Package: didehpc (via r-universe)

September 26, 2024

Title DIDE HPC Support

Version 0.3.22

Description DIDE HPC support.

License MIT + file LICENSE

URL https://github.com/mrc-ide/didehpc

BugReports https://github.com/mrc-ide/didehpc/issues

Depends R (>= 3.2.2)

Imports conan (>= 0.1.1), crayon, context (>= 0.5.0), getPass, glue, httr (>= 1.0.0), ids, jsonlite (>= 1.6), queuer (>= 0.5.0), rematch, storr (>= 1.1.1), xml2 (>= 1.0.0)

Suggests R6, callr, knitr, mockery, pkgdepends (>= 0.1.0), redux, rmarkdown, rrq (>= 0.7.0), testthat, withr

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Remotes mrc-ide/conan, mrc-ide/context, mrc-ide/queuer, mrc-ide/rrq

Encoding UTF-8

Language en-GB

Repository https://mrc-ide.r-universe.dev

RemoteUrl https://github.com/mrc-ide/didehpc

RemoteRef master

RemoteSha aa7ee21336397ad740a191f8c2da7c358698be77

Contents

cluster_load 2
didehpc_config 2
path_mapping 5
queue_didehpc 6

valid_clusters 11
web_client 11
web_login 15
worker_resource 15

Index **17**

`cluster_load` *Overall cluster load*

Description

Overall cluster load for all clusters that you have access to.

Usage

```
cluster_load(credentials = NULL)
```

Arguments

`credentials`	Your credentials

`didehpc_config` *Configuration*

Description

Collects configuration information. Unfortunately there's a fairly complicated process of working out what goes where so documentation coming later.

Usage

```
didehpc_config(
  credentials = NULL,
  home = NULL,
  temp = NULL,
  cluster = NULL,
  shares = NULL,
  template = NULL,
  cores = NULL,
  wholenode = NULL,
  parallel = NULL,
  workdir = NULL,
  use_workers = NULL,
  use_rrq = NULL,
  worker_timeout = NULL,
  worker_resource = NULL,
```

```
  conan_bootstrap = NULL,
  r_version = NULL,
  use_java = NULL,
  java_home = NULL
)

didehpc_config_global(..., check = TRUE)
```

Arguments

credentials	Either a list with elements username, password, or a path to a file containing lines username=<username> and password=<password> or your username (in which case you will be prompted graphically for your password).
home	Path to network home directory, on local system
temp	Path to network temp directory, on local system
cluster	Name of the cluster to use; one of valid_clusters() or one of the aliases (small/little/dide/ide; big/mrc).
shares	Optional additional share mappings. Can either be a single path mapping (as returned by path_mapping() or a list of such calls.
template	A job template. On fi–dideclusthn this can be "GeneralNodes" or "8Core". On "fi–didemrchnb" this can be "GeneralNodes", "12Core", "16Core", "12and16Core", "20Core", "24Core", "32Core", or "MEM1024" (for nodes with 1Tb of RAM; we have three - two of which have 32 cores, and the other is the AMD epyc with 64). On the new "wpia-hn" cluster, you should currently use "AllNodes". See the main cluster documentation if you tweak these parameters, as you may not have permission to use all templates (and if you use one that you don't have permission for the job will fail). For training purposes there is also a "Training" template, but you will only need to use this when instructed to.
cores	The number of cores to request. If specified, then we will request this many cores from the windows queuer. If you request too many cores then your task will queue forever! 24 is the largest this can be on fi–dideclusthn. On fi–didemrchnb, the GeneralNodes template has mainly 20 cores or less, with a single 64 core node, and the 32Core template has 32 core nodes. On wpia-hn, all the nodes are 32 core. If cores is omitted then a single core is assumed, unless wholenode is TRUE.
wholenode	If TRUE, request exclusive access to whichever compute node is allocated to the job. Your code will have access to all the cores and memory on the node.
parallel	Should we set up the parallel cluster? Normally if more than one core is implied (via the cores or wholenode arguments, then a parallel cluster will be set up (see Details). If parallel is set to FALSE then this will not occur. This might be useful in cases where you want to manage your own job level parallelism (e.g. using OpenMP) or if you're just after the whole node for the memory).
workdir	The path to work in on the cluster, if running out of place.
use_workers	Submit jobs to an internal queue, and run them on a set of workers submitted separately? If TRUE, then enqueue and the bulk submission commands no longer submit to the DIDE queue. Instead they create an *internal* queue that workers

can poll. After queuing tasks, use `submit_workers` to submit workers that will process these tasks, terminating when they are done. You can use this approach to throttle the resources you need.

