloudbuild.yaml")
```

Arguments

`x`	A Build object perhaps created with cr_build_make or cr_build_yaml
`file`	Where to write the yaml file

See Also

Other Cloud Build functions: Build(), RepoSource(), Source(), StorageSource(), cr_build_artifacts(), cr_build_make(), cr_build_status(), cr_build_upload_gcs(), cr_build_wait(), cr_build_yaml_artifact(), cr_build_yaml(), cr_build()

Examples

```
cr_project_set("my-project")
# write from creating a Yaml object
image = "gcr.io/my-project/my-image$BUILD_ID"
run_yaml <- cr_build_yaml(steps = c(
    cr_buildstep("docker", c("build","-t",image,".")),
    cr_buildstep("docker", c("push",image)),
    cr_buildstep("gcloud", c("beta","run","deploy", "test1", "--image", image))),
  images = image)

## Not run:
cr_build_write(run_yaml)

## End(Not run)

# write from a Build object
build <- cr_build_make(system.file("cloudbuild/cloudbuild.yaml",
                                   package = "googleCloudRunner"))

## Not run:
cr_build_write(build)

## End(Not run)
```

---

cr_build_yaml *Create a cloudbuild Yaml object in R*

Description

This can be written to disk or used directly with functions such as cr_build

Usage

```
cr_build_yaml(
  steps,
  timeout = NULL,
  logsBucket = NULL,
  options = NULL,
  substitutions = NULL,
  tags = NULL,
  secrets = NULL,
  images = NULL,
  artifacts = NULL
)
```

Arguments

steps	A vector of cr_buildstep
timeout	How long the entire build will run. If not set will be 10mins
logsBucket	Where logs are written. If you don't set this field, Cloud Build will use a default bucket to store your build logs.
options	A named list of options
substitutions	Build macros that will replace entries in other elements
tags	Tags for the build
secrets	A secrets object
images	What images will be build from this cloudbuild
artifacts	What artifacts may be built from this cloudbuild

See Also

Build configuration overview for cloudbuild.yaml

Other Cloud Build functions: Build(), RepoSource(), Source(), StorageSource(), cr_build_artifacts(), cr_build_make(), cr_build_status(), cr_build_upload_gcs(), cr_build_wait(), cr_build_write(), cr_build_yaml_artifact(), cr_build()

Examples

```
cr_project_set("my-project")
image <- "gcr.io/my-project/my-image"
cr_build_yaml(steps = c(
    cr_buildstep("docker", c("build","-t",image,".")),
    cr_buildstep("docker", c("push",image)),
    cr_buildstep("gcloud", c("beta","run","deploy", "test1", "--image", image))),
  images = image)
```

---

cr_build_yaml_artifact

Add an artifact for cloudbuild.yaml

Description

Add artifact objects to a build

Usage

```
cr_build_yaml_artifact(paths, bucket_dir = NULL, bucket = cr_bucket_get())
```

Arguments

paths	Which files from the working directory to upload to cloud storage once the build is finished. Can use globs but see details of cr_build_artifacts on how that affects downloads
bucket_dir	The directory in the bucket the files will be uploaded to
bucket	the bucket to send to

See Also

Other Cloud Build functions: Build(), RepoSource(), Source(), StorageSource(), cr_build_artifacts(), cr_build_make(), cr_build_status(), cr_build_upload_gcs(), cr_build_wait(), cr_build_write(), cr_build_yaml(), cr_build()

Examples

```
cr_project_set("my-project")
r <- "write.csv(mtcars,file = 'artifact.csv')"
cr_build_yaml(
  steps = cr_buildstep_r(r),
  artifacts = cr_build_yaml_artifact('artifact.csv', bucket = "my-bucket")
  )
```

---

cr_deploy_badger *Deploy a Cloud Run app to display build badges*

Description

This uses https://github.com/kelseyhightower/badger to create badges you can display in README.md etc. showing the current status of a Cloud Build

Usage

```
cr_deploy_badger(
  badger_image = "gcr.io/hightowerlabs/badger:0.0.1",
  json = Sys.getenv("GAR_CLIENT_JSON"),
  region = cr_region_get()
)
```

Arguments

badger_image	The docker image from the badger project to use
json	The clientId JSON file of the project to create within
region	The Cloud Run region

`cr_deploy_docker` *Deploy a local Dockerfile to be built on ContainerRegistry*

Description

Build a local Dockerfile in the cloud. See googleCloudRunner website for help how to generate Dockerfiles. If you want the docker to build on each commit, see also cr_deploy_docker_trigger

Usage

```
cr_deploy_docker(
  local,
  image_name = remote,
  dockerfile = NULL,
  remote = basename(local),
  tag = c("latest", "$BUILD_ID"),
  timeout = 600L,
  bucket = cr_bucket_get(),
  projectId = cr_project_get(),
  launch_browser = interactive(),
  kaniko_cache = TRUE,
  predefinedAcl = "bucketOwnerFullControl",
  ...
)
```

Arguments

`local`	The folder containing the Dockerfile to build
`image_name`	The name of the docker image to be built either full name starting with gcr.io or constructed from the image_name and projectId via `gcr.io/{projectId}/{image_name}`
`dockerfile`	An optional Dockerfile built to support the script. Not needed if 'Dockerfile' exists in folder. If supplied will be copied into deployment folder and called "Dockerfile"
`remote`	The folder on Google Cloud Storage
`tag`	The tag or tags to be attached to the pushed image - can use `Build` macros
`timeout`	Amount of time that this build should be allowed to run, to second
`bucket`	The GCS bucket that will be used to deploy code source
`projectId`	The projectId
`launch_browser`	Whether to launch the logs URL in a browser once deployed
`kaniko_cache`	If TRUE will use kaniko cache for Docker builds.
`predefinedAcl`	Access setting for the bucket used in deployed. Set to "bucketLevel" if using bucket level access
`...`	Arguments passed on to `cr_buildstep_docker`

image The image tag that will be pushed, starting with gcr.io or created by combining with projectId if not starting with gcr.io

location Where the Dockerfile to build is in relation to dir

Details

This lets you deploy local folders with Dockerfiles, automating saving the source on Google Cloud Storage.

To deploy builds on git triggers and sources such as GitHub, see the examples of cr_buildstep_docker or the use cases on the website

See Also

If you want the docker to build on each commit, see cr_deploy_docker_trigger

Other Deployment functions: cr_deploy_docker_trigger(), cr_deploy_packagetests(), cr_deploy_pkgdown(), cr_deploy_run_website(), cr_deploy_run(), cr_deploy_r()

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_email_set("123456@projectid.iam.gserviceaccount.com")
cr_bucket_set("my-bucket")

b <- cr_deploy_docker(system.file("example/", package="googleCloudRunner"))


## End(Not run)
```

---

cr_deploy_docker_trigger

Deploy Docker build from a Git repo

Description

This helps the common use case of building a Dockerfile based on the contents of a GitHub repo, and sets up a build trigger so it will build on every commit.

