Package ‘googleCloudRunner’

February 6, 2020

Type Package

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

Version 0.1.1

Description Tools to easily enable R scripts in the Google Cloud Platform.
Utilise cloud services such as Cloud Run <https://cloud.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), googleAuthR (>= 1.1.1),
googleCloudStorageR (>= 0.5.1), httr (>= 1.4.1), jsonlite (>= 1.5), methods, openssl (>= 1.4.1), stats, utils, yaml (>= 2.2.0)

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

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.0.2

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 2020-02-06 10:50:06 UTC
R topics documented:

- Build .................................................. 3
- BuildTrigger ........................................... 5
- cr_bucket_set .......................................... 6
- cr_build .................................................. 7
- cr_buildstep ........................................... 8
- cr_buildstep_bash ...................................... 11
- cr_buildstep_decrypt .................................. 12
- cr_buildstep_df ........................................ 13
- cr_buildstep_docker ..................................... 14
- cr_buildstep_edit ....................................... 15
- cr_buildstep_extract ................................... 16
- cr_buildstep_gitsetup .................................. 17
- cr_buildstep_mailgun ................................... 18
- cr_buildstep_nginx_setup ................................ 19
- cr_buildstep_pkgdown ................................... 20
- cr_buildstep_r .......................................... 21
- cr_buildstep_run ........................................ 23
- cr_buildstep_slack ....................................... 24
- cr_buildtrigger ......................................... 25
- cr_buildtrigger_delete .................................. 27
- cr_buildtrigger_edit ..................................... 27
- cr_buildtrigger_get ...................................... 28
- cr_buildtrigger_list ..................................... 29
- cr_buildtrigger_make .................................... 29
- cr_buildtrigger_run ...................................... 30
- cr_build_artifacts ....................................... 30
- cr_build_make ........................................... 32
- cr_build_schedule_http ................................ 33
- cr_build_source ......................................... 34
- cr_build_status .......................................... 35
- cr_build_upload_gcs ..................................... 35
- cr_build_wait ............................................ 37
- cr_build_write .......................................... 37
- cr_build_yaml ........................................... 38
- cr_build_yaml_artifact .................................. 39
- cr_deploy_docker ......................................... 40
- cr_deploy_gadget ......................................... 42
- cr_deploy_github_docker ................................ 42
- cr_deploy_git_html ....................................... 43
- cr_deploy_pkgdown ....................................... 45
- cr_deploy_r ............................................... 46
- cr_deploy_run ............................................ 48
- cr_email_get ............................................. 50
- cr_plumber_pubsub ...................................... 51
- cr_project_set .......................................... 52
- cr_pubsub ............................................... 52
**Build**

**Description**

Build Object

**Usage**

```r
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,
)```

**Index**

<table>
<thead>
<tr>
<th>Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cr_region_set</td>
<td>53</td>
</tr>
<tr>
<td>cr_run</td>
<td>53</td>
</tr>
<tr>
<td>cr_run_get</td>
<td>54</td>
</tr>
<tr>
<td>cr_run_list</td>
<td>55</td>
</tr>
<tr>
<td>cr_schedule</td>
<td>56</td>
</tr>
<tr>
<td>cr_schedule_delete</td>
<td>57</td>
</tr>
<tr>
<td>cr_schedule_get</td>
<td>58</td>
</tr>
<tr>
<td>cr_schedule_list</td>
<td>58</td>
</tr>
<tr>
<td>cr_schedule_pause</td>
<td>59</td>
</tr>
<tr>
<td>cr_schedule_run</td>
<td>60</td>
</tr>
<tr>
<td>cr_sourcerepo_list</td>
<td>61</td>
</tr>
<tr>
<td>GitHubEventsConfig</td>
<td>61</td>
</tr>
<tr>
<td>googleCloudRunner</td>
<td>62</td>
</tr>
<tr>
<td>HttpTarget</td>
<td>62</td>
</tr>
<tr>
<td>Job</td>
<td>63</td>
</tr>
<tr>
<td>RepoSource</td>
<td>65</td>
</tr>
<tr>
<td>Source</td>
<td>66</td>
</tr>
<tr>
<td>StorageSource</td>
<td>67</td>
</tr>
<tr>
<td>Build Object</td>
<td>69</td>
</tr>
</tbody>
</table>
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.
BuildTrigger

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 repository changes.

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
  name
- **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_delete(), cr_buildtrigger_edit(),
cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_make(), 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
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

```r
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_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_r(), cr_buildstep_slack()

Examples

```r
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_bash

```r
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**

```r
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_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_slack()`, `cr_buildstep()`
Examples

```r
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.