`use_rrq` Use `rrq` to run a set of workers on the cluster. This is an experimental option, and the interface here may change. For now all this does is ensure a few additional packages are installed, and tweaks some environment variables in the generated batch files. Actual rrq workers are submitted with the `submit_workers` method of the object.

`worker_timeout` When using workers (via `use_workers` or `use_rrq`, the length of time (in seconds) that workers should be willing to set idle before exiting. If set to zero then workers will be added to the queue, run jobs, and immediately exit. If greater than zero, then the workers will wait at least this many seconds after running the last task before quitting. The number provided can be `Inf`, in which case the worker will never exit (but be careful to clean the worker up in this case!). The default is 600s (10 minutes) should be more than enough to get your jobs up and running. Once workers are established you can extend or reset the timeout by sending the `TIMEOUT_SET` message (proper documentation will come for this soon).

`worker_resource`

Optionally, an object created by `worker_resource()` which controls the resources used by workers where these are different to jobs directly submitted by `$enqueue()`. This is only meaningful if you are using `use_rrq = TRUE`.

`conan_bootstrap`

Logical, indicating if we should use the shared conan "bootstrap" library stored on the temporary directory. Setting this to `FALSE` will install all dependencies required to install packages first into a temporary location (this may take a few minutes) before installation. Generally leave this as-is.

`r_version` A string, or `numeric_version` object, describing the R version required. Not all R versions are known to be supported, so this will check against a list of installed R versions for the cluster you are using. If omitted then: if your R version matches a version on the cluster that will be used, or the oldest cluster version that is newer than yours, or the most recent cluster version.

`use_java` Logical, indicating if the script is going to require Java, for example via the rJava package.

`java_home` A string, optionally giving the path of a custom Java Runtime Environment, which will be used if the use_java logical is true. If left blank, then the default cluster Java Runtime Environment will be used.

`...` arguments to `didehpc_config`

`check` Logical, indicating if we should check that the configuration object can be created

Resources and parallel computing

If you need more than one core per task (i.e., you want the each task to do some parallel processing *in addition* to the parallelism between tasks) you can do that through the configuration options here.

The `template` option chooses among templates defined on the cluster.

If you specify `cores`, the HPC will queue your job until an appropriate number of cores appears for the selected template. This can leave your job queuing forever (e.g., selecting 20 cores on a 16Core template) so be careful.

Alternatively, if you specify `wholenode` as TRUE, then you will have exclusive access to whichever compute node is allocated to your job, reserving all of its cores.

If more than 1 core is requested, either by choosing `wholenode`, or by specifying a `cores` value greater than 1) on startup, a `parallel` cluster will be started, using `parallel::makePSOCKcluster` and this will be registered as the default cluster. The nodes will all have the appropriate context loaded and you can immediately use them with `parallel::clusterApply` and related functions by passing `NULL` as the first argument. The cluster will be shut down politely on exit, and logs will be output to the "workers" directory below your context root.

Workers and rrq

The options `use_workers` and `use_rrq` interact, share some functionality, but are quite different.

With `use_workers`, jobs are never submitted when you run `enqueue` or one of the bulk submission commands in `queuer`. Instead you submit workers using `submit_workers` and then the submission commands push task ids onto a Redis queue that the workers monitor.

With `use_rrq`, `enqueue` etc still work as before, plus you *must* submit workers with `submit_workers`. The difference is that any job may access the `rrq_controller` and push jobs onto a central pool of tasks.

I'm not sure at this point if it makes any sense for the two approaches to work together so this is disabled for now. If you think you have a use case please let me know.

`path_mapping` *Describe a path mapping*

Description

Describe a path mapping for use when setting up jobs on the cluster.