Usage

```
cr_deploy_docker_trigger(
  repo,
  image,
  trigger_name = paste0("docker-", image),
  image_tag = c("latest", "$SHORT_SHA", "$BRANCH_NAME"),
  ...,
```

```
  substitutions = NULL,
  ignoredFiles = NULL,
  includedFiles = NULL,
  timeout = NULL,
  projectId = cr_project_get()
)
```

Arguments

repo	The git repo holding the Dockerfile from cr_buildtrigger_repo
image	The name of the image you want to build
trigger_name	The trigger name
image_tag	What to tag the build docker image
...	Arguments passed on to cr_buildstep_docker tag The tag or tags to be attached to the pushed image - can use Build macros location Where the Dockerfile to build is in relation to dir dockerfile Specify the name of the Dockerfile found at location kaniko_cache If TRUE will use kaniko cache for Docker builds.
substitutions	A named list of Build macro variables
ignoredFiles	ignored_files and included_files are file glob matches extended with support for "**".
includedFiles	If any of the files altered in the commit pass the ignored_files
timeout	Amount of time that this build should be allowed to run, to second
projectId	The project to build under

Details

This creates a buildtrigger to do a kamiko cache enabled Docker build upon each commit, as defined by your repo settings via cr_buildtrigger_repo. It will build all tags concurrently.

See Also

cr_deploy_docker which lets you build Dockerfiles for more generic use cases

Other Deployment functions: cr_deploy_docker(), cr_deploy_packagetests(), cr_deploy_pkgdown(), cr_deploy_run_website(), cr_deploy_run(), cr_deploy_r()

Examples

```
## Not run:
repo <- cr_buildtrigger_repo("MarkEdmondson1234/googleCloudRunner")
cr_deploy_docker_trigger(repo, "test", dir = "cloud_build")

## End(Not run)
```

cr_deploy_gadget *Launch the googleCloudRunner deployment RStudio gadget*

Description

You can assign a hotkey to the addin via Tools > Addins > Browse Addins > Keyboard shortcuts. CTRL+SHIFT+D is a suggested hotkey.

Usage

```
cr_deploy_gadget()
```

cr_deploy_packagetests *Deploy a cloudbuild.yml for R package tests and upload to Codecov*

Description

This tests an R package each time you commit, and uploads the test coverage results to Codecov

Usage

```
cr_deploy_packagetests(
  steps = NULL,
  cloudbuild_file = "cloudbuild-tests.yml",
  env = c("NOT_CRAN=true"),
  test_script = NULL,
  codecov_script = NULL,
  codecov_token = "$_CODECOV",
  build_image = "gcr.io/gcer-public/packagetools:latest",
  create_trigger = c("file", "inline", "no"),
  trigger_repo = NULL,
  ...
)
```

Arguments

steps	extra steps to run before the cr_buildstep_packagetests steps run (such as decryption of auth files)
cloudbuild_file	The cloudbuild yaml file to write to. See create_trigger
env	Environment arguments to be set during the test script runs
test_script	The script that will perform tests. If NULL a default script is used in system.file("r_buildsteps","dev
codecov_script	The script that will perform coverage. If NULL a default script is used in system.file("r_buildsteps",

codecov_token	If using codecov, supply your codecov token here.
build_image	The docker image that will be used to run the R code for the test scripts
create_trigger	If creating a trigger, whether to create it from the cloudbuild_file or inline
trigger_repo	If not NULL, a cr_buildtrigger_repo where a buildtrigger will be created via cr_buildtrigger
...	Arguments passed on to cr_build_make
	yaml A Yaml object created from cr_build_yaml or a file location of a .yaml/.yml cloud build file
	artifacts Artifacts that may be built via cr_build_yaml_artifact
	options Options
	source A Source object specifying the location of the source files to build, usually created by cr_build_source
	timeout Amount of time that this build should be allowed to run, to second
	images A list of images to be pushed upon the successful completion of all build
	substitutions Substitutions data for 'Build' resource

Details

The trigger repository needs to hold an R package configured to do tests upon.

For GitHub, the repository will need to be linked to the project you are building within, via https://console.cloud.google.com/cloud-build/triggers/connect

If your tests need authentication details, add these via cr_buildstep_secret to the steps argument, which will prepend decrypting the authentication file before running the tests.

If you want codecov to ignore some files then also deploy a .covrignore file to your repository - see covr website at https://covr.r-lib.org/ for details.

See Also

Create your own custom deployment using cr_buildstep_packagetests which this function uses with some defaults

Other Deployment functions: cr_deploy_docker_trigger(), cr_deploy_docker(), cr_deploy_pkgdown(), cr_deploy_run_website(), cr_deploy_run(), cr_deploy_r()

Examples

```
# create a local cloudbuild.yml file for packagetests
pd <- cr_deploy_packagetests(create_trigger = "no")
pd

# add a decryption step for an auth file
cr_deploy_packagetests(
  steps = cr_buildstep_secret("my_secret", "auth.json"),
  env = c("NOT_CRAN=true", "MY_AUTH_FILE=auth.json"),
  timeout = 1200,
```

```
  create_trigger = "no"
)


# creating a buildtrigger repo for trigger_repo
repo <- cr_buildtrigger_repo("MarkEdmondson1234/googleCloudRunner",
                             branch = "master")

## Not run:

# will create the file in the repo, and point a buildtrigger at it
cr_deploy_packagetests(create_trigger = "file", trigger_repo = repo)


# will make an inline build within a buildtrigger
cr_deploy_packagetests(create_trigger = "inline", trigger_repo = repo)

## End(Not run)

unlink("cloudbuild-tests.yml")
```