### Usage

```r
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_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `crBuildstepSlack()`, `cr_buildstep()`
Examples

```r
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

```r
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_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_slack()`, `cr_buildstep()`

Examples

```r
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)
```
Create a build step to build and push a docker image

Usage

```r
cr_buildstep_docker(
  image,
  tag = "$BUILD_ID",
  location = ".",
  projectId = cr_project_get(),
  ...
)
```

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 to attached to the pushed image - can use Build macros
- **location**: Where the Dockerfile to build is in relation to dir
- **projectId**: The projectId
- **...**: Further arguments passed in to `cr_buildstep`

See Also

Other Cloud Buildsteps: `cr_buildstep_bash()`, `cr_buildstep_decrypt()`, `cr_buildstep_df()`, `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_slack()`, `cr_buildstep()`

Examples

```r
# 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")
```
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)

## End(Not run)

---

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

**Description**
Useful for editing existing buildsteps

**Usage**

```r
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`  
  Prefix 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_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_slack()`, `cr_buildstep()`
cr_buildstep_extract

Extract a buildstep from a Build object

Examples

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

cr_buildstep_extract(built, step = 1)

edit_me <- cr_buildstep_extract(built, step = 2)

edit_me

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")
```

Description

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

Usage

```r
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_gitsetup(), cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_run(), cr_buildstep_r(), cr_buildstep_slack(), cr_buildstep()

Examples

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

cr_buildstep_extract(built, step = 1)

cr_buildstep_extract(built, step = 2)
```
**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**

```r
cr_buildstep_gitsetup(
  keyring = "my-keyring",
  key = "github-key",
  cipher = "id_rsa.enc",
  ...
)

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

**Arguments**

- **keyring**: The Key Management Store keyring containing the git ssh key
- **key**: The Key Management Store key containing the git ssh key
- **cipher**: The filename of the encrypted git ssh key that has been checked into the repository
- ... Further arguments passed in to **cr_buildstep**
- **git_args**: The arguments to send to git

**Details**

The key should be encrypted offline using `gcloud kms` or similar first. See **cr_buildstep_decrypt** for details.

By default the encrypted key should then be at the root of your **Source** object called "id_rsa.enc"
**cr_buildstep** must come after **cr_buildstep_gitsetup**

**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_mailgun()**, **cr_buildstep_nginx_setup()**, **cr_buildstep_pkgdown()**, **cr_buildstep_run()**, **cr_buildstep_r()**, **cr_buildstep_slack()**, **cr_buildstep()**
Examples

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

# assumes you have previously saved git ssh key via KMS called "git_key"
cr_build_yaml(
    steps = c(
        cr_buildstep_gitsetup("my_keyring", "git_key"),
        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

```r

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`
cr_buildstep_nginx_setup

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_gitsetup()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_slack()`, `cr_buildstep()

Examples

```r
 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@verified_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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>html_folder</td>
<td>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.</td>
</tr>
<tr>
<td>...</td>
<td>Other arguments passed to <code>cr_buildstep_bash</code></td>
</tr>
</tbody>
</table>

**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_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_r()`, `cr_buildstep_slack()`, `cr_buildstep()`

**Examples**

```r
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_pkgdown**  
Create buildsteps for deploying an R pkgdown website to GitHub

**Description**

Create buildsteps for deploying an R pkgdown website to GitHub

**Usage**

```r
cr_buildstep_pkgdown(
  github_repo, 
  git_email, 
  keyring = "my-keyring", 
  key = "github-key", 
  env = NULL, 
  cipher = "id_rsa.enc", 
  build_image = "gcr.io/gcer-public/packagetools:master"
)
```
Arguments

- **github_repo**: The GitHub repo to deploy pkgdown website from and to.
- **git_email**: The email the git commands will be identifying as
- **keyring**: The Key Management Store keyring containing the git ssh key
- **key**: The Key Management Store key containing the gitssh key
- **env**: A character vector of env arguments to set for all steps
- **cipher**: The filename of the encrypted git ssh key that has been checked into the repository
- **build_image**: A docker image with pkgdown installed

Details

It's 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

See Also

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

Examples