Usage

```
path_mapping(name, path_local, path_remote, drive_remote)
```

Arguments

`name`	Name of this map. Can be anything at all, and is used for information purposes only.
`path_local`	The point where the drive is attached locally. On Windows this will be something like "Q:/", on Mac something like "/Volumes/mountname", and on Linux it could be anything at all, depending on what you used when you mounted it (or what is written in `/etc/fstab`)

`path_remote` The network path for this drive. It will look something like `\\\\fi--didef3.dide.ic.ac.uk\\tmp\\`. Unfortunately backslashes are really hard to get right here and you will need to use twice as many as you expect (so *four* backslashes at the beginning and then two for each separator). If this makes you feel bad know that you are not alone: https://xkcd.com/1638 – alternatively you may use forward slashes in place of backslashes (e.g. `//fi--didef3.dide.ic.ac.uk/tmp`)

`drive_remote` The place to mount the drive on the cluster. We're probably going to mount things at Q: and T: already so don't use those. And things like C: are likely to be used. Perhaps there are some guidelines for this somewhere?

Author(s)

Rich FitzJohn

`queue_didehpc` *Create a queue object*

Description

Create a queue object. This is an R6::R6Class object which you interact with by calling "methods" which are described below, and on the help page for queuer::queue_base, from which this derives.

Usage

```
queue_didehpc(
  context,
  config = didehpc_config(),
  root = NULL,
  initialise = TRUE,
  provision = NULL,
  login = NULL
)
```

Arguments

`context` A context

`config` Optional dide configuration information.

`root` A root directory, not usually needed

`initialise` Passed through to the base queue. If you set this to `FALSE` you will not be able to submit tasks. By default if `FALSE` this also sets `provision` to `later` and `login` to `FALSE`.

`provision` A provisioning strategy to use. Options are

- `verylazy` (the default) which installs packages if any declared package is not present, or if the remote library has already been provisioned. This is lazier than the `lazy` policy and faster as it avoids fetching package metadata, which may take a few seconds. If you have manually adjusted your library (especially by removing packages) you will probably want to use `lazy` or `upgrade` to account for dependencies of your declared packages.
- `lazy`: which tells `pkgdepends` to be "lazy" - this prefers to minimise installation time and does not upgrade packages unless required.
- `upgrade`: which tells `pkgdepends` to always try and upgrade
- `later`: don't do anything on creation
- `fake`: don't do anything but mark the queue as being already provisioned (this option can come in useful if you really don't want to risk any accidental package installation)

`login` Logical, indicating if we should immediately login. If `TRUE`, then you will be prompted to login immediately, rather than when a request to the web portal is made.

Super class

`queuer::queue_base` -> `queue_didehpc`

Public fields

`config` Your `didehpc_config()` for this queue. Do not change this after queue creation as changes may not take effect as expected.

`client` A web_client object used to communicate with the web portal. See the help page for its documentation, but you will typically not need to interact with this.

Methods

Public methods:

- `queue_didehpc_$new()`
- `queue_didehpc_$login()`
- `queue_didehpc_$cluster_load()`
- `queue_didehpc_$reconcile()`
- `queue_didehpc_$submit()`
- `queue_didehpc_$submit_workers()`
- `queue_didehpc_$stop_workers()`
- `queue_didehpc_$rrq_controller()`
- `queue_didehpc_$unsubmit()`
- `queue_didehpc_$dide_id()`
- `queue_didehpc_$dide_log()`
- `queue_didehpc_$provision_context()`
- `queue_didehpc_$install_packages()`

Method `new()`**:** Constructor

Usage:

```
queue_didehpc_$new(
  context,
  config,
  root,
  initialise,
  provision,
  login,
  client = NULL
)
```

Arguments:

`context, config, root, initialise, provision, login` See above

`client` A web_client object, primarily useful for testing the package

Method `login()`**:** Log onto the web portal. This will be called automatically at either when creating the object (by default) or when you make your first request to the portal. However, you can call this to refresh the session too.

Usage:

```
queue_didehpc_$login(refresh = TRUE)
```

Arguments:

`refresh` Logical, indicating if we should try logging on again, even if it looks like we already have. This will refresh the session, which is typically what you want to do.

Method `cluster_load()`**:** Report on the overall cluster usage

Usage:

```
queue_didehpc_$cluster_load(cluster = NULL, nodes = TRUE)
```

Arguments:

`cluster` Cluster to show; if `TRUE` show the entire cluster (via `load_overall`), if `NULL` defaults to the value `config$cluster`

`nodes` Show the individual nodes when printing

Method `reconcile()`**:** Attempt to reconcile any differences in task state between our database and the HPC queue. This is needed when tasks have crashed, or something otherwise bad has happened and you have tasks stuck in `PENDING` or `RUNNING` that are clearly not happy. This function does not play well with workers and you should not use it if using them.