---

cr_deploy_pkgdown *Deploy a cloudbuild.yml for a pkgdown website of an R package*

Description

This builds a pkgdown website each time the trigger fires and deploys it to git

Usage

```
cr_deploy_pkgdown(
  github_repo,
  secret,
  steps = NULL,
  create_trigger = c("file", "inline", "no"),
  cloudbuild_file = "cloudbuild-pkgdown.yml",
  git_email = "googlecloudrunner@r.com",
  env = NULL,
  build_image = "gcr.io/gcer-public/packagetools:latest",
  post_setup = NULL,
  post_clone = NULL
)
```

Arguments

github_repo	The GitHub repo to deploy pkgdown website from and to.
secret	The name of the secret on Google Secret Manager for the git ssh private key

steps	extra steps to run before the pkgdown website steps run
create_trigger	If not "no" then the buildtrigger will be setup for you via cr_buildtrigger, if "file" will create a buildtrigger pointing at cloudbuild_file, if "inline" will put the build inline within the trigger (no file created)
cloudbuild_file	The cloudbuild yaml file to write to
git_email	The email the git commands will be identifying as
env	A character vector of env arguments to set for all steps
build_image	A docker image with pkgdown installed
post_setup	Steps that occur after git setup
post_clone	A cr_buildstep that occurs after the repo is cloned

Details

The trigger repository needs to hold an R package configured to build a pkgdown website.

For GitHub, the repository will also need to be linked to the project you are building within, via https://console.cloud.google.com/cloud-build/triggers/connect

The git ssh keys need to be deployed to Google Secret Manager for the deployment of the website - see cr_buildstep_git - this only needs to be done once per Git account.

See Also

Create your own custom deployment using cr_buildstep_pkgdown which this function uses with some defaults.

Other Deployment functions: cr_deploy_docker_trigger(), cr_deploy_docker(), cr_deploy_packagetests(), cr_deploy_run_website(), cr_deploy_run(), cr_deploy_r()

Examples

```
pd <- cr_deploy_pkgdown("MarkEdmondson1234/googleCloudRunner",
                        secret = "my_git_secret",
                        create_trigger = "no")
pd
file.exists("cloudbuild-pkgdown.yml")
unlink("cloudbuild-pkgdown.yml")

## Not run:
cr_deploy_pkgdown("MarkEdmondson1234/googleCloudRunner",
                  secret = "my_git_secret",
                  create_trigger = "inline")


## End(Not run)
```

`cr_deploy_r` *Deploy an R script with an optional schedule*

Description

Will create a build to run an R script in Cloud Build with an optional schedule from Cloud Scheduler

Usage

```
cr_deploy_r(
  r,
  schedule = NULL,
  source = NULL,
  run_name = NULL,
  r_image = "rocker/verse",
  pre_steps = NULL,
  post_steps = NULL,
  timeout = 600L,
  ...,
  email = cr_email_get(),
  region = cr_region_get(),
  projectId = cr_project_get(),
  launch_browser = interactive()
)
```

Arguments

`r`	R code to run or a file containing R code ending with .R, or the gs:// location on Cloud Storage of the R file you want to run
`schedule`	A cron schedule e.g. `"15 5 * * *"`
`source`	A Source object specifying the location of the source files to build, usually created by cr_build_source
`run_name`	What name the R code will identify itself as. If `NULL` one is autogenerated.
`r_image`	The R docker environment executing the R code
`pre_steps`	Other cr_buildstep to run before the R code executes
`post_steps`	Other cr_buildstep to run after the R code executes
`timeout`	Amount of time that this build should be allowed to run, to second
`...`	Arguments passed on to `cr_buildstep_r` `name` The docker image that will run the R code, usually from rocker-project.org `r_source` Whether the R code will be from a runtime file within the source or at build time copying over from a local R file in your session `prefix` prefixed to name - set to "" to suppress. Will be suppressed if `name` starts with gcr.io
`email`	The email that will authenticate the job set via cr_email_set

region	The region usually set with cr_region_set
projectId	ID of the project
launch_browser	Whether to launch the logs URL in a browser once deployed

Details

If schedule=NULL then the R script will be run immediately on Cloud Build via cr_build.

If schedule carries a cron job string (e.g. "15 5 * * *") then the build will be scheduled via Cloud Scheduler to run as described in cr_build_schedule_http

The R script will execute within the root directory of which Source you supply, usually created via cr_build_source. Bear in mind if the source changes then the code scheduled may need updating.

The r_image dictates what R libraries the R environment executing the code of r will have, via the underlying Docker container usually supplied by rocker-project.org. If you want custom R libraries beyond the default, create a docker container with those R libraries installed (perhaps via cr_deploy_docker)

Value

If scheduling then a Job, if building immediately then a Build

See Also

If you want to run R code upon certain events like GitHub pushes, look at cr_buildtrigger

Other Deployment functions: cr_deploy_docker_trigger(), cr_deploy_docker(), cr_deploy_packagetests(), cr_deploy_pkgdown(), cr_deploy_run_website(), cr_deploy_run()

Examples

```
r_lines <- c("list.files()",
             "library(dplyr)",
             "mtcars %>% select(mpg)",
             "sessionInfo()")
source <- cr_build_source(RepoSource("googleCloudStorageR",
                                     branchName = "master"))

## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_email_set("123456@projectid.iam.gserviceaccount.com")

# check the script runs ok
cr_deploy_r(r_lines, source = source)

# schedule the script
cr_deploy_r(r_lines, schedule = "15 21 * * *", source = source)

## End(Not run)
```

cr_deploy_run *Deploy to Cloud Run*

Description

Deploy R api plumber scripts, HTML files or other images create the Docker image, add the build to Cloud Build and deploy to Cloud Run

Usage

```
cr_deploy_run(
  local,
  remote = basename(local),
  dockerfile = NULL,
  image_name = remote,
  tag = "$BUILD_ID",
  region = cr_region_get(),
  bucket = cr_bucket_get(),
  projectId = cr_project_get(),
  launch_browser = interactive(),
  timeout = 600L,
  ...
)

cr_deploy_html(
  html_folder,
  remote = basename(html_folder),
  image_name = remote,
  tag = "$BUILD_ID",
  region = cr_region_get(),
  bucket = cr_bucket_get(),
  projectId = cr_project_get(),
  launch_browser = interactive(),
  timeout = 600L
)

cr_deploy_plumber(
  api,
  remote = basename(api),
  dockerfile = NULL,
  image_name = remote,
  tag = "$BUILD_ID",
  region = cr_region_get(),
  bucket = cr_bucket_get(),
  projectId = cr_project_get(),
  launch_browser = interactive(),
  timeout = 600L
```