```r
# example set via build trigger macro _GITHUB_REPO
cr_buildstep_pkgdown("$_GITHUB_REPO",
  "cloudbuild@google.com")

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

# build process
build_yaml <- cr_build_yaml(steps = steps)
my_source <- cr_build_source(RepoSource("my_repo", branch="master"))
build <- cr_build_make(build_yaml, source = my_source)
```

---

**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

```r
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`
- `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.

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_gitsetup()`, `cr_buildstep_mailgun()`, `cr_buildstep_nginx_setup()`, `cr_buildstep_pkgdown()`, `cr_buildstep_run()`, `cr_buildstep_slack()`, `cr_buildstep`

Examples

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

# Not run:

```r
# 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:master")

---

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

**Description**

Create buildsteps to deploy to Cloud Run

**Usage**

```r

 cr_buildstep_run(  
 name,  
 image,  
 allowUnauthenticated = TRUE,  
 region = cr_region_get(),  
 concurrency = 80,  
 ...  
)
```

**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 image can serve. Can be up to 80.
- ...  
  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_gitsetup(),
            cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_r(),
            cr_buildstep_slack(), cr_buildstep()

---

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**

```r
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](https://github.com/technosophos/slack-notify).

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_gitsetup(),
            cr_buildstep_mailgun(), cr_buildstep_nginx_setup(), cr_buildstep_pkgdown(), cr_buildstep_r(),
            cr_buildstep_slack(), 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

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

### Arguments

- **name**: User assigned name of the trigger
- **trigger**: The trigger source which will be a `RepoSource` or a `GitHubEventsConfig`
- **build**: A file location within the trigger source to use for the build steps, or a `Build` object
- **description**: Human-readable description of this trigger
- **tags**: Tags for annotation of a ‘BuildTrigger’
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
projectId ID of the project for which to configure automatic builds

See Also
Other BuildTrigger functions: BuildTrigger(), GitHubEventsConfig(), cr_buildtrigger_delete(),
cr_buildtrigger_edit(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_make(),
cr_buildtrigger_run()

Examples

```r
cr_project_set("my-project")
cr_bucket_set("my-bucket")
cloudbuild <- system.file("cloudbuild/cloudbuild.yaml", 
                          package = "googleCloudRunner")
bb <- cr_build_make(cloudbuild, projectId = "test-project")
github <- GitHubEventsConfig("MarkEdmondson1234/googleCloudRunner", 
                           branch = "master")
# creates a trigger with named substitutions
ss <- list(\_MYVAR\_MYVAR = "TEST1",
            \_GITHUB\_GITHUB = "MarkEdmondson1234/googleCloudRunner")

## Not run:
cr_buildtrigger("trig1", trigger = github, build = bb)
cr_buildtrigger("trig2", trigger = github, 
                  build = bb, 
                  substitutions = ss)

# create a trigger that will build from the file in the repo
# this is similar to what cr_deploy_docker_github() does

build_docker <- cr_build_make(
    cr_build_yaml(
        steps = cr_buildstep_docker("build-dockerfile"), 
        images = "gcr.io/my-project/my-image"
    ))

cr_buildtrigger("trig4", trigger = github, 
                build = build_docker)

## 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

```python
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_edit(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_make(), 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

```python
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


**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_delete()`, `cr_buildtrigger_get()`, `cr_buildtrigger_list()`, `cr_buildtrigger_make()`, `cr_buildtrigger_run()`, `cr_buildtrigger_edit()`

**Examples**