Usage:

```
queue_didehpc_$reconcile(task_ids = NULL)
```

Arguments:

`task_ids` A vector of tasks to check

Method `submit()`**:** Submit a task to the queue. Ordinarily you do not call this directly, it is called by the `$enqueue()` method of queuer::queue_base when you create a task. However, you can use this to resubmit a task that has failed if you think it will run successfully a second time (e.g., because you cancelled it the first time around).

Usage:

```
queue_didehpc_$submit(task_ids, names = NULL, depends_on = NULL)
```

Arguments:

`task_ids` A vector of task identifiers to submit.

`names` Optional names for the tasks.

`depends_on` Optional vector of dependencies, named by task id

Method `submit_workers()`**:** Submit workers to the queue. This only works if `use_rrq` or `use_workers` is `TRUE` in your configuration. See `vignette("workers")` for more information.

Usage:

```
queue_didehpc_$submit_workers(n, timeout = 600, progress = NULL)
```

Arguments:

`n` The number of workers to submit

`timeout` The time to wait, in seconds, for all workers to come online. An error will be thrown if this time is exceeded.

`progress` Logical, indicating if a progress bar should be printed while waiting for workers.

Method `stop_workers()`**:** Stop workers running on the cluster. See `vignette("workers")` for more information. By default workers will timeout after 10 minutes of inactivity.

Usage:

```
queue_didehpc_$stop_workers(worker_ids = NULL)
```

Arguments:

`worker_ids` Vector of worker names to try and stop. By default all workers are stopped.

Method `rrq_controller()`**:** Return an rrq::rrq_controller object, if you have set `use_rrq` or `use_workers` in your configuration. This is a lightweight queue using your workers which is typically much faster than submitting via `$enqueue()`. See `vignette("workers")` for more information.

Usage:

```
queue_didehpc_$rrq_controller()
```

Method `unsubmit()`**:** Unsubmit tasks from the cluster. This removes the tasks from the queue if they have not been started yet, and stops them if currently running. It will have no effect if the tasks are completed (successfully or errored)

Usage:

```
queue_didehpc_$unsubmit(task_ids)
```

Arguments:

`task_ids` Can be a task id (string), a vector of task ids, a task, a list of tasks, a bundle returned by enqueue_bulk, or a list of bundles.

Method `dide_id()`**:** Find the DIDE task id of your task. This is the number displayed in the web portal.

Usage:

```
queue_didehpc_$dide_id(task_ids)
```

Arguments:

`task_ids` Vector of task identifiers to look up

Method `dide_log()`**:** Return the pre-context log of a task. Use this to find out what has happened to a task that has unexpectedly failed, but for which `$log()` is uninformative.

Usage:

```
queue_didehpc_$dide_log(task_id)
```

Arguments:

`task_id` A single task id to check

Method `provision_context()`**:** Provision your context for running on the cluster. This sets up the remote set of packages that your tasks will use. See `vignette("packages")` for more information.

Usage:

```
queue_didehpc_$provision_context(
  policy = "verylazy",
  dryrun = FALSE,
  quiet = FALSE,
  show_progress = NULL,
  show_log = TRUE
)
```

Arguments:

`policy` The installation policy to use, as interpreted by `pkgdepends::pkg_solution` - so this should be `verylazy`/`lazy` (install missing packages but don't upgrade unless needed) or `upgrade` (upgrade packages as possible). In addition you can also use `later` which does nothing, or `fake` which pretends that it ran the provisioning. See `vignette("packages")` for details on these options.

`dryrun` Do a dry run installation locally - this just checks that your requested set of packages is plausible, but does this without submitting a cluster job so it may be faster.

`quiet` Logical, controls printing of informative messages

`show_progress` Logical, controls printing of a spinning progress bar

`show_log` Logical, controls printing of the log from the cluster

Method `install_packages()`**:** Install packages on the cluster. This can be used to more directly install packages on the cluster than the `$provision_context` method that you would typically use. See `vignette("packages")` for more information.