)

Arguments

local	A folder containing the scripts and Dockerfile to deploy to Cloud Run
remote	The folder on Google Cloud Storage, and the name of the service on Cloud Run
dockerfile	An optional Dockerfile built to support the script. Not needed if 'Dockerfile' exists in folder. If supplied will be copied into deployment folder and called "Dockerfile"
image_name	The gcr.io image name that will be deployed and/or built
tag	The tag or tags to be attached to the pushed image - can use Build macros
region	The Cloud Run endpoint set by CR_REGION env arg
bucket	The Cloud Storage bucket that will hold the code
projectId	The projectId where it all gets deployed to
launch_browser	Whether to launch the logs URL in a browser once deployed
timeout	Amount of time that this build should be allowed to run, to second
...	Arguments passed on to cr_buildstep_run name Name for deployment on Cloud Run image The name of the image to create or use in deployment - gcr.io allowUnauthenticated TRUE if can be reached from public HTTP address. concurrency How many connections each container instance can serve. Can be up to 80. port Container port to receive requests at. Also sets the $PORT environment variable. Must be a number between 1 and 65535, inclusive. To unset this field, pass the special value "default". max_instances the desired maximum nuimber of container instances. "default" is 1000, you can get more if you requested a quota instance. For Shiny instances on Cloud Run, this needs to be 1. memory The format for size is a fixed or floating point number followed by a unit: G, M, or K corresponding to gigabyte, megabyte, or kilobyte, respectively, or use the power-of-two equivalents: Gi, Mi, Ki corresponding to gibibyte, mebibyte or kibibyte respectively. The default is 256Mi cpu 1 or 2 CPUs for your instance env_vars Environment arguments passed to the Cloud Run container at runtime. Distinct from env that run at build time.
html_folder	the folder containing all the html
api	A folder containing the R script using plumber called api.R and all its dependencies

Details

These deploy containers to Cloud Run, a scale 0-to-millions container-as-a-service on Google Cloud Platform.

cr_deploy_html

Deploy html files to a nginx server on Cloud Run.

Supply the html folder to host it on Cloud Run. Builds the dockerfile with the html within it, then deploys to Cloud Run

Will add a `default.template` file to the html folder that holds the nginx configuration

cr_deploy_plumber

The entrypoint for CloudRun will be via a plumber script called api.R - this should be included in your local folder to deploy. From that api.R you can source or call other resources in the same folder, using relative paths.

The function will create a local folder called "deploy" and a tar.gz of that folder which is what is being uploaded to Google Cloud Storage

See Also

cr_deploy_run_website which has more features like rending Rmd files and deploying upon each git commit

Other Deployment functions: `cr_deploy_docker_trigger()`, `cr_deploy_docker()`, `cr_deploy_packagetests()`, `cr_deploy_pkgdown()`, `cr_deploy_run_website()`, `cr_deploy_r()`

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_bucket_set("my-bucket")
cr_deploy_run(system.file("example/", package = "googleCloudRunner"))


## End(Not run)

## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_bucket_set("my-bucket")

cr_deploy_html("my_folder")


## End(Not run)

## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_bucket_set("my-bucket")

cr_deploy_plumber(system.file("example/", package = "googleCloudRunner"))
```

```
## End(Not run)
```

`cr_deploy_run_website` *Deploy HTML built from a repo each commit*

Description

This lets you set up triggers that will update an R generated website each commit.

Usage

```
cr_deploy_run_website(
  repo,
  image = paste0("website-", format(Sys.Date(), "%Y%m%d")),
  rmd_folder = NULL,
  html_folder = NULL,
  image_tag = "$SHORT_SHA",
  timeout = 600L,
  edit_r = NULL,
  r_image = "gcr.io/gcer-public/packagetools:latest",
  allowUnauthenticated = TRUE,
  region = cr_region_get(),
  projectId = cr_project_get()
)
```

Arguments

`repo`	A git repository defined in cr_buildtrigger_repo
`image`	The name of the image you want to build
`rmd_folder`	A folder of Rmd files within GitHub source that will be built into HTML for serving via render
`html_folder`	A folder of html to deploy within GitHub source. Will be ignored if rmd_folder is not NULL
`image_tag`	What to tag the build docker image
`timeout`	Timeout for the build
`edit_r`	If you want to change the R code to render the HTML, supply R code via a file or string of R as per cr_buildstep_r
`r_image`	The image that will run the R code from `edit_r`
`allowUnauthenticated`	TRUE if can be reached from public HTTP address.
`region`	The region for cloud run
`projectId`	The project to build under

Details

This lets you render the Rmd (or other R functions that produce HTML) in a folder for your repo, which will then be hosted on a Cloud Run enabled with nginx. Each time you push to git with modified Rmd code, it will build the new HTML and push an update to the website.

This default R code is rendered in the rmd_folder:

```
lapply(list.files('.',pattern = '.Rmd',full.names = TRUE),rmarkdown::render,output_format
= 'html_document')
```

See Also

cr_deploy_html that lets you deploy just HTML files and cr_deploy_pkgdown for running pkgdown websites.

Other Deployment functions: cr_deploy_docker_trigger(), cr_deploy_docker(), cr_deploy_packagetests(), cr_deploy_pkgdown(), cr_deploy_run(), cr_deploy_r()

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
your_repo <- cr_buildtrigger_repo("MarkEdmondson1234/googleCloudRunner")
cr_deploy_run_website(your_repo, rmd_folder = "vignettes")

# change the Rmd rendering to pkgdown
r <- "devtools::install();pkgdown::build_site()"

cr_deploy_run_website(your_repo,
                      image = paste0(your_repo, "-pkgdown"),
                      rmd_folder = ".",
                      edit_r = r)


## End(Not run)
```

---

cr_email_get *Get/Set cloud build email*

Description

Needed so Cloud Scheduler can run Cloud Build jobs - can also set via environment argument CR_BUILD_EMAIL

Usage

```
cr_email_get()

cr_email_set(cloudbuildEmail)
```

Arguments

cloudbuildEmail	The Cloud Build service email

See Also

https://console.cloud.google.com/cloud-build/settings

Examples

```
cr_email_set("myemail@domain.com")
cr_email_get()
```

---

cr_jwt_create *Create a JSON Web Token (JWT) from your service client and call Google services*

Description

This can be used to call authenticated services such as Cloud Run.