```r
## Not run:

github <- GitHubEventsConfig("MarkEdmondson1234/googleCloudRunner",
   branch = "master")
between <- 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)
```

**See Also**

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

---

cr_buildtrigger_make  Create a buildtrigger object

Description
Create a buildtrigger object

Usage

   cr_buildtrigger_make(...)

Arguments

   ...  Arguments passed on to cr_buildtrigger
trigger  The trigger source which will be a RepoSource or a GitHubEventsCon-
fig
build  A file location within the trigger source to use for the build steps, or a
   Build object
projectId  ID of the project for which to configure automatic builds
name  User assigned name of the trigger
description  Human-readable description of this trigger
tags  Tags for annotation of a ‘BuildTrigger’
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

See Also

Other BuildTrigger functions: BuildTrigger(), GitHubEventsConfig(), cr_buildtriggert_delete(),
cr_buildtriggert_edit(), cr_buildtriggert_get(), cr_buildtriggert_list(), cr_buildtriggert_run(),
cr_buildtriggert()
Usage

```r
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
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
```

```r
build <- cr_build(ba)
built <- cr_build_wait(build)

cr_build_artifacts(built)
```

```r
## End(Not run)
```
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,
  projectId = cr_project_get()
)

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
projectId ID of the project

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, projectId = "test-project")
**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**

```r
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.gserviceaccount.com` 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**

```r
cloudbuild <- system.file("cloudbuild/cloudbuild.yaml", package = "googleCloudRunner")
b1 <- cr_build_make(cloudbuild, projectId = "my-project")

## Not run:
cr_schedule("15 5 * * *", name="cloud-build-test1",
             httpTarget = cr_build_schedule_http(b1))
```
# 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("15 5 * * *", name="itworks-schedule",
    httpTarget = cr_build_schedule_http(itworks))

## End(Not run)

---

## cr_build_source

### Build a source object

**Description**

Build a source object

**Usage**

```r
cr_build_source(x)
```

**Arguments**

- `x` A `RepoSource` or a `StorageSource` object

**Examples**

```r
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)
```
**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

```r
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


- 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_yml()`, `cr_build()`

---

**cr_build_upload_gcs**  
Create a StorageSource

### Description

This creates a **StorageSource** object after uploading to Google Cloud Storage
Usage

```r
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.
- **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

```r
## 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**

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

**Arguments**

- **op** The operation build object to wait for
- **projectId** The projectId

**Value**

A `Build` object

**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_artifact()`, `cr_build()`

**cr_build_write**

*Write out a Build object to cloudbuild.yaml*

**Description**

Write out a Build object to `cloudbuild.yaml`

**Usage**

```python
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
cr_build_yaml

See Also

Other Cloud Build functions: Build(), RepoSource(), Source(), StorageSource(), cr_build_artifacts(),

Examples

```r
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)
```

Description

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

Usage

```r
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 \texttt{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: \texttt{Build()}, \texttt{RepoSource()}, \texttt{Source()}, \texttt{StorageSource()}, \texttt{cr\_build\_artifacts()}, \texttt{cr\_build\_make()}, \texttt{cr\_build\_status()}, \texttt{cr\_build\_upload\_gcs()}, \texttt{cr\_build\_wait()}, \texttt{cr\_build\_write()}, \texttt{cr\_build\_yaml\_artifact()}, \texttt{cr\_build()}

Examples

\begin{verbatim}
  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)
\end{verbatim}

---

\texttt{cr\_build\_yaml\_artifact}

Add an artifact for cloudbuild.yaml

---

Description

Add artifact objects to a build

Usage

\begin{verbatim}
  cr_build_yaml_artifact(paths, bucket_dir = NULL, bucket = cr_bucket_get())
\end{verbatim}
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

```r
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")
)
```

### Description

Build a local Dockerfile in the cloud. See googleCloudRunner website for help how to generate Dockerfiles.

### Usage

```r
cr_deploy_docker(
  local,
  image_name = remote,
  dockerfile = NULL,
  remote = basename(local),
  tag = "$BUILD_ID",
  timeout = 600L,
  bucket = cr_bucket_get(),
  projectId = cr_project_get(),
  launch_browser = interactive(),
  ...
)
```
**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 to 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
- ... Other arguments passed to cr_buildstep_docker

**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**

Other Deployment functions: cr_deploy_git_html(), cr_deploy_github_docker(), cr_deploy_pkgdown(), cr_deploy_run(), cr_deploy_r()

**Examples**

```r
## 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_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**

```r
cr_deploy_gadget()
```

---

cr_deploy_github_docker  
Deploy Docker build from a GitHub repo (Experimental)