Usage:

```
queue_didehpc_$install_packages(
  packages,
  repos = NULL,
  policy = "lazy",
  dryrun = FALSE,
  show_progress = NULL,
  show_log = TRUE
)
```

Arguments:

packages A character vector of packages to install. These can be names of CRAN packages or GitHub references etc; see pkgdepends::new_pkg_installation_proposal() and vignette("packages") for more details

repos A character vector of repositories to use when installing. A suitable CRAN repo will be added if not detected.

policy The installation policy to use, as interpreted by pkgdepends::pkg_solution - so this should be lazy (install missing packages but don't upgrade unless needed) or upgrade (upgrade packages as possible). In addition you can also use later which does nothing, or fake which pretends that it ran the provisioning. See vignette("packages") for details on these options.

dryrun Do a dry run installation locally - this just checks that your requested set of packages is plausible, but does this without submitting a cluster job so it may be faster.

show_progress Logical, controls printing of a spinning progress bar

show_log Logical, controls printing of the log from the cluster

---

valid_clusters *Valid DIDE clusters*

Description

Valid cluster names

Usage

```
valid_clusters()
```

---

web_client *DIDE cluster web client*

Description

DIDE cluster web client

DIDE cluster web client

Details

Client for the DIDE cluster web interface.

Methods

Public methods:

- web_client$new()
- web_client$login()
- web_client$logout()
- web_client$logged_in()
- web_client$check_access()
- web_client$submit()
- web_client$cancel()
- web_client$log()
- web_client$status_user()
- web_client$status_job()
- web_client$load_node()
- web_client$load_overall()
- web_client$load_show()
- web_client$headnodes()
- web_client$r_versions()
- web_client$api_client()

Method `new()`**:** Create an API client for the DIDE cluster

Usage:

```
web_client$new(
  credentials = NULL,
  cluster_default = "fi--dideclusthn",
  login = FALSE,
  client = NULL
)
```

Arguments:

`credentials` Either your username, or a list with at least the element `username` and possibly the name 'password. If not given, your password will be prompted for at login.

`cluster_default` The default cluster to use; this can be overridden in any command.

`login` Logical, indicating if we should immediately login

`client` Optional API client object - if given then we prefer this object rather than trying to create a new client with the given credentials.

Method `login()`**:** Log in to the cluster

Usage:

```
web_client$login(refresh = TRUE)
```

Arguments:

`refresh` Logical, indicating if we should login even if it looks like we are already (useful if login has expired)

Method `logout()`**:** Log the client out

Usage:

```
web_client$logout()
```

Method `logged_in()`**:** Test whether the client is logged in, returning `TRUE` or `FALSE`.

Usage:

```
web_client$logged_in()
```

Method `check_access()`**:** Validate that we have access to a given cluster

Usage:

```
web_client$check_access(cluster = NULL)
```

Arguments:

`cluster` The name of the cluster to check, defaulting to the value given when creating the client.

Method `submit()`**:** Submit a job to the cluster

Usage:

```
web_client$submit(
  path,
  name,
  template,
  cluster = NULL,
  resource_type = "Cores",
  resource_count = 1,
  depends_on = NULL
)
```

Arguments:

`path` The path to the job to submit. This must be a windows (UNC) network path, starting with two backslashes, and must be somewhere that the cluster can see.

`name` The name of the job (will be displayed in the web interface).

`template` The name of the template to use.

`cluster` The cluster to submit to, defaulting to the value given when creating the client.

`resource_type` The type of resource to request (either `Cores` or `Nodes`)

`resource_count` The number of resources to request

`depends_on` Optional. A vector of dide ids that this job depends on.

Method `cancel()`**:** Cancel a cluster task

Usage:

```
web_client$cancel(dide_id, cluster = NULL)
```

Arguments:

`dide_id` The DIDE task id for the task

`cluster` The cluster that the task is running on, defaulting to the value given when creating the client.

Returns: A named character vector with a status reported by the cluster head node. Names will be the values of `dide_id` and values one of `OK`, `NOT_FOUND`, `WRONG_USER`, `WRONG_STATE`, `ID_ERROR`

Method `log()`**:** Get log from job

Usage:

```
web_client$log(dide_id, cluster = NULL)
```

Arguments:

`dide_id` The DIDE task id for the task

`cluster` The cluster that the task is running on, defaulting to the value given when creating the client.

Method `status_user()`**:** Return status of all your jobs

Usage:

```
web_client$status_user(state = "*", cluster = NULL)
```

Arguments:

`state` The state the job is in. Can be one of `Running`, `Finished`, `Queued`, `Failed` or `Cancelled`. Or give `*` for all states (this is the default).

`cluster` The cluster to query, defaulting to the value given when creating the client.