Usage

```
cr_jwt_create(the_url, service_json = Sys.getenv("GCE_AUTH_FILE"))

cr_jwt_token(signed_jwt, the_url)

cr_jwt_with_httr(req, token)

cr_jwt_with_curl(h = curl::new_handle(), token)

cr_jwt_async(urls, token, ...)
```

Arguments

the_url	The URL of the service you want to call
service_json	The account service key JSON that will be used to generate the JWT
signed_jwt	A JWT created from cr_jwt_create
req	A httr request to the service running on the_url, using httr verbs such as GET
token	The token created via cr_jwt_token
h	A curl handle such as set with new_handle
urls	URLs to request asynchronously
...	Other arguments passed to new_handle

Details

For certain Google services a JWT is needed to authenticate access, which is distinct from OAuth2. An example of this is authenticated Cloud Run such as deployed when using cr_run and parameter `allowUnauthenticated = FALSE`. These functions help you call your services by generating the JWT from your service account key.

The token is set to expire in 1 hour, so it will need refreshing before then by calling this function again.

See Also

Service-to-service authentication on GCP

Other Cloud Run functions: `cr_plumber_pubsub()`, `cr_run_get()`, `cr_run_list()`, `cr_run()`

Examples

```
## Not run:

# The private authenticated access only Cloud Run service
the_url <- "https://authenticated-cloudrun-ewjogewawq-ew.a.run.app/"

# creating the JWT and token
jwt <- cr_jwt_create(the_url)
token <- cr_jwt_token(jwt, the_url)

# call Cloud Run app using token with any httr verb
library(httr)
res <- cr_jwt_with_httr(
  GET("https://authenticated-cloudrun-ewjogewawq-ew.a.run.app/hello"),
  token)
content(res)

# call Cloud Run app with curl - you can pass in a curl handle
library(curl)
h <- new_handle()
handle_setopt(h, customrequest = "PUT")
handle_setform(h, a = "1", b = "2")
h <- cr_jwt_with_curl(h, token = token)
r <- curl_fetch_memory("https://authenticated-cloudrun-ewjogewawq-ew.a.run.app/hello", h)
cat(rawToChar(r$content))

# use curls multi-asynch functions
many_urls <- paste0("https://authenticated-cloudrun-ewjogewawq-ew.a.run.app/hello",
                    paste0("?param="),1:6)
cr_jwt_async(many_urls, token = token)

## End(Not run)
```

`cr_plumber_pubsub` *Plumber - Pub/Sub parser*

Description

A function to use in plumber scripts to accept Pub/Sub messages

Usage

```
cr_plumber_pubsub(message = NULL, pass_f = function(x) x)
```

Arguments

`message`	The pubsub message
`pass_f`	An R function that will work with the data parsed out of the pubsub `message$data` field.

Details

This function is intended to be used within plumb API scripts. It needs to be annotated with a `@post` URL route and a `@param message The pubsub message` as per the plumber documentation.

`pass_f` should be a function you create that accepts one argument, the data from the pubsub `message$data` field. It is unencoded for you. Make sure the function returns a 200 response otherwise pub/sub will keep resending the message! `return(TRUE)` is adequate.

The Docker container for the API will need to include `googleCloudRunner` installed in its R environment to run this function. This is available in the public `gcr.io/gcer-public/cloudrunner` image.

Use cr_pubsub to test this function once deployed.

See Also

Google Pub/Sub tutorial for Cloud Run. You can set up Pub/Sub messages from Google Cloud Storage buckets via gcs_create_pubsub

Other Cloud Run functions: `cr_jwt_create()`, `cr_run_get()`, `cr_run_list()`, `cr_run()`

Examples

```
## Not run:

# within a plumber api.R script:

# example function echos back pubsub message
pub <- function(x){
  paste("Echo:", x)
}
```

```
#' Recieve pub/sub message
#' @post /pubsub
#' @param message a pub/sub message
function(message=NULL){
  googleCloudRunner::cr_plumber_pubsub(message, pub)
  }


## End(Not run)
```

`cr_project_set` *Get/Set the projectId for your CloudRun services*

Description

Can also use environment argument GCE_DEFAULT_PROJECT_ID

Usage

```
cr_project_set(projectId)

cr_project_get()
```

Arguments

`projectId`	The projectId

Examples

```
cr_project_get()
```

`cr_pubsub` *Send a message to pubsub*

Description

Useful for testing Cloud Run pubsub deployments

Usage

```
cr_pubsub(endpoint, payload = jsonlite::toJSON("hello"))
```

Arguments

`endpoint`	The url endpoint of the PubSub service
`payload`	Will be base64 encoded and placed in `message$data`

cr_region_set *Get/Set the endpoint for your CloudRun services*

Description

Can also use environment argument CR_REGION

Usage

```
cr_region_set(
  region = c("europe-west1", "us-central1", "asia-northeast1", "us-east1")
)

cr_region_get()
```

Arguments

region	Region for the endpoint

Examples

```
cr_region_get()
```

cr_run *Create a CloudRun service.*

Description

Deploys an existing gcr.io image.

Usage

```
cr_run(
  image,
  name = basename(image),
  allowUnauthenticated = TRUE,
  concurrency = 1,
  port = NULL,
  max_instances = "default",
  memory = "256Mi",
  cpu = 1,
  timeout = 600L,
  region = cr_region_get(),
  projectId = cr_project_get(),
  launch_browser = interactive(),
```

```
  env_vars = NULL,
  ...
)
```

Arguments

image	The name of the image to create or use in deployment - gcr.io
name	Name for deployment on Cloud Run
allowUnauthenticated	TRUE if can be reached from public HTTP address.
concurrency	How many connections each container instance can serve. Can be up to 80.
port	Container port to receive requests at. Also sets the $PORT environment variable. Must be a number between 1 and 65535, inclusive. To unset this field, pass the special value "default".
max_instances	the desired maximum nuimber of container instances. "default" is 1000, you can get more if you requested a quota instance. For Shiny instances on Cloud Run, this needs to be 1.
memory	The format for size is a fixed or floating point number followed by a unit: G, M, or K corresponding to gigabyte, megabyte, or kilobyte, respectively, or use the power-of-two equivalents: Gi, Mi, Ki corresponding to gibibyte, mebibyte or kibibyte respectively. The default is 256Mi
cpu	1 or 2 CPUs for your instance
timeout	Amount of time that this build should be allowed to run, to second
region	The endpoint region for deployment
projectId	The GCP project from which the services should be listed
launch_browser	Whether to launch the logs URL in a browser once deployed
env_vars	Environment arguments passed to the Cloud Run container at runtime. Distinct from env that run at build time.
...	Arguments passed on to cr_buildstep_run

Details

Uses Cloud Build to deploy an image to Cloud Run

See Also

Google Documentation for Cloud Run

Use cr_deploy_docker or similar to create image, cr_deploy_run to automate building and deploying, cr_deploy_plumber to deploy plumber APIs.