**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**

```r
cr_deploy_github_docker(
  x,
  image = x,
  branch = ".*",
  image_tag = "$SHORT_SHA",
  dockerfile_location = ".",
  github_tag = NULL,
  timeout = 600L,
  projectId = cr_project_get()
)
```

**Arguments**

- **x**: The GitHub repo e.g. MarkEdmondson1234/googleCloudRunner
- **image**: The name of the image you want to build
- **branch**: A regex of the GitHub branches that will trigger a build
- **image_tag**: What to tag the build docker image
- **dockerfile_location**: Where the Dockerfile sits within the GitHub repo
- **github_tag**: Regexes matching what tags to build. If not NULL then argument branch will be ignored
- **timeout**: timeout for the Docker build
- **projectId**: The project to build under
cr_deploy_git_html

Details

Build trigger API is experimental so this function is in development.

See Also

cr_deploy_docker which lets you build Dockerfiles for more generic use cases

Other Deployment functions: cr_deploy_docker(), cr_deploy_git_html(), cr_deploy_pkdown(), cr_deploy_run(), cr_deploy_r()

---

cr_deploy_git_html  
Deploy HTML built from a repo each commit (Experimental)

Description

This lets you set up triggers that will update a website each commit. You need to mirror the GitHub/Bitbucket repo onto Google Cloud Repositories for this to work.

Usage

cr_deploy_git_html(
  x,
  image = paste0(x, "-html"),
  rmd_folder = NULL,
  html_folder = NULL,
  branch = ".*",
  image_tag = "$SHORT_SHA",
  github_tag = NULL,
  timeout = 600L,
  edit_r = NULL,
  r_image = "gcr.io/gcer-public/packagetools:master",
  allowUnauthenticated = TRUE,
  region = cr_region_get(),
  projectID = cr_project_get()
)

Arguments

  x  The GitHub repo e.g. MarkEdmondson1234/googleCloudRunner
  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
  branch  A regex of the GitHub branches that will trigger a build
  image_tag  What to tag the build docker image
cr_deploy_git_html

github_tag   Regexes matching what tags to build. If not NULL then argument branch will be ignored

timeout     timeout for the Docker 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

Build trigger API is experimental so this function is in development.

This default R code is rendered in the rmd_folder:
lapply(list.files('.','pattern = '.Rmd',full.names = TRUE),rmarkdown::render,output_format = 'html_document')

You need to mirror the GitHub/Bitbucket repo onto Google Cloud Repositories for this to work

See Also

cr_deploy_html that lets you deploy HTML files

Other Deployment functions: cr_deploy_docker(), cr_deploy_github_docker(), cr_deploy_pkgdown(), cr_deploy_run(), cr_deploy_r()

Examples

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

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

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

## End(Not run)
```
Description

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

Usage

```r
cr_deploy_pkgdown(
  steps = NULL,
  cloudbuild_file = "cloudbuild.yml",
  git_email = "googlecloudrunner@r.com",
  keyring = "my-keyring",
  key = "github-key",
  env = NULL,
  cipher = "id_rsa.enc",
  build_image = "gcr.io/gcer-public/packagetools:master"
)
```

Arguments

- **steps**: extra steps to run before the pkgdown website steps run
- **cloudbuild_file**: The cloudbuild yaml file to write to
- **git_email**: The email the git commands will be identifying as
- **keyring**: The Key Management Store keyring containing the git ssh key
- **key**: The Key Management Store key containing the git ssh key
- **env**: A character vector of env arguments to set for all steps
- **cipher**: The filename of the encrypted git ssh key that has been checked into the repository
- **build_image**: A docker image with pkgdown installed

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](https://console.cloud.google.com/cloud-build/triggers/connect)

The git ssh keys need to be deployed to Google KMS for the deployment of the website - see [cr_buildstep_git](#) - this only needs to be done once per Git account. You then need to commit the encrypted ssh key (by default called id_rsa.enc)
See Also

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

Other Deployment functions: `cr_deploy_docker()`, `cr_deploy_git_html()`, `cr_deploy_github_docker()`, `cr_deploy_run()`, `cr_deploy_r()`

Examples

```r
pd <- cr_deploy_pkgdown()
file.exists("cloudbuild.yml")

unlink("cloudbuild.yml")
```

---

**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**