Method `status_job()`**:** Return status of a single job

Usage:

```
web_client$status_job(dide_id, cluster = NULL)
```

Arguments:

`dide_id` The id of the job - this will be an integer

`cluster` The cluster to query, defaulting to the value given when creating the client.

Method `load_node()`**:** Return an overall measure of cluster use, one entry per node within a cluster.

Usage:

```
web_client$load_node(cluster = NULL)
```

Arguments:

`cluster` The cluster to query, defaulting to the value given when creating the client.

Method `load_overall()`**:** Return an overall measure of cluster use, one entry per cluster that you have access to. Helper function; wraps around `load_overall` and `load_node` and always shows the output.

Usage:

```
web_client$load_overall()
```

Method `load_show()`**:**

Usage:

```
web_client$load_show(cluster = NULL, nodes = TRUE)
```

Arguments:

`cluster` Cluster to show; if `TRUE` show the entire cluster (via `load_overall`), if `NULL` defaults to the value given when the client was created.

`nodes` Show the nodes when printing

Method `headnodes()`**:** Return a vector of known cluster headnodes. Typically `valid_clusters()` will be faster. This endpoint can be used as a relatively fast "ping" to check that you are logged in the client and server are talking properly.

Usage:

```
web_client$headnodes(forget = FALSE)
```

Arguments:

`forget` Logical, indicating we should re-fetch the value from the server where we have previously fetched it.

Method `r_versions()`**:** Return a vector of all available R versions

Usage:

```
web_client$r_versions()
```

Method `api_client()`**:** Returns the low-level API client for debugging

Usage:

```
web_client$api_client()
```

`web_login` *Test cluster login*

Description

Test cluster login

Usage

```
web_login(credentials = NULL)
```

Arguments

`credentials`	Your credentials

`worker_resource` *Specify worker resources*

Description

Specify resources for worker processes. If given, the values here will override those in `didehpc_config()`. See `vignette("workers")` for more details.

Usage

```
worker_resource(
  template = NULL,
  cores = NULL,
  wholenode = NULL,
  parallel = NULL
)
```

Arguments

`template`	A job template. On fi–dideclusthn this can be "GeneralNodes" or "8Core". On "fi–didemrchnb" this can be "GeneralNodes", "12Core", "16Core", "12and16Core", "20Core", "24Core", "32Core", or "MEM1024" (for nodes with 1Tb of RAM; we have three - two of which have 32 cores, and the other is the AMD epyc with 64). On the new "wpia-hn" cluster, you should currently use "AllNodes". See the main cluster documentation if you tweak these parameters, as you may not have permission to use all templates (and if you use one that you don't have permission for the job will fail). For training purposes there is also a "Training" template, but you will only need to use this when instructed to.
`cores`	The number of cores to request. If specified, then we will request this many cores from the windows queuer. If you request too many cores then your task will queue forever! 24 is the largest this can be on fi–dideclusthn. On fi–didemrchnb, the GeneralNodes template has mainly 20 cores or less, with a single 64 core node, and the 32Core template has 32 core nodes. On wpia-hn, all the nodes are 32 core. If `cores` is omitted then a single core is assumed, unless `wholenode` is TRUE.
`wholenode`	If TRUE, request exclusive access to whichever compute node is allocated to the job. Your code will have access to all the cores and memory on the node.
`parallel`	Should we set up the parallel cluster? Normally if more than one core is implied (via the `cores` or `wholenode` arguments, then a parallel cluster will be set up (see Details). If `parallel` is set to `FALSE` then this will not occur. This might be useful in cases where you want to manage your own job level parallelism (e.g. using OpenMP) or if you're just after the whole node for the memory).

Value

A list with class `worker_resource` which can be passed into `didehpc_config`

Index

cluster_load, 2

didehpc_config, 2, *16*
didehpc_config(), *7*, *15*
didehpc_config_global *(*didehpc_config*)*, 2

path_mapping, 5
path_mapping(), *3*
pkgdepends::new_pkg_installation_proposal(), *11*

queue_didehpc, 6
queue_didehpc_ *(*queue_didehpc*)*, 6
queuer::queue_base, *6–8*

R6::R6Class, *6*
rrq::rrq_controller, *9*

valid_clusters, 11
valid_clusters(), *3*, *15*

web_client, *7*, *8*, 11
web_login, 15
worker_resource, 15
worker_resource(), *4*