Deploying Cloud Run using Cloud Build

Other Cloud Run functions: cr_jwt_create(), cr_plumber_pubsub(), cr_run_get(), cr_run_list()

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_run("gcr.io/my-project/my-image")
cr_run("gcr.io/cloud-tagging-10302018/gtm-cloud-image:stable",
       env_vars = c("CONTAINER_CONFIG=xxxxxxx"))

## End(Not run)
```

`cr_run_get` *Get information about a Cloud Run service.*

Description

Get information about a Cloud Run service.

Usage

```
cr_run_get(name, projectId = cr_project_get())
```

Arguments

`name`	The name of the service to retrieve
`projectId`	The projectId to get from

Details

This returns details on a particular deployed Cloud Run service.

See Also

Google Documentation on namespaces.services.get

Other Cloud Run functions: `cr_jwt_create()`, `cr_plumber_pubsub()`, `cr_run_list()`, `cr_run()`

cr_run_list *List CloudRun services.*

Description

List the Cloud Run services you have access to

Usage

```
cr_run_list(
  projectId = cr_project_get(),
  labelSelector = NULL,
  limit = NULL,
  summary = TRUE
)
```

Arguments

projectId	The GCP project from which the services should be listed
labelSelector	Allows to filter resources based on a label
limit	The maximum number of records that should be returned
summary	If TRUE will return only a subset of info available, set to FALSE for all metadata

See Also

Google Documentation for Cloud Run

Other Cloud Run functions: cr_jwt_create(), cr_plumber_pubsub(), cr_run_get(), cr_run()

cr_schedule *Creates or updates a Cloud Scheduler job.*

Description

Creates or updates a Cloud Scheduler job.

Usage

```
cr_schedule(
  name,
  schedule = NULL,
  httpTarget = NULL,
  description = NULL,
  overwrite = FALSE,
  timeZone = Sys.timezone(),
  region = cr_region_get(),
  projectId = cr_project_get()
)
```

Arguments

name	Optionally caller-specified in CreateJob, after
schedule	A cron schedule e.g. "15 5 * * *"
httpTarget	A HTTP target object HttpTarget
description	Optionally caller-specified in CreateJob or
overwrite	If TRUE and an existing job with the same name exists, will overwrite it with the new parameters
timeZone	Specifies the time zone to be used in interpreting schedule. If set to NULL will be "UTC". Note that some time zones include a provision for daylight savings time.
region	The region usually set with cr_region_set
projectId	The GCP project to run within usually set with cr_project_set

Value

A gar_scheduleJob class object

See Also

Google Documentation for Cloud Scheduler

Other Cloud Scheduler functions: HttpTarget(), Job(), cr_build_schedule_http(), cr_schedule_delete(), cr_schedule_get(), cr_schedule_list(), cr_schedule_pause(), cr_schedule_run()

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_schedule("test",
      "* * * * *",
      httpTarget = HttpTarget(uri="https://code.markedmondson.me"))

# schedule a cloud build (no source)
build1 <- cr_build_make("cloudbuild.yaml")
cr_schedule("cloud-build-test", "15 5 * * *",
             httpTarget = cr_build_schedule_http(build1))

# schedule a cloud build with code source from GCS bucket
my_gcs_source <- cr_build_upload_gcs("my_folder", bucket = cr_get_bucket())
build <- cr_build_make("cloudbuild.yaml", source = my_gcs_source))
cr_schedule("cloud-build-test2", "15 5 * * *",
            httpTarget = cr_build_schedule_http(build))

# update a schedule with the same name - only supply what you want to change
cr_schedule("cloud-build-test2", "12 6 * * *", overwrite=TRUE)

# By default will use the timezone as specified by Sys.timezone() - change
```

```
# this by supplying it directly
cr_schedule("timzone-utc", "12 2 * * *", timeZone = "UTC")

## End(Not run)
```

`cr_schedule_delete` *Deletes a scheduled job.*

Description

Deletes a scheduled job.

Usage

```
cr_schedule_delete(x, region = cr_region_get(), projectId = cr_project_get())
```

Arguments

`x`	The name of the scheduled job or a Job object
`region`	The region to run within
`projectId`	The projectId

See Also

cloudscheduler.projects.locations.jobs.delete

Other Cloud Scheduler functions: `HttpTarget()`, `Job()`, `cr_build_schedule_http()`, `cr_schedule_get()`, `cr_schedule_list()`, `cr_schedule_pause()`, `cr_schedule_run()`, `cr_schedule()`

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_schedule_delete("cloud-build-test1")

## End(Not run)
```

cr_schedule_get *Gets a scheduler job.*

Description

Gets a scheduler job.

Usage

```
cr_schedule_get(name, region = cr_region_get(), projectId = cr_project_get())
```

Arguments

name	Required - a string or a schedule Job object
region	The region to run within
projectId	The projectId

See Also

Google Documentation

Other Cloud Scheduler functions: HttpTarget(), Job(), cr_build_schedule_http(), cr_schedule_delete(), cr_schedule_list(), cr_schedule_pause(), cr_schedule_run(), cr_schedule()

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_schedule_get("cloud-build-test1")

## End(Not run)
```

cr_schedule_list *Lists Cloud Scheduler jobs.*

Description

Lists cloud scheduler jobs including targeting, schedule and authentication

Usage

```
cr_schedule_list(region = cr_region_get(), projectId = cr_project_get())
```