```r
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`

- `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.
cr_deploy_r

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
e-mail            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(), cr_deploy_git_html(), cr_deploy_github_docker(),
cr_deploy_pkgdown(), 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)

---

**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**

```r
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 to 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
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

Other Deployment functions: `cr_deploy_docker()`, `cr_deploy_git_html()`, `cr_deploy_github_docker()`, `cr_deploy_pkgdown()`, `cr_deploy_r()`

Examples

```r
## 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_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

```r
cr_email_get()

cr_email_set(cloudbuildEmail)
```
Arguments

  cloudbuildEmail

      The Cloud Build service email

See Also

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

---

**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 plumber 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.

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.

See Also

  Google Pub/Sub tutorial for Cloud Run
  Other Cloud Run functions: cr_run_get(), cr_run_list(), cr_run()
Examples

```r
## 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

```

```
cr_pubsub Send a message to pubsub

Description

Useful for testing Cloud Run pubsub deployments
```
**cr_region_set**

*Usage*

```r
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*

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

*Arguments*

- **region**
  
  Region for the endpoint

---

**cr_run**  
*Create a CloudRun service.*

*Description*

Deploys an existing gcr.io image.

*Usage*

```r
cr_run(
  image,
  name = basename(image),
  allowUnauthenticated = TRUE,
  concurrency = 1,
  timeout = 600L,
  region = cr_region_get(),
  projectId = cr_project_get(),
  launch_browser = interactive()
)
```
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 image can serve. Can be up to 80.
- **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

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_plumber_pubsub()`, `cr_run_get()`, `cr_run_list()

Examples

```r
## Not run:
cr_project_set("my-project")
cr_region_set("europe-west1")
cr_run("gcr.io/my-project/my-image")
## End(Not run)
```

---

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

Description

Get information about a Cloud Run service.

Usage

```r
cr_run_get(name, projectId = cr_project_get())
```
**cr_run_list**

### 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_plumber_pubsub()`, `cr_run_list()`, `cr_run()`

### Description

List the Cloud Run services you have access to

### Usage

```r
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_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,
    region = cr_region_get(),
    projectId = cr_project_get()
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Optionally caller-specified in CreateJob, after</td>
</tr>
<tr>
<td>schedule</td>
<td>A cron schedule e.g. &quot;15 5 * * *&quot;</td>
</tr>
<tr>
<td>httpTarget</td>
<td>A HTTP target object <code>HttpTarget</code></td>
</tr>
<tr>
<td>description</td>
<td>Optionally caller-specified in CreateJob or</td>
</tr>
<tr>
<td>overwrite</td>
<td>If TRUE and an existing job with the same name exists, will overwrite it with the new parameters</td>
</tr>
<tr>
<td>region</td>
<td>The region usually set with <code>cr_region_set</code></td>
</tr>
<tr>
<td>projectId</td>
<td>The GCP project to run within usually set with <code>cr_project_set</code></td>
</tr>
</tbody>
</table>

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)

## 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

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

### Arguments

- **name**: Required
- **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

```r
## 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

```r
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()`

**Examples**

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

## End(Not run)
```

---

**Description**

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

**Usage**

```r
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

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

## End(Not run)
```

**cr_sourcerepo_list**  List source repositories available under a project

**Description**

List source repositories available under a project

**Usage**

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

**Arguments**

- `projectId` The projectId that holds the repositories

**GitHubEventsConfig**  GitHubEventsConfig Object

**Description**

GitHubEventsConfig Object

**Usage**

```r
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_delete(), cr_buildtrigger_edit(), cr_buildtrigger_get(), cr_buildtrigger_list(), cr_buildtrigger_make(), 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

HttpTarget

HttpTarget Object

Description

HttpTarget Object
### Usage

```r
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 base64 encoded 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
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**

```python
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

```r
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

```r
## 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

as.yaml, 7
Build, 3, 7, 16, 18, 21, 25, 29, 31–33, 35–40, 47, 66–68
BuildTrigger, 5, 26–30, 62
cr_bucket_get (cr_bucket_set), 6
cr_bucket_set, 6
cr_build, 5, 7, 13, 31–33, 35–40, 47, 66–68
cr_build_artifacts, 5, 7, 30, 32, 35–40, 66–68
cr_build_make, 5, 7, 31, 32, 35–40, 66–68
cr_build_schedule_http, 33, 47, 56–60, 63, 65
cr_build_source, 4, 7, 32, 34, 46, 47, 66
cr_build_status, 5, 7, 31, 32, 35, 36–40, 66–68
cr_build_upload_gcs, 5, 7, 31, 32, 35, 37–40, 66–68
cr_build_wait, 5, 7, 31, 32, 35, 36, 37, 38–40, 66–68
cr_build_write, 5, 7, 31, 32, 35–37, 37, 39, 40, 66–68
cr_build_yaml, 5, 7, 31, 32, 35–38, 38, 40, 66–68
cr_build_yaml_artifact, 5, 7, 31, 32, 35–39, 39, 66–68
cr_buildstep, 8, 11–17, 19–24, 39, 47
cr_buildstep_bash, 10, 11, 12–17, 19–22, 24
cr_buildstep_decrypt, 10, 11, 12, 13–17, 19–22, 24
cr_buildstep_df, 10–12, 13, 14–17, 19–22, 24
cr_buildstep_docker, 10–13, 14, 15–17, 19–22, 24, 41
cr_buildstep_edit, 10–14, 15, 16, 17, 19–22, 24
cr_buildstep_extract, 10–15, 16, 17, 19–22, 24
cr_buildstep_git, 45
cr_buildstep_git
(cr_buildstep_gitsetup), 17
cr_buildstep_gitsetup, 10–16, 17, 19–22, 24
cr_buildstep_mailgun, 10–17, 18, 20–22, 24
cr_buildstep_nginx_setup, 10–17, 19, 19, 21, 22, 24
cr_buildstep_pkgdown, 10–17, 19, 20, 20, 22, 24, 46
cr_buildstep_r, 10–21, 21, 24, 44
cr_buildstep_run, 10–17, 19–22, 23, 24
cr_buildstep_slack, 10–17, 19–22, 24, 24
39, 45, 50
cr_buildtrigger, 6, 25, 27–30, 47, 62
cr_buildtrigger_delete, 6, 26, 27, 28–30, 62
cr_buildtrigger_edit, 6, 26, 27, 27, 28–30, 62
cr_buildtrigger_get, 6, 26–28, 29, 30, 50
cr_buildtrigger_list, 6, 26–28, 29, 30, 62
cr_buildtrigger_make, 6, 26–28, 29, 30, 62
cr_buildtrigger_run, 6, 26–30, 30, 62
cr_deploy_docker, 40, 43, 44, 46, 47, 50, 54
cr_deploy_gadget, 42
cr_deploy_github_docker, 41, 42, 44, 46, 47, 50
cr_deploy_git_html, 41, 43, 46, 47, 50
44, 48
43, 44, 45, 47, 50
41
42
44
44
41, 43, 44, 46, 47, 48, 54
45
50
33, 47
50
69
cr_plumber_pubsub, 51, 54, 55
cr_project_get (cr_project_set), 52
cr_project_set, 52, 56
cr_pubsub, 52
cr_region_get (cr_region_set), 53
cr_region_set, 47, 53, 56
cr_run, 51, 53, 55
cr_run_get, 51, 54, 54, 55
cr_run_list, 51, 54, 55, 55
cr_schedule, 33, 56, 57–60, 63, 65
cr_schedule_delete, 33, 56, 57, 58–60, 63, 65
cr_schedule_get, 33, 56, 57, 58, 59, 60, 63, 65
cr_schedule_list, 33, 56–58, 58, 60, 63, 65
cr_schedule_pause, 33, 56–59, 59, 60, 63, 65
cr_schedule_resume, 59
cr_schedule_resume (cr_schedule_pause), 59
cr_schedule_run, 33, 56–60, 60, 63, 65
cr_sourcerpo_list, 61

GitHubEventsConfig, 6, 25–30, 61
googleCloudRunner, 62

HttpTarget, 33, 56–60, 62, 64, 65

I, 13

Job, 33, 47, 56–60, 63, 63

plumb, 51

render, 43


Source, 4, 5, 7, 17, 31, 32, 35–40, 46, 47, 66, 66, 68

StorageSource, 5, 7, 31, 32, 34–40, 66, 67, 67