Arguments

region	The region to run within
projectId	The projectId

See Also

Google Documentation

Other Cloud Scheduler functions: HttpTarget(), Job(), cr_build_schedule_http(), cr_schedule_delete(), cr_schedule_get(), cr_schedule_pause(), cr_schedule_run(), cr_schedule()

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_schedule_list()


## End(Not run)
```

---

cr_schedule_pause *Pauses and resumes a scheduled job.*

Description

If a job is paused then the system will stop executing the job until it is re-enabled via cr_schedule_resume.

Usage

```
cr_schedule_pause(x, region = cr_region_get(), projectId = cr_project_get())

cr_schedule_resume(x, region = cr_region_get(), projectId = cr_project_get())
```

Arguments

x	The name of the scheduled job or a Job object
region	The region to run within
projectId	The projectId

Details

The state of the job is stored in state; if paused it will be set to Job.State.PAUSED. A job must be in Job.State.ENABLED to be paused.

See Also

cloudscheduler.projects.locations.jobs.pause

cloudscheduler.projects.locations.jobs.resume

Other Cloud Scheduler functions: HttpTarget(), Job(), cr_build_schedule_http(), cr_schedule_delete(), cr_schedule_get(), cr_schedule_list(), cr_schedule_run(), cr_schedule()

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_schedule_pause("cloud-build-test1")
cr_schedule_resume("cloud-build-test1")

## End(Not run)
```

---

cr_schedule_run *Forces a job to run now.*

Description

When this method is called, Cloud Scheduler will dispatch the job, even if the job is already running.

Usage

```
cr_schedule_run(x, region = cr_region_get(), projectId = cr_project_get())
```

Arguments

x	The name of the scheduled job or a Job object
region	The region to run within
projectId	The projectId

See Also

cloudscheduler.projects.locations.jobs.run

Other Cloud Scheduler functions: HttpTarget(), Job(), cr_build_schedule_http(), cr_schedule_delete(), cr_schedule_get(), cr_schedule_list(), cr_schedule_pause(), cr_schedule()

Examples

```
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_schedule_run("cloud-build-test1")

## End(Not run)
```

`cr_setup` *A helper setup function for setting up use with googleCloudRunner*

Description

A helper setup function for setting up use with googleCloudRunner

Usage

```
cr_setup()
```

See Also

Other setup functions: `cr_setup_auth()`, `cr_setup_service()`, `cr_setup_test()`

`cr_setup_auth` *Create a service account for googleCloudRunner*

Description

This will use your Google OAuth2 user to create a suitable service account

Usage

```
cr_setup_auth(
  email = Sys.getenv("GARGLE_EMAIL"),
  file = "googlecloudrunner-auth-key.json",
  session_user = NULL
)
```

Arguments

`email`	What email to open OAuth2 with
`file`	Where to save the authentication file
`session_user`	1 for user level, 2 for project level, leave `NULL` to be prompted

Value

TRUE if the file is ready to be setup by cr_setup, FALSE if need to stop

See Also

Other setup functions: `cr_setup_service()`, `cr_setup_test()`, `cr_setup()`

`cr_setup_service` *Give a service account the right permissions for googleCloudRunner operations*

Description

Give a service account the right permissions for googleCloudRunner operations

Usage

```
cr_setup_service(
  account_email,
  roles = cr_setup_role_lookup("local"),
  json = Sys.getenv("GAR_CLIENT_JSON"),
  email = Sys.getenv("GARGLE_EMAIL")
)

cr_setup_role_lookup(
 type = c("local", "cloudrun", "bigquery", "secrets", "cloudbuild", "cloudstorage",
    "schedule_agent", "run_agent")
)
```

Arguments

`account_email`	The service account email e.g. `accountId@projectid.iam.gserviceaccount.com` or 12345678@cloudbuild.gserviceaccount.com
`roles`	the roles to grant access - default is all googleCloudRunner functions
`json`	the project clientId JSON
`email`	the email of an Owner/Editor for the project
`type`	the role

See Also

Other setup functions: `cr_setup_auth()`, `cr_setup_test()`, `cr_setup()`

cr_setup_test *Run tests over your setup*

Description

This allows you to check if your setup works - run cr_setup first.

Usage

```
cr_setup_test()
```

See Also

Other setup functions: `cr_setup_auth()`, `cr_setup_service()`, `cr_setup()`

cr_sourcerepo_list *List source repositories available under a project*

Description

List source repositories available under a project

Usage

```
cr_sourcerepo_list(projectId = cr_project_get())
```

Arguments

projectId	The projectId that holds the repositories

GitHubEventsConfig *GitHubEventsConfig Object*

Description

GitHubEventsConfig Object

Usage

```
GitHubEventsConfig(
  x,
  event = c("push", "pull"),
  branch = ".*",
  tag = NULL,
  commentControl = c("COMMENTS_DISABLED", "COMMENTS_ENABLED")
)
```

Arguments

x	The repository in format owner/repo e.g. MarkEdmondson1234/googleCloudRunner
event	Whether to trigger on push or pull GitHub events
branch	Regex of branches to match
tag	If a push request, regexes matching what tags to build. If not NULL then argument branch will be ignored
commentControl	If a pull request, whether to require comments before builds are triggered.

Details

The syntax of the regular expressions accepted is the syntax accepted by RE2 and described at https://github.com/google/re2/wiki/Syntax

Value

GitHubEventsConfig object

See Also

Other BuildTrigger functions: BuildTrigger(), cr_buildtrigger_copy(), cr_buildtrigger_delete(), cr_buildtrigger_edit(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_repo(), cr_buildtrigger_run(), cr_buildtrigger()

---

googleCloudRunner *Launch R scripts into the Google Cloud via Cloud Build, Cloud Run and Cloud Scheduler*

Description

See website for more details: https://code.markedmondson.me/googleCloudRunner/

See website for more details: https://code.markedmondson.me/googleCloudRunner/

HttpTarget *HttpTarget Object*

Description

HttpTarget Object

Usage

```
HttpTarget(
  headers = NULL,
  body = NULL,
  oauthToken = NULL,
  uri = NULL,
  oidcToken = NULL,
  httpMethod = NULL
)
```

Arguments

headers	A named list of HTTP headers e.g. `list(Blah = "yes",Boo = "no")`
body	HTTP request body. Just send in the R object/list, which will be base64encoded correctly
oauthToken	If specified, an OAuth token will be generated and attached as an Authorization header in the HTTP request. This type of authorization should be used when sending requests to a GCP endpoint.
uri	Required
oidcToken	If specified, an OIDC token will be generated and attached as an Authorization header in the HTTP request. This type of authorization should be used when sending requests to third party endpoints or Cloud Run.
httpMethod	Which HTTP method to use for the request

Value

HttpTarget object

See Also

https://cloud.google.com/scheduler/docs/reference/rest/v1/projects.locations.jobs#HttpTarget

Other Cloud Scheduler functions: `Job()`, `cr_build_schedule_http()`, `cr_schedule_delete()`, `cr_schedule_get()`, `cr_schedule_list()`, `cr_schedule_pause()`, `cr_schedule_run()`, `cr_schedule()`

Job *Job Schedule Object*

Description

Job Schedule Object

Usage

```
Job(
  attemptDeadline = NULL,
  pubsubTarget = NULL,
  httpTarget = NULL,
  timeZone = NULL,
  description = NULL,
  appEngineHttpTarget = NULL,
  status = NULL,
  retryConfig = NULL,
  state = NULL,
  name = NULL,
  lastAttemptTime = NULL,
  scheduleTime = NULL,
  schedule = NULL,
  userUpdateTime = NULL
)
```

Arguments

attemptDeadline	The deadline for job attempts
pubsubTarget	Pub/Sub target
httpTarget	A HTTP target object HttpTarget
timeZone	Specifies the time zone to be used in interpreting schedule. If set to NULL will be "UTC". Note that some time zones include a provision for daylight savings time.
description	Optionally caller-specified in CreateJob or
appEngineHttpTarget	App Engine HTTP target
status	Output only
retryConfig	Settings that determine the retry behavior
state	Output only
name	Optionally caller-specified in CreateJob, after
lastAttemptTime	Output only
scheduleTime	Output only
schedule	A cron schedule e.g. "15 5 * * *"
userUpdateTime	Output only

Details

Configuration for a job.The maximum allowed size for a job is 100KB.

Value

Job object

See Also

Other Cloud Scheduler functions: HttpTarget(), cr_build_schedule_http(), cr_schedule_delete(), cr_schedule_get(), cr_schedule_list(), cr_schedule_pause(), cr_schedule_run(), cr_schedule()

RepoSource *RepoSource Object*

Description

RepoSource Object

Usage

```
RepoSource(
  repoName = NULL,
  tagName = NULL,
  commitSha = NULL,
  branchName = NULL,
  dir = NULL,
  projectId = NULL
)
```

Arguments

repoName	Name of the Cloud Source Repository
tagName	Regex matching tags to build
commitSha	Explicit commit SHA to build
branchName	Regex matching branches to build e.g. ".*"
dir	Directory, relative to the source root, in which to run the build
projectId	ID of the project that owns the Cloud Source Repository

Details

Location of the source in a Google Cloud Source Repository.

Only one of commitSha, branchName or tagName are allowed.

If you want to use GitHub or BitBucket repos, you need to setup mirroring them via Cloud Source Repositories https://source.cloud.google.com/

Value

RepoSource object

See Also

Other Cloud Build functions: Build(), Source(), StorageSource(), cr_build_artifacts(), cr_build_make(), cr_build_status(), cr_build_upload_gcs(), cr_build_wait(), cr_build_write(), cr_build_yaml_artifact(), cr_build_yaml(), cr_build()

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")
## Not run:

my_repo <- cr_build_source(
  RepoSource("github_markedmondson1234_googlecloudrunner",
             branchName="master"))

build <- cr_build(
  cr_build_yaml(steps =
    cr_buildstep("gcloud", c("-c","ls -la"),
                 entrypoint = "bash",
                 dir = "")),
 source = my_repo)


## End(Not run)
```

Source *Source Object*

Description

It is suggested to use cr_build_source instead to build sources

Usage

```
Source(storageSource = NULL, repoSource = NULL)
```

Arguments

storageSource	If provided via StorageSource, get the source from this location in Google Cloud Storage
repoSource	If provided via RepoSource, get the source from this location in a Cloud Source

Details

Location of the source in a supported storage service.

Value

Source object

See Also

Other Cloud Build functions: Build(), RepoSource(), StorageSource(), cr_build_artifacts(), cr_build_make(), cr_build_status(), cr_build_upload_gcs(), cr_build_wait(), cr_build_write(), cr_build_yaml_artifact(), cr_build_yaml(), cr_build()

Examples

```
cr_project_set("my-project")
cr_bucket_set("my-bucket")
my_gcs_source <- Source(storageSource=StorageSource("my_code.tar.gz",
                                                    "gs://my-bucket"))
my_repo_source <- Source(repoSource=RepoSource("https://my-repo.com",
                                               branchName="master"))

## Not run:

build1 <- cr_build("cloudbuild.yaml", source = my_gcs_source)
build2 <- cr_build("cloudbuild.yaml", source = my_repo_source)


## End(Not run)
```

StorageSource *StorageSource Object*

Description

StorageSource Object

Usage

```
StorageSource(object, bucket = NULL, generation = NULL)
```

Arguments

object	Google Cloud Storage object containing the source. This object must be a gzipped archive file (.tar.gz) containing source to build.
bucket	Google Cloud Storage bucket containing the source
generation	Google Cloud Storage generation for the object. If the generation is omitted, the latest generation will be used.

Details

Location of the source in an archive file in Google Cloud Storage.

Value

StorageSource object

See Also

Other Cloud Build functions: Build(), RepoSource(), Source(), cr_build_artifacts(), cr_build_make(), cr_build_status(), cr_build_upload_gcs(), cr_build_wait(), cr_build_write(), cr_build_yaml_artifact(), cr_build_yaml(), cr_build()

Examples

```
## Not run:
cr_project_set("my-project")
cr_bucket_set("my-bucket")
# construct Source object
my_gcs_source <- Source(storageSource=StorageSource("my_code.tar.gz",
                                                    "gs://my-bucket"))
build1 <- cr_build("cloudbuild.yaml", source = my_gcs_source)

# helper that tars and adds to Source() for you
my_gcs_source2 <- cr_build_upload_gcs("my_folder")
build2 <- cr_build("cloudbuild.yaml", source = my_gcs_source2)


## End(Not run)
```

